MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE CIÊNCIAS DA SAÚDE DE PORTO ALEGRE
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM TECNOLOGIAS DA INFORMAÇÃO E GESTÃO EM SAÚDE

Alan Baronio Menegotto

Uma Arquitetura de Aprendizado Profundo Multimodal para Auxílio no Diagnóstico de Hepatocarcinoma

Porto Alegre
2019

Alan Baronio Menegotto

Uma Arquitetura de Aprendizado Profundo Multimodal para Auxílio no Diagnóstico de Hepatocarcinoma

Dissertação apresentada ao programa acadêmico de Pós-graduação em Tecnologias da Informação e Gestão em Saúde da Universidade Federal de Ciências da Saúde de Porto Alegre como requisito parcial para obtenção do grau de mestre. Área de Concentração: Tecnologias da Informação Inteligente. Linha de Pesquisa: Sistemas Inteligentes e Aplicações na Saúde

Orientador: Prof. Dr. Silvio Cesar Cazella

Coorientadora: Profa. Dra. Carla Diniz Lopes Becker

Porto Alegre

2019

Catalogação na Publicação

Menegotto, Alan Baronio
Uma Arquitetura de Aprendizado Profundo Multimodal para Auxílio no Diagnóstico de Hepatocarcinoma / Alan Baronio Menegotto. -- 2019.
149 p. : 30 cm.

Dissertação (mestrado) -- Universidade Federal de Ciências da Saúde de Porto Alegre, Programa de Pós-Graduação em Tecnologias da Informação e Gestão em Saúde, 2019.

Orientador(a): Silvio Cesar Cazella ; coorientador(a): Carla Diniz Lopes Becker.

1. Aprendizado de Máquina Profundo Multimodal. 2. Hepatocarcinoma. 3. Auxílio-Diagnóstico Computadorizado. I. Título.

Sistema de Geração de Ficha Catalográfica da UFCSPA com os dados fornecidos pelo(a) autor(a).

Alan Baronio Menegotto

Uma Arquitetura de Aprendizado Profundo Multimodal para Auxílio no Diagnóstico de Hepatocarcinoma

Dissertação apresentada ao programa acadêmico de Pós-graduação em Tecnologias da Informação e Gestão em Saúde da Universidade Federal de Ciências da Saúde de Porto Alegre como requisito parcial para obtenção do grau de mestre. Área de Concentração: Tecnologias da Informação Inteligente. Linha de Pesquisa: Sistemas Inteligentes e Aplicações na Saúde.

Orientador: Prof. Dr. Silvio Cesar Cazella

Coorientadora: Profa. Dra. Carla Diniz Lopes Becker

Aprovada em: _____ de _____ de _____.

BANCA EXAMINADORA

Prof. Dr. Altamiro Susin
UFRGS

Prof. Dra. Thatiane Alves Pianoschi
UFCSPA

Prof. Dr. Luciano Blomblerg
UFCSPA

Agradecimentos

Agradeço inicialmente o Prof. Dr. Silvio Cesar Cazella e a Profa. Dra. Carla Diniz Lopes Becker pela oportunidade de orientação, pelo tempo despendido e pelo conhecimento repassado. Agradeço também pela paciência e amor incondicional de minha esposa Clarissa Capp, meu filho Arthur Capp Menegotto, meu pai Hilario Menegotto, minha mãe Ana Mari Baronio Menegotto e meu irmão Guilherme Baronio Menegotto. Por fim agradeço o pessoal da CGTIC do Hospital de Clínicas de Porto Alegre que concedeu a liberação para as aulas, os congressos, a docência e todas as demais atividades relacionadas ao mestrado.

Resumo

Introdução: O aprendizado de máquina profundo é uma técnica de inteligência artificial que vem sendo empregada com sucesso na construção de sistemas de auxílio-diagnóstico computadorizado. O auxílio-diagnóstico computadorizado de hepatocarcinoma, doença que não possui sintomas patognomônicos, comumente é realizado utilizando exclusivamente atributos clínicos, atributos genéticos ou exames radiológicos. A fusão destas múltiplas modalidades de dados poderia melhorar o desempenho desse tipo de sistema. Este estudo descreve as etapas de criação e avaliação de um algoritmo de aprendizado de máquina profundo para auxílio-diagnóstico computadorizado de hepatocarcinoma que combina exames laboratoriais, atributos clínicos, antropométricos e sociodemográficos com exames de imagem. **Objetivo:** O objetivo geral é desenvolver uma arquitetura de auxílio-diagnóstico computadorizado de hepatocarcinoma baseado em um algoritmo de aprendizado de máquina profundo supervisionado associando exames laboratoriais, atributos clínicos, antropométricos, sociodemográficos com exames de imagem. **Métodos:** A natureza das tarefas envolvidas indica que o tipo de pesquisa realizado é primariamente experimental. O desenvolvimento de um algoritmo de aprendizado de máquina profundo para auxílio-diagnóstico computadorizado de hepatocarcinoma é composto por tarefas como obtenção e preparação das bases de dados utilizadas para treinamento e testes, definição das arquiteturas de redes convolucionais unimodais e multimodais utilizadas, execução de experimentos e avaliação dos resultados. **Resultados:** O desempenho obtido pela arquitetura desenvolvida utilizando uma rede Xception modificada, aplicando pré-processamento nas imagens e combinando estas múltiplas modalidades de dados através de fusão de dados intermediária foi: acurácia = 86.9%, precisão = 89.6%, revocação = 86.9% e F-Score = 86.7%. O ganho de precisão obtido com a abordagem multimodal, quando comparada a abordagem unimodal com a rede convolucional Xception foi de 28.8% nos experimentos de verificação. **Conclusão:** Os experimentos realizados sugerem uma superioridade de desempenho ao utilizar abordagens multimodais com fusão de dados intermediária e imagens pré-processadas para sistemas de auxílio-diagnóstico de hepatocarcinoma implementados com aprendizado de máquina profundo. Além disso, nesta base de dados o desempenho da arquitetura proposta é superior ao desempenho médio de especialistas que utilizam exclusivamente imagens de tomografia computadorizada no diagnóstico. Entretanto, o viés intrínseco da aplicação de técnicas de aprendizado de máquina profundo demanda que o algoritmo desenvolvido seja testado em novas bases de dados antes da aplicação no dia-a-dia em rotinas assistenciais.

Palavras-chaves: aprendizado de máquina profundo multimodal. hepatocarcinoma. auxílio-diagnóstico computadorizado.

Abstract

Introduction: Deep learning is a promising artificial intelligence technique to develop computer-aided diagnosis systems. Computer-aided diagnosis of hepatocellular carcinoma, a disease without pathognomonic symptoms, is often performed using exclusively clinical attributes, genetic attributes or imaging. Fusing these multiple data modalities may increase the performance of computer-aided diagnosis systems for hepatocarcinoma. This study describes the development and evaluation of a multimodal deep learning algorithm for computer-aided diagnosis of hepatocarcinoma combining laboratory tests, clinical, anthropometric and sociodemographic attributes with imaging exams. **Objective:** The overall objective is to develop a computer-aided hepatocarcinoma diagnosis architecture based on a supervised deep learning algorithm that fuses laboratory tests, clinical, anthropometric and sociodemographic attributes with imaging. **Methods:** The nature of these tasks indicates that the type of this research is primarily experimental. Create an algorithm for computer-aided diagnosis of hepatocarcinoma using multimodal deep learning approaches consists of tasks such as search, download and preprocess databases for training and testing, develop unimodal and multimodal deep neural network architectures, execute experiments and evaluate results. **Results** The performance obtained by the developed architecture using a fine-tuned Xception convolutional neural network, applying imaging preprocessing and joining multiple data modalities through intermediate fusion was: accuracy = 86.9%, precision = 89.6%, recall = 86.9% and F-Score = 86.7%. The precision gain achieved with the multimodal approach in this convolutional neural network was 28.8% during the verification experiments. **Conclusion:** Evidence found in experiments suggests that using multimodal approaches with intermediate data fusion and preprocessed images results in superior performance for computer-aided diagnosis of hepatocarcinoma systems developed with deep learning techniques. Moreover, the computer-aided diagnosis performance of the proposed architecture in this database is greater than the average specialist's performance reported in the literature who exclusively use computed tomography images in the diagnostic. However, the intrinsic bias of the application of deep learning techniques demands that the developed algorithm should be tested with new samples before application in day-to-day healthcare routine.

Key-words: multimodal deep learning. hepatocarcinoma. computer-aided diagnosis.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AASLD	*American Association for the Study of Liver Diseases*
AE	*AutoEncoder*
AMI	*Amazon Machine Image*
AUC	*Area Under the Curve*
AUROC	*Area Under the Receiver Operating Characteristics*
BLCL	*Barcelona Clinic Liver Cancer*
CLAHE	*Contrast Limited Adaptive Histogram Equalization*
CPTAC-PDA	*Clinical Proteomic Tumor Analysis Consortium Pancreatic Ductal Adenocarcinoma*
CPU	*Central Processing Unit*
CSV	*Comma-separated Value*
DAE	*Denoising Autoencoder*
DICOM	*Digital Imaging and Communications in Medicine*
EASL	*European Association for the Study of the Liver*
EBS	*Elastic Block Store*
ECU	*EC2 Compute Unit*
EEG	Eletroencefalografia
FN	Falso Negativo
FP	Falso Positivo
GHz	GigaHertz
GPU	*Graphics Processing Unit*
HCC	Hepatocarcinoma
ID	Identificador
ILSVRC	*ImageNet Large Scale Visual Recognition Challenge*

INR	*International Normalized Ratio*
KB	*Kilobyte*
LI-RADS	*Liver Imaging Reporting e Data System*
LSTM	*Long Short-Term Memory*
MS-COCO	*Microsoft Common Objects in Context*
NHGRI	*National Human Genome Research Institute*
PNG	*Portable Network Graphics*
RAM	*Random Access Memory*
REST	*Representational State Transfer*
RGB	*Red, Green and Blue*
ROC	*Receiver Operating Characteristic*
SAE	*Stacked Autoencoder*
sDAE	*Stacked Denoising Autoencoder*
SGD	*Stochastic Gradient Descent*
SSD	*Solid-State Drive*
TCGA	*The Cancer Genome Atlas*
TCGA-KIRP	*The Cancer Genome Atlas Cervical Kidney Renal Papillary Cell Carcinoma*
TCGA-LIHC	*The Cancer Genome Atlas Liver Hepatocellular Carcinoma*
TCGA-STAD	*The Cancer Genome Atlas Stomach Adenocarcinoma*
TCIA	*The Cancer Imaging Archive*
UUID	*Universally Unique Identifier*
VAE	*Variational Autoencoder*
vCPU	*Virtual Central Processing Unit*
VN	Verdadeiros Negativos
VP	Verdadeiros Positivos
VPP	Valor Preditivo Positivo

Sumário

1 Introdução

Em 2018 aproximadamente 18 milhões de pessoas em todo mundo foram acometidas por algum tipo de câncer e, destas, 9.6 milhões de pessoas vieram a óbito por causa da doença (International Agency for Research on Cancer, 2018a). Os tipos de câncer com maior taxa de mortalidade em 2018 foram: pulmão (1.761.007 mortes), estômago (782.685 mortes), fígado (781.631 mortes), mama (626.679 mortes) e colorretal (551.269 mortes) (International Agency for Research on Cancer, 2018b). Apesar dos recentes avanços no diagnóstico e tratamento do câncer, estimativas de diferentes institutos apontam para um incremento substancial nos casos de câncer nas próximas décadas, causado principalmente por fatores como aumento e envelhecimento da população, rotinas estressantes e contexto social (International Agency for Research on Cancer, 2018a) (ZHENG et al., 2018) (Cancer Research UK, 2018).

O aumento de casos de câncer causará uma sobrecarga no sistema assistencial (VALERY et al., 2018) (ELLISON et al., 2018) e uma das alternativas existentes para mitigar este problema é aumentar a eficiência do profissional assistencial através da utilização de sistemas de auxílio-diagnóstico computadorizado (do inglês, Computer-Aided Diagnosis - CAD) específicos para o câncer. O auxílio-diagnóstico computadorizado é uma área de pesquisa que emprega diferentes técnicas como mineração de dados (WANG et al., 2017), pesquisa operacional (PRICE et al., 2016) e inteligência artificial (ESTEVA et al., 2017) para o desenvolvimento de ferramentas que apoiam o profissional assistencial durante seu processo de trabalho.

A popularização de estudos que utilizam algoritmos de aprendizado de máquina (do inglês, *machine learning*) e possuem desempenho equiparável a especialistas médicos no auxílio-diagnóstico de câncer de pele, mama e pulmão (ESTEVA et al., 2017) (SUN; ZHENG; QIAN, 2016) (KALLENBERG et al., 2016) começou a ocorrer a partir de 2012, quando ocorreram significativos avanços no desempenho computacional do *hardware* disponível e de técnicas de inteligência artificial que resultaram no aumento da precisão obtida pelos computadores em tarefas como classificação de imagens, reconhecimento de fala e processamento de linguagem natural (LECUN; BENGIO; HINTON, 2015).

O câncer de fígado, também conhecido como carcinoma hepatocelular ou hepatocarcinoma é uma doença silenciosa que não possui sintomas patognomônicos (ATTWA; EL-ETREBY, 2015). A falta de sintomas faz com que o diagnóstico seja frequentemente realizado em fases tardias da doença, quando já não existem mais tratamentos efetivos. A taxa de sobrevida depende do estadiamento do carcinoma no momento do diagnóstico e costuma ser extremamente curta na maioria dos casos (TSAI et al., 2018).

O protocolo de diagnóstico de hepatocarcinoma recomendado por diferentes instituições de referência em doenças hepáticas sugere a utilização de mais de um tipo de exame de imagem quando o tamanho dos nódulos investigados possuem mais de um centímetro (GALLE et al., 2018) (MARRERO et al., 2018). Não foram encontrados na revisão de literatura trabalhos que implementem o fluxo de diagnóstico recomendado de forma estrita, ou seja, utilizando múltiplas modalidades de exames de imagem dependendo do tamanho dos nódulos e do contexto do paciente.

A principal questão que norteia esta pesquisa é: 'A utilização de resultados de exames laboratoriais como Tempo de Protrombina, Albumina, Bilirrubinas Totais e Creatinina em conjunto com biomarcadores tumorais como Alfafetoproteína, dados antropométricos, clínicos e sociodemográficos associados a exames de imagem pode aumentar a precisão de um algoritmo de auxílio-diagnóstico computadorizado de hepatocarcinoma?'

Ao que tudo indica, a utilização de múltiplas modalidades de dados em sistemas de auxílio-diagnóstico computadorizado de hepatocarcinoma pode significar uma melhora de desempenho neste tipo de sistema, mas não foi encontrada na literatura nenhuma evidência que comprove esta hipótese. A presente pesquisa tem por objetivo explorar esta lacuna da literatura e fornecer subsídios que possam confirmar a superioridade de desempenho de arquiteturas multimodais neste tipo de sistema.

O projeto referente a esta dissertação está registrado na Comissão de Pesquisa da Universidade Federal de Ciências da Saúde de Porto Alegre sob o número 064/2018.

1.1 Justificativa

A ausência de sintomas patognomônicos dificulta o processo diagnóstico de pacientes acometidos por hepatocarcinoma. A utilização de dados armazenados em sistemas de prontuário eletrônico pode fornecer um panorama global do estado de saúde do paciente e agilizar o processo diagnóstico. Apesar da disponibilidade destes dados em instituições assistenciais de médio e grande porte, a quantidade de sistemas de auxílio-diagnóstico de hepatocarcinoma que utilizam múltiplas modalidades de dados como insumo para detecção da doença alvo é inferior a quantidade de trabalhos encontrados na literatura que processam uma única modalidade de dado para auxílio-diagnóstico de hepatocarcinoma. A revisão sistemática descrita no Capítulo 3 foi realizada no começo deste projeto e buscou identificar as técnicas de inteligência artificial aplicadas para auxílio-diagnóstico de hepatocarcinoma. Foram encontrados 46 artigos e destes, somente 6 utilizam mais de uma modalidade de dados: (ALI et al., 2017) e (ABAJIAN et al., 2018) combinam exames de imagem e atributos clínicos; (WANG et al., 2018), (CHEN et al., 2017), (RAMKUMAR et al., 2017) e (MIOTTO; LI; DUDLEY, 2016) combinam informações textuais (ex: laudo radiológico) e informações do prontuário eletrônico do paciente.

Não foram encontrados na revisão realizada trabalhos que utilizem dados laboratoriais, antropométricos e sociodemográficos associados a exames de imagem como insumo de entrada em sistemas de auxílio-diagnóstico computadorizado de hepatocarcinoma e a ausência de evidências contrárias serve como ponto de partida para o desenvolvimento deste projeto.

A aplicação de redes neurais convolucionais profundas para extração e classificação de padrões desconhecidos ou invisíveis a olho nu possibilita a construção de sistemas de apoio à decisão que efetivamente melhoram o processo diagnóstico e aumentam as probabilidades de cura do paciente em casos positivos da doença (LITJENS et al., 2017).

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste projeto é desenvolver uma arquitetura para auxílio-diagnóstico computadorizado de hepatocarcinoma baseada em um algoritmo de aprendizado de máquina profundo supervisionado que utiliza como entrada dados laboratoriais, atributos clínicos, antropométricos e sociodemográficos associados a exames de imagem.

1.2.2 Objetivos Específicos

1. Realizar uma revisão sistemática para levantar o estado da arte e as técnicas comumente empregadas no auxílio-diagnóstico computadorizado de hepatocarcinoma;
2. Obter de bases de dados públicas os resultados dos exames de imagem correlacionados com os resultados de exames laboratoriais, atributos sociodemográficos, clínicos e antropométricos;
3. Preparar a base de dados para utilização no processo de treinamento, validação e testes: pré-processamento, equalização, divisão das bases;
4. Projetar a algoritmo de aprendizado de máquina profundo multimodal que será utilizada para auxílio-diagnóstico de hepatocarcinoma;
5. Implementar a arquitetura de auxílio-diagnóstico computadorizado de hepatocarcinoma;
6. Realizar experimentos com o uso da arquitetura desenvolvida e coletar o desempenho utilizando métricas de classificação;
7. Avaliar o desempenho da arquitetura desenvolvida.

1.3 Contribuições Esperadas

O desempenho de um algoritmo de aprendizado de máquina profundo para problemas de classificação é calculado utilizando métricas que avaliam a quantidade de acertos e erros do modelo como por exemplo acurácia, precisão e revocação,. Maiores detalhes sobre estas métricas serão abordadas no decorrer desta dissertação.

A resposta da questão de pesquisa permitirá identificar se a aplicação de múltiplas modalidades de dados como entrada de um algoritmo de aprendizado de máquina profundo pode aumentar a precisão de sistemas de auxílio-diagnóstico de hepatocarcinoma construídos com essa técnica e também quantificar o ganho de acurácia e precisão da aplicação das abordagens multimodais em sistemas construídos comumente com entradas unimodais.

Caso comprovado, o aumento de precisão pode ser estendido para sistemas de auxílio-diagnóstico de outras doenças cuja detecção também pode ser realizada com múltiplas modalidades de dados. Desta forma, a utilização desses sistemas pode resultar num tratamento mais rápido e assertivo, facilitando o trabalho assistencial e aumentando a probabilidade de cura do paciente (TSAI et al., 2018).

1.4 Estrutura da Dissertação

O Capítulo 2 apresenta uma visão geral sobre os conceitos pertinentes para compreensão das atividades descritas nesta dissertação. O Capítulo 3 descreve as revisões de literatura realizadas durante o projeto e sumariza os trabalhos relacionados encontrados. O Capítulo 4 detalha os passos percorridos para criação e experimentação do algoritmo de auxílio-diagnóstico de hepatocarcinoma utilizando aprendizado de máquina profundo. O Capítulo 5 descreve os resultados obtidos com os experimentos realizados para comparação entre as abordagens unimodais e multimodais e o Capítulo 6 apresenta uma discussão dos resultados dos experimentos, compara o resultado obtido com trabalhos relacionados e apresenta as limitações e possíveis pontos de melhora que podem ser trabalhados em projetos futuros.

2 Referencial Teórico

2.1 Aprendizado de Máquina Profundo

2.1.1 Aprendizado de Máquina

O aprendizado de máquina é uma subárea da inteligência artificial que desenvolve técnicas para criação de programas de computador capazes de aprender a realizar uma tarefa T pela experiência E e cujo desempenho, medido em P, melhora conforme aumenta a experiência E na execução da tarefa T (MITCHELL, 1997). São exemplos deste tipo de programa: 1) programa capaz de projetar temperaturas futuras a partir do processamento histórico de variáveis climáticas; 2) programa capaz de detectar diabetes a partir do processamento de uma base histórica de atributos clínicos de pacientes. Dentre os algoritmos de aprendizado de máquina utilizados para criação deste tipo de aplicação pode-se citar: Florestas Aleatórias, Redes Neurais Artificiais e Máquinas de Vetores de Suporte.

Conforme (RUSSELL; NORVIG; DAVIS, 2010), existem três formas de aprendizado de máquina: 1) Supervisionado: quando todos os dados de treinamento são rotulados; 2) Semi-supervisionado: quando uma parcela do conjunto de dados de treinamento é rotulada; 3) Não Supervisionado: quando nenhum dado de treinamento é previamente rotulado. Existem autores como (CHOLLET, 2017a) que consideram subtipos dessas formas de aprendizado de máquina como formas distintas desta técnica: 1) Aprendizado Auto-Supervisionado: o algoritmo utiliza métodos para supervisionar seu próprio progresso; 2) Aprendizado por Reforço: um agente externo não-humano supervisiona o aprendizado e realiza ações de recompensa ou penalização para guiar o processo de aprendizagem.

A denominação do tipo de problema de aprendizado de máquina é feita conforme o objetivo desejado: em problemas de *regressão* o objetivo é predizer valores numéricos contínuos e em problemas de *classificação* o objetivo é atribuir cada vetor de entrada a um dentre um número finito de categorias discretas (BISHOP, 2006). Calcular a sobrevida em anos de um paciente diagnosticado com hepatocarcinoma baseado em informações históricas e contextuais utilizando aprendizado de máquina é um exemplo de problema de regressão. Detectar uma possível doença a partir de atributos clínicos com aprendizado de máquina é um exemplo de problema de classificação. Atualmente a utilização do aprendizado de máquina não se resume a estes dois tipos de problemas. A evolução da técnica expandiu a gama de categorias de problema resolvidos. Como exemplo pode-se citar: transcrição de múltiplas modalidades de dados em texto, síntese e amostragem de sinais, imputação de dados faltantes e remoção de ruídos (GOODFELLOW; BENGIO; COURVILLE, 2016a).

A construção de um programa utilizando técnicas de aprendizado de máquina não é uma tarefa exclusivamente de programação de computadores. Além da correta escolha, construção e parametrização do algoritmo, é necessário utilizar uma base de dados de treinamento e validação alinhada com os objetivos desejados. Caso contrário, pode-se incorrer em erros como: 1) Erro de Ajuste: acontece quando o modelo estatístico criado pelo algoritmo oscila e não adere ao modelo que representa o padrão desejado. Este comportamento pode acontecer quando o algoritmo não consegue extrair o conhecimento devido à quantidade inadequada de atributos utilizados, quando o conjunto de dados utilizado para treinamento possui baixa qualidade ou quando a quantidade de dados disponíveis para treinamento não representa todo o conhecimento; 2) Erro de Generalização: ocorre quando o modelo estatístico criado pelo algoritmo adapta-se ao conjunto de dados de treinamento porém não consegue bom desempenho em novos conjuntos de dados (BISHOP, 2006).

Uma das possíveis soluções para evitar os problemas supracitados é utilizar estratégias de particionamento e rotação de bases de dados. Na estratégia de particionamento denominada *Hold-Out*, os dados são particionados aleatoriamente em dois conjuntos distintos sem sobreposição: um é utilizado no treinamento e outro na validação do modelo. Quando o conjunto de dados é pequeno, estratégias mais elaboradas podem ser utilizadas. A estratégia conhecida como validação cruzada particiona os dados na proporção de conjuntos $\frac{(S-1)}{S}$ para treinamento e validação, e utiliza estes múltiplos conjuntos para garantir a qualidade do modelo estatístico criado. Quando $S = N$, ou seja, o número de partições do conjunto é igual a quantidade total de dados disponíveis, a estratégia recebe o nome de *Leave-one-out*. Quando $S \neq N$ a estratégia recebe o nome de *k-fold cross-validation*. Nesse caso, k representa a quantidade de partições que podem ser iteradas de forma sequencial ou aleatória (BISHOP, 2006).

A avaliação de um algoritmo de aprendizado de máquina é realizada medindo a capacidade de generalização do algoritmo através de métricas de desempenho específicas para o tipo de problema que está sendo tratado.

O desempenho de um algoritmo de aprendizado de máquina em problemas de regressão pode ser medido com uma das métricas a seguir: 1) Erro Médio Quadrático: calculado através da média do somatório da diferença entre a regressão e o valor real; 2) Raiz do Erro Médio Quadrático: calculada através da raiz quadrada do Erro Médio Quadrático; 3) Coeficiente de Determinação: calculado pela razão da soma dos quadrados da regressão e a soma dos quadrados totais (KUHN; JOHNSON, 2013a).

O desempenho de um algoritmo de aprendizado de máquina em problemas de classificação pode ser medido utilizando métodos estatísticos calculados a partir de métricas extraídas durante o processo de testes. As métricas que servem de base para os métodos estatísticos são: quantidade de verdadeiros positivos, quantidade de verdadeiros negativos,

quantidade de falsos positivos e quantidade de falsos negativos. Essas métricas costumam ser dispostas em uma matriz quadrada conhecida como matriz de confusão, cuja ordem é dependente da quantidade de classes alvo.

Os métodos estatísticos utilizados para medir o desempenho de algoritmos de aprendizado de máquina em problemas de classificação são: 1) Acurácia, Precisão, Revocação e Especificidade: calculadas a partir da razão das métricas quantitativas em relação ao total de classificações realizadas; 2) *Logarithmic Loss*: também conhecida como *Log Loss*, é utilizada em problemas de classificação multi-categóricos. Cada categoria possui uma probabilidade e a avaliação ocorre através de um somatório logarítmico que penaliza predições erradas (falsos negativos). Quanto maior o valor, pior a precisão do classificador; 3) Curva ROC: gráfico que ilustra a relação entre a taxa de verdadeiros positivos e a taxa de verdadeiros negativos; 4) Curva AUC: gráfico que ilustra a relação entre a revocação e a especificidade da classificação. Além do gráfico, a área da curva pode ser calculada através de integração (KUHN; JOHNSON, 2013b) ou métodos numéricos (THARWAT, 2018) e também pode ser utilizada como medida de desempenho; 5) *F1 Score*: também conhecida como *F-Measure*, representa a média harmônica entre a precisão e a revocação do classificador. Quanto maior o valor, mais robusta e precisa é a classificação (SOKOLOVA; JAPKOWICZ; SZPAKOWICZ, 2006).

2.1.2 Aprendizado de Máquina Profundo

O aprendizado de máquina profundo é uma subárea do aprendizado de máquina que coloca ênfase no aprendizado através de sucessivas camadas. Cada camada é responsável pelo aprendizado de parte do padrão e esta é a grande diferença desta abordagem em relação ao aprendizado de máquina tradicional (CHOLLET, 2017b). Este subtipo de aprendizado de máquina surgiu a partir da criação de técnicas que permitem o treinamento de arquiteturas profundas (HINTON; OSINDERO; TEH, 2006) e a partir da evolução do *hardware* utilizado (LITJENS et al., 2017). São exemplos de programas construídos com aprendizado de máquina profundo: 1) um programa capaz de identificar nódulos cancerígenos em exames radiológicos a partir do processamento de uma base de imagens previamente classificada; 2) um programa capaz de identificar em tempo real placas de trânsito a partir do processamento de uma base de vídeos previamente classificada.

A escolha de um algoritmo de aprendizado de máquina profundo é dependente de fatores como tipo do problema e natureza dos dados de entrada. As subseções 2.1.2.1, 2.1.2.2, 2.1.2.3 e 2.1.2.4 descrevem algoritmos e arquiteturas de aprendizado de máquina profundo comumente encontradas na literatura para construção de sistemas de auxílio-diagnóstico.

2.1.2.1 Redes Neurais Profundas

Rede neural artificial é uma estrutura computacional massivamente paralelizada formada por simples unidades de processamento denominadas neurônios artificiais. Esta estrutura, também conhecida como rede conexionista, é fortemente inspirada no modelo do cérebro humano para adquirir, processar e armazenar conhecimento. Cada neurônio possui um peso associado e em sua saída existe uma função de ativação. A função de ativação é a representação computacional das sinapses cerebrais e os neurônios funcionam como interruptores elétricos estimulados ou inibidos conforme as entradas recebidas (HAYKIN; ELEKTROINGENIEUR, 2009).

A arquitetura de uma rede neural artificial é composta por pelo menos três camadas de neurônios interligadas, denominadas 'camada de entrada', 'intermediária' e 'de saída'. A camada intermediária é responsável por processar e armazenar o conhecimento nos pesos de cada neurônio. Os dados chegam a camada intermediária através da camada de entrada e o resultado do processamento é obtido na camada de saída. A quantidade de neurônios que compõe cada camada depende do contexto e do problema que está sendo tratado (RUSSELL; NORVIG, 2016).

Uma rede neural profunda é uma rede neural artificial com n camadas intermediárias, sendo $n \geq 1$ e a profundidade da rede é medida pela quantidade de camadas intermediárias, também conhecidas como camadas ocultas (GOODFELLOW; BENGIO; COURVILLE, 2016b). A Figura 1 ilustra um exemplo desse tipo de arquitetura.

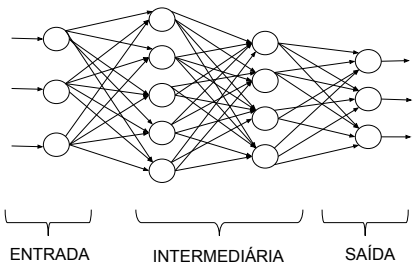


Figura 1 – Exemplo de Rede Neural Profunda. Fonte: Do Autor.

2.1.2.2 Redes Neurais Convolucionais

Redes neurais convolucionais são um tipo de rede neural artificial profunda para aprendizado supervisionado que utiliza a operação de convolução matemática sobre tensores para extração de características em pelo menos uma das camadas da rede (GOODFELLOW; BENGIO; COURVILLE, 2016c). Redes neurais convolucionais suportam tensores de múltiplas dimensões como entrada: um tensor 1D pode representar uma entrada sequencial, um tensor 2D pode representar uma imagem e um tensor 3D pode representar um vídeo.

As camadas intermediárias de uma rede neural convolucional são responsáveis pela

extração de padrões dos tensores. Os padrões extraídos são utilizados como entrada para a camada densa, uma rede neural artificial geralmente profunda e totalmente conectada cuja saída representa o resultado da rede para determinada entrada. A Figura 2 ilustra a arquitetura da rede neural convolucional conhecida como LeNet, utilizada em larga escala no final do século XX em bancos e instituições postais dos EUA para reconhecimento ótico de caracteres (LECUN et al., 1998).

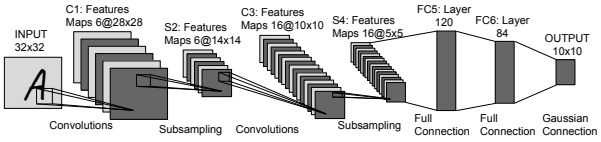


Figura 2 – Arquitetura Lenet-5. Fonte: (LECUN et al., 1998).

A utilização de redes neurais convolucionais para reconhecimento de imagens popularizou-se após 2012 quando a rede neural convolucional profunda denominada AlexNET (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) conseguiu uma taxa de erro de 16% no desafio de classificação de imagens conhecido como *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC). Neste desafio os algoritmos competem na realização de tarefas como classificação e localização de objetos em imagens. O banco de imagens utilizado nessa competição é o *ImageNet*, composto por 14 milhões de imagens de 1.000 categorias diferentes (RUSSAKOVSKY et al., 2015).

O resultado da rede AlexNET representava metade da taxa de erro do estado da arte neste tipo de problema na época e alavancou a popularização deste tipo de rede. A partir de 2012, diversas redes neurais convolucionais começaram a ser criadas e treinadas em bases de imagem como ImageNet, CIFAR-10 e MS-COCO por grupos de pesquisa, empresas públicas e empresas privadas. Em 2016, a taxa de erro do melhor algoritmo foi de 2.99% e ultrapassou o resultado obtido por seres humanos, que é de 5% (RUSSAKOVSKY et al., 2015). As redes neurais convolucionais que se destacaram no ILSRVC entre 2012 e 2017 (último ano do concurso) podem ser vistas na Tabela 1.

Tabela 1 – Destaques do ILSRVC entre 2012 e 2017.

Ano	**Nome**	**Referência**	**Taxa de Erro (Top-5)**
2012	AlexNET	(KRIZHEVSKY; SUTSKEVER; HINTON, 2012)	16,4%
2013	ZFNet	(ZEILER; FERGUS, 2014)	11,7%
2014	Inception	(SZEGEDY et al., 2015)	6,7%
2014	VGG	(SIMONYAN; ZISSERMAN, 2015)	7,3%
2015	ResNET	(HE et al., 2016a)	3,57%
2016	Trimps-Soushen	-*	2,99%
2016	ResNeXt	(XIE et al., 2017)	3,03%
2017	SENet	(HU; SHEN; SUN, 2018)	2,25%

* Não foi escrito artigo pelo Ministério da Segurança Pública da China

A construção e o treinamento de redes neurais profundas tornou-se viável devido ao poder de processamento fornecido por CPUs e GPUs de alto desempenho e a imensa quantidade de dados produzida e armazenada pela sociedade digital. Além disso, a criação de ferramentas que abstraem conceitos matemáticos complexos e fornecem funções para composição das redes neurais convolucionais também impulsionou a disseminação deste tipo de rede para resolver problemas que utilizam imagem como insumo de entrada. Dentre estas ferramentas pode-se citar: Tensorflow (ABADI et al., 2016), Keras (CHOLLET, 2015), Caffe (JIA et al., 2014) e PyTorch (KETKAR, 2017). Além da facilidade para criação de programas utilizando aprendizado de máquina profundo, essas ferramentas vêm com implementações de redes profundas do estado da arte que podem ser utilizadas de duas formas:

1. pesos e/ou hiperparâmetros do pré-treinamento são reutilizados sem nenhum ajuste. Geralmente este tipo de abordagem é realizado quando o pré-treinamento foi realizado em classes de imagem do mesmo domínio da aplicação que está sendo construída, quando a rede já consegue um bom desempenho sobre um novo domínio de imagem sem nenhum ajuste ou quando a rede utiliza técnicas para aprender novos domínios sem esquecer os antigos (LI; HOIEM, 2017). Como exemplo, pode-se citar a utilização da técnica de aprendizado de representação progressivo para detecção de objetos salientes em imagens sem a necessidade de pré-treinamento (CHEN et al., 2016).

2. pesos do pré-treinamento são lidos e a rede passa por ciclos de ajuste e treinamento específicos. Esta é a abordagem realizada para adaptar uma rede pré-treinada a um novo domínio de imagens e costuma ser a estratégia com melhor custo-benefício quando se aplica redes neurais convolucionais profundas na construção de algoritmos de auxílio-diagnóstico computadorizado que utilizam exames de imagem como insumo de entrada. Como exemplo, pode-se citar o auxílio-diagnóstico de câncer de pele

(ESTEVA et al., 2017), auxílio-diagnóstico de câncer de pulmão (ALZUBAIDI et al., 2017) e auxílio-diagnóstico de câncer cerebral (KAMNITSAS et al., 2017).

2.1.2.3 Redes de Crença Profunda

Rede de Crença Profunda foi a primeira arquitetura de rede neural denominada como profunda (BENGIO et al., 2007). As redes de crença profunda funcionam de forma não supervisionada, ou seja, os dados de entrada não são rotulados e a própria rede é capaz de classificá-los através de uma estratégia generativa cujo objetivo é obter uma representação compacta dos padrões que formam o conjunto de dados de entrada (HINTON; OSINDERO; TEH, 2006).

As camadas de uma rede de crença profunda são compostas por Máquinas de Boltzmann Recorrentes (do inglês, *Recurrent Boltzmann Machines*) (SMOLENSKY, 1986) empilhadas. As camadas comunicam-se entre si mas não existe comunicação lateral entre os neurônios ocultos de uma camada.

Redes de crença profunda são utilizadas com êxito em aplicações de processamento de sinais como processamento de eletroencefalograma (EEG) (MOVAHEDI; COYLE; SEJDIC, 2018). A Figura 3 ilustra uma rede de crença profunda.

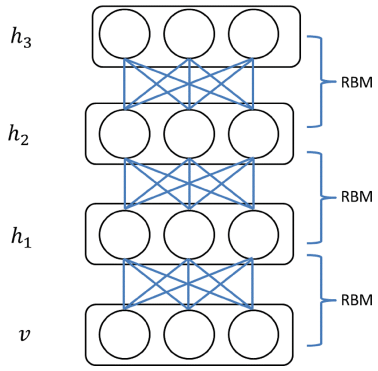


Figura 3 – Exemplo de Rede de Crença Profunda. Fonte: (MOVAHEDI; COYLE; SEJDIĆ, 2017)

2.1.2.4 AutoEncoders

AutoEncoders (AE) são redes neurais artificiais generativas não supervisionadas. O objetivo de um AutoEncoder é obter uma cópia fiel da entrada em sua saída. Para isto, a rede é organizada em duas estruturas distintas: o gerador (*encoder*) e o juiz (*decoder*).

Os AutoEncoders funcionam de forma iterativa: o gerador cria versões da entrada até que o juiz, baseado em uma 'verdade absoluta', aceita a versão gerada. São acrescidas restrições para que a rede seja forçada a priorizar as principais características da entrada para reprodução, de forma a evitar uma cópia literal da mesma. Assim, obtém-se apenas o espaço latente da entrada, ou seja, as principais características da sua representação de

forma compactada (GOODFELLOW; BENGIO; COURVILLE, 2016d). A figura 4 ilustra um exemplo de AutoEncoder:

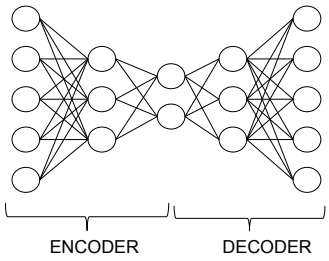


Figura 4 – Exemplo de AutoEncoder. Fonte: Do Autor.

Existem diferentes tipos de redes AutoEncoder, denominadas conforme seu objetivo e funcionamento:

- *Denoising AutoEncoders* (DAE) modificam aleatoriamente a entrada para evitar uma cópia literal, forçando a rede a reconstruir as partes modificadas (VINCENT et al., 2010). Podem ser utilizados de forma empilhada, quando recebem o nome de *Stacked Denoising AutoEncoders* (sDAE).

- *Sparse AutoEncoders* (SAE) possuem um número maior de neurônios nas camadas intermediárias. Estes neurônios são ativados de forma aleatória, evitando assim a cópia literal da entrada (MAKHZANI; FREY, 2013).

- *Variational AutoEncoders* (VAE) mapeiam os componentes da entrada em um espaço latente contínuo através de representações matemáticas como média e desvio padrão da distribuição. A função de desempenho da rede é medida através da diferença entre as distribuições, que indica neste contexto qual a quantidade de informação perdida durante o processo generativo (KINGMA; WELLING, 2013).

2.1.3 Aprendizado de Máquina Multimodal

A utilização de dados obtidos de múltiplos sensores como entrada de algoritmos de aprendizado de máquina é denominada aprendizado de máquina multimodal (LAHAT; ADALI; JUTTEN, 2015). As múltiplas entradas de dados podem ser compostas de dados da mesma natureza, como por exemplo, imagens de tomografia computadorizada e ressonância magnéticas combinadas para auxílio-diagnóstico de gliomas (LI et al., 2017) ou naturezas distintas, como por exemplo fotografias combinadas com a atributos clínicos para auxílio-diagnóstico de melanomas (KAWAHARA et al., 2019). O emprego de múltiplas modalidades de entrada em algoritmos de aprendizado de máquina profundo recebe o nome de aprendizado de máquina profundo multimodal.

A junção das múltiplas modalidades de dados é realizada através de técnicas de fusão de dados. A tipificação da fusão é feita conforme o local onde as modalidades de dados são combinadas (LAHAT; ADALI; JUTTEN, 2015) (RAMACHANDRAM; TAYLOR, 2017):

- Fusão Antecipada: do inglês, *Early Fusion*, ocorre quando os dados são combinados antes da camada de entrada do algoritmo de aprendizado de máquina. Neste tipo de fusão de dados o algoritmo de aprendizado de máquina desconhece a existência das múltiplas modalidades, visto que a entrada do algoritmo de aprendizado de máquina aceita somente uma única forma de representação de dados e continua sendo unimodal. O prognóstico de resposta à quimioradioterapia em casos de câncer retal utilizando exames de imagens e informações do prontuário eletrônico do paciente descrito em (BIBAULT et al., 2018) é um exemplo de utilização deste tipo de fusão de dados.

- Fusão Intermediária: do inglês, *Intermediate Fusion*, também conhecida como *Joint Fusion* ou *Feature-Level Fusion*, ocorre nas camadas intermediárias, quando todas as modalidades de dados estão na mesma forma de representação. A utilização deste tipo de fusão de dados exige que a rede possua pelo menos uma entrada por modalidade de dados. A arquitetura de aprendizado de máquina profundo multimodal para auxílio-diagnóstico de câncer oral descrita em (SONG et al., 2018) é um exemplo de aplicação deste tipo de fusão de dados.

- Fusão Tardia: do inglês, *Late Fusion*, também conhecida como *Coordinated Fusion*, ocorre quando cada modalidade de dado é processada separadamente e o resultado destes processamentos é combinado através de algoritmos de classificação e/ou estratégias de votação (MANGAI et al., 2010). O auxílio-diagnóstico de subtipos de transcritomas de adenocarcinoma pulmonar combinando a saída do processamento de imagens histopatológicas por redes neurais convolucionais profundas e o resultado do processamento destas mesmas imagens em AutoEncoders conforme descrito em (ANTONIO et al., 2018) é um exemplo de aplicação deste tipo de fusão de dados.

A multimodalidade em algoritmos de aprendizado de máquina profundo oferece uma série de vantagens em relação à multimodalidade no aprendizado de máquina tradicional. São elas: 1) é possível implementar a extração de características em dados com diferentes formas de representação; 2) a fase de pré-processamento dos dados é menor visto que é possível um processamento fim-a-fim dependendo do tipo de problema; 3) permite a utilização da fusão antecipada, intermediária ou tardia. Apesar das vantagens os algoritmos de aprendizado de máquina profundo exigem maior poder computacional e maior quantidade de dados rotulados na fase de treinamento (RAMACHANDRAM; TAYLOR, 2017).

2.2 Hepatocarcinoma

2.2.1 Conceitos Básicos

Câncer é o nome de um grupo de doenças que afeta seres vivos multicelulares e cuja principal característica é o crescimento anormal de células. Este crescimento anormal, denominado neoplasia, pode ser causado por fatores genéticos, etários, comportamentais ou ambientais. As células cancerosas podem ser benignas ou malignas. Células benignas são semelhantes às células sadias afetadas, não se espalham para regiões adjacentes e raramente causam risco de vida ao paciente. Células malignas possuem um comportamento errático, podem espalhar-se para regiões adjacentes e causam risco de vida ao paciente (Instituto Nacional de Cancer Jose Alencar Gomes da Silva, 2018). A classificação das neoplasias é feita baseada no tipo de tecido e no comportamento das células cancerosas. Papilomas, adenomas e rabdomiomas são exemplos de neoplasias benignas. Carcinomas, sarcomas e linfomas são exemplos de neoplasias malignas (BOMFORD et al., 2012a).

O câncer de fígado é causado pelo crescimento anormal das células hepáticas. Quando o tumor origina-se no próprio fígado denomina-se câncer de fígado primário e quando se origina em outro órgão (metástase) denomina-se câncer de fígado secundário.

Além da origem, o tipo e o local da neoplasia também servem para classificação do câncer de fígado primário. Adenoma hepatocelular e hiperplasia nodular focal hepática são tumores benignos originados a partir de hepatócitos. Adenoma do ducto biliar e cistoadenoma biliar são tumores benignos pouco prevalentes na população que acometem os ductos biliares internos do fígado. Hemangioma hepático e angiomiolipoma hepático são tumores benignos do tecido mesenquimal do fígado (SHERLOCK; DOOLEY et al., 2012a).

O carcinoma hepatocelular, também conhecido como hepatocarcinoma, é o tipo de tumor maligno do fígado com maior prevalência na população. Este tipo de câncer afeta os hepatócitos e comumente está associado a fatores genéticos e comportamentais aliados à presença de outras doenças no fígado como hepatite A-E, cirrose e a doença hepática gordurosa não-alcoólica. O colangiocarcinoma é o segundo tipo de tumor maligno do fígado mais prevalente na população. Neste tipo de tumor os ductos biliares internos do fígado são afetados e sua ocorrência está associada à infecções no fígado, hepatite C e hepatolitíase. Angiossarcoma hepático é o tipo de sarcoma mais prevalente no fígado, apesar de ser um tumor de baixa prevalência na população. Este tipo de tumor afeta o endotélio das veias do fígado e está associado à exposição do paciente a agentes químicos. O câncer de fígado pode ser diagnosticado através de ultrassonografia, tomografia computadorizada, ressonância magnética, laparoscopia, biópsia e exames laboratoriais (SHERLOCK; DOOLEY et al., 2012b).

O funcionamento hepático é comumente medido através da escala *Child-Pugh*. A

partir dos valores de resultados de exames de sangue como Bilirrubina, Albumina e Tempo de Protrombina associados à observação da presença de Ascite e Encefalopatia Hepática é realizado um cálculo com base em valores de referência cujo resultado é enquadrado categoricamente entre A (5-6 pontos), B (7-9 pontos) e C (10-15 pontos). Quanto maior a quantidade de pontos, menor é a chance de sobrevivência do paciente devido à insuficiência hepática (Working Subgroup for Clinical Practice Guidelines for Primary Biliary Cirrhosis, 2014).

O estadiamento é o método utilizado para classificar a extensão de um tumor. Em hepatocarcinomas, o estadiamento pode ser medido por diferentes escalas: 1) Escala BCLC: proposta pela *Barcelona Clinic Liver Cancer*, mede o estágio do hepatocarcinoma em razão do tamanho e da quantidade dos nódulos cancerígenos associados a classificação *Child Pugh*; 2) Escala TNM: mede o estágio do hepatocarcinoma em razão do tamanho do tumor primário e da disseminação de metástases; 3) Escala Okuda: classifica o estágio do hepatocarcinoma em razão do tamanho dos nódulos e do nível de Albumina e Bilirrubina no sangue 4) Escala CLIP: classifica o estágio do hepatocarcinoma baseada em fatores como escala *Child Pugh*, atributos clínicos, tamanho e quantidade de nódulos cancerígenos (MARRERO et al., 2018).

2.2.2 Diagnóstico

A diretriz clínica recomendada para diagnóstico de hepatocarcinoma pela Associação Americana para o Estudo de Doenças do Fígado (em inglês, *American Association for the Study of Liver Diseases* - AASLD) sugere que a escolha dos exames diagnósticos utilizados seja relativa ao contexto assistencial. A AASLD não recomenda a utilização exclusiva de marcadores tumorais (especificamente Alfafetoproteína) para diagnóstico de hepatocarcinoma e recomenda cautela na utilização de novos biomarcadores. Conforme pode ser visto na Figura 5, deve-se utilizar Ultrassom e o biomarcador Alfafetoproteína (AFP) para pacientes sob vigilância. Caso encontre-se lesões com mais de 1 cm ou $AFP > 20$ ng/mL, recomenda-se realização de tomografia computadorizada ou ressonância magnética multifásica e avaliação dos achados com base na escala *Liver Imaging Reporting and Data System* (LI-RADS) (HEIMBACH et al., 2018) (MARRERO et al., 2018). A escala LI-RADS foi criada em 2011 pela Associação Americana de Radiologia para normatizar os achados em exames de imagem para diagnóstico de hepatocarcinoma (CHERNYAK et al., 2018) e em 2018 a escala foi incorporada à recomendação da AASLD. Quanto maior o grau da escala, maior o tamanho do achado e consequentemente maior a certeza em relação ao diagnóstico realizado.

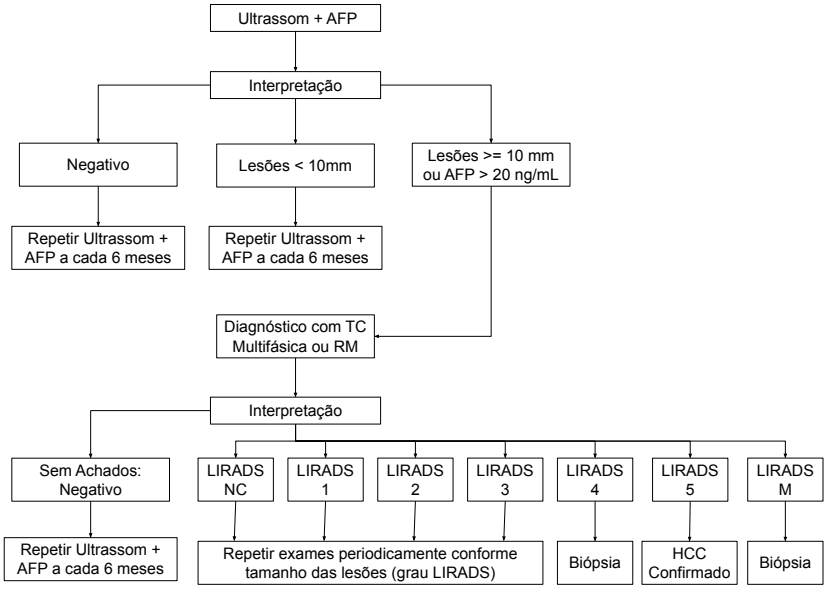


Figura 5 – Protocolo de Diagnóstico da AASLD. Baseado em (HEIMBACH et al., 2018).

A diretriz clínica recomendada para diagnóstico de hepatocarcinoma pela Associação Europeia para o Estudo do Fígado (em inglês, *European Association For The Study Of The Liver and Others* - EASL) sugere o emprego de mais de uma técnica diagnóstica para nódulos com tamanho entre 1 cm e 2 cm. A recomendação da EASL para nódulos com tamanho maior que 2 cm é utilizar somente exames radiológicos no caso do diagnóstico ser realizado em centros de excelência e com equipamentos de ponta. No caso do diagnóstico não ser realizado em centros de excelência a recomendação da EASL é usar mais de uma técnica diagnóstica. Para nódulos com tamanho menor que 1 cm a recomendação é repetir o ultrassom a cada 4 meses. (GALLE et al., 2018). A Figura 6 ilustra o fluxo do diagnóstico recomendado pela EASL.

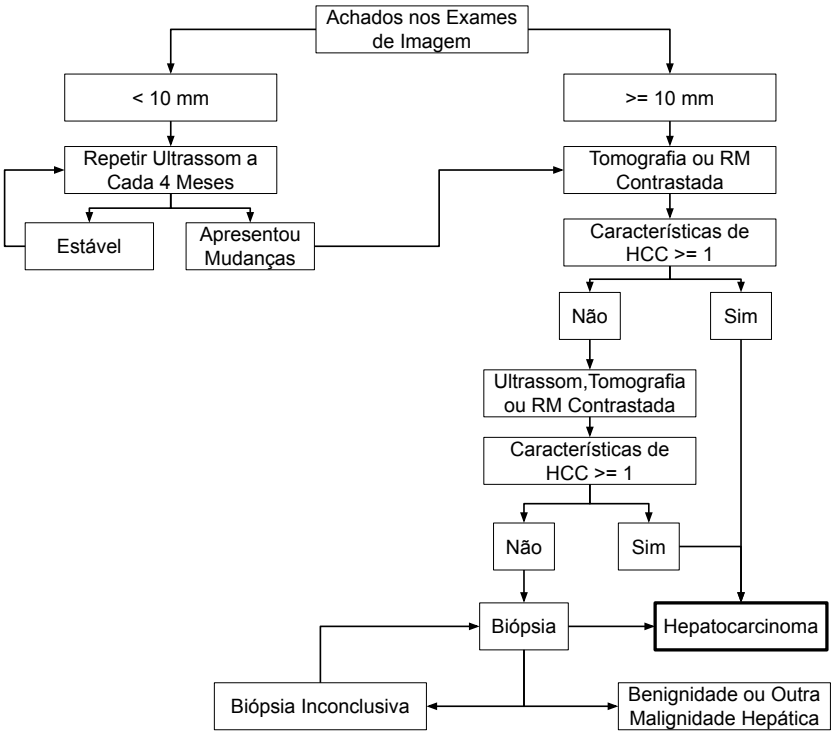


Figura 6 – Protocolo de Diagnóstico da EASL. Baseado Em: (GALLE et al., 2018).

Não foi encontrado na literatura estudo que indique o tempo médio de duração do processo diagnóstico. Possivelmente, a inexistência de estudos sobre este assunto seja causada pela diversidade de exames diagnósticos realizados dependendo do estágio da doença e pela diferença do tempo de entrega de cada um dos fornecedores destes exames. Em casos positivos da doença, o caminho mais curto pelo fluxograma de diagnóstico requer a realização de pelo menos um exame de tomografia computadorizada ou ressonância magnética e o caminho mais longo passa pela realização de biópsias após o resultado destes exames de imagem serem inconclusivos.

2.2.3 Tratamento

Os protocolos de tratamento de hepatocarcinoma baseiam-se no estadiamento da doença. Ambas as condutas recomendadas pela EASL e pela AASLD utilizam a escala BCLC como métrica para tomada de decisão. A figura 7 ilustra a recomendação da EASL para o tratamento de hepatocarcinoma.

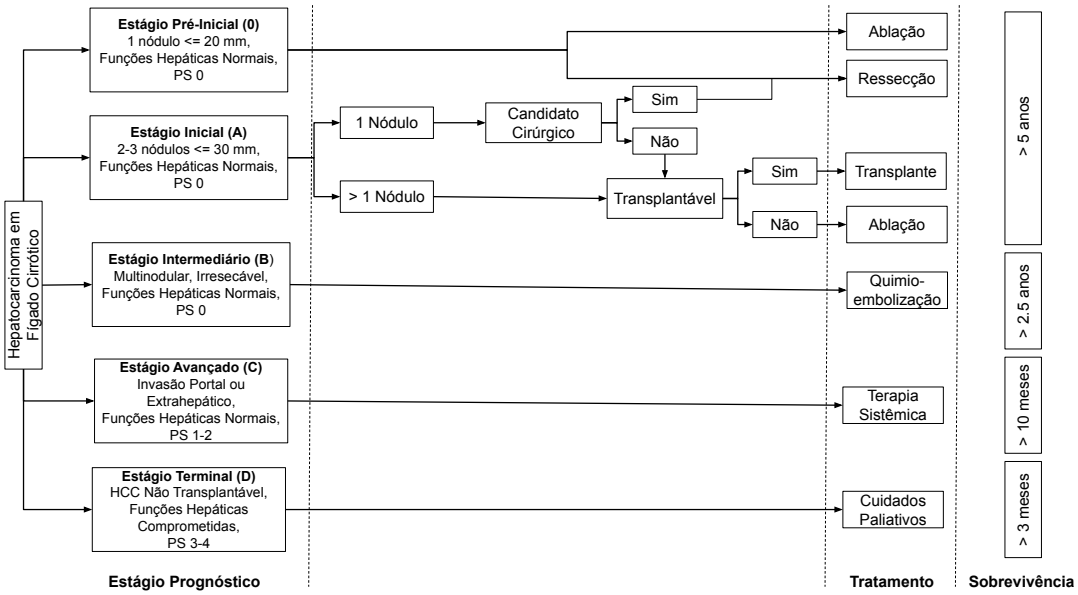


Figura 7 – Protocolo de Tratamento Recomendado pela EASL. Baseado Em: (GALLE et al., 2018)

A ablação e ressecção dos nódulos cancerígenos são realizadas em fases iniciais da doença (BLCL-0, BCLC-A e BCLC-B). Quando a doença encontra-se em estágio intermediário (BCLC-C) é realizada quimioembolização dos nódulos cancerígenos. Em fases mais avançadas da doença (BCLC-D) a terapia sistêmica (com Sorafenibe no Brasil) é o tratamento recomendado e em fase terminal, quando não existem mais chances de cura, recomendam-se apenas cuidados paliativos visando o bem-estar do paciente (Alejandro Forner and María Reig and Jordi Bruix, 2018).

3 Trabalhos Relacionados

A popularização no uso de técnicas de aprendizado de máquina profundo multimodal acentuou-se a partir de 2015 como pode ser visto no gráfico da Figura 8 que contém dados extraídos do *Google Trends* para o termo '*Multimodal Deep Learning*'. O eixo Y deste gráfico representa um valor normalizado entre 0 e 100 calculado a partir da razão entre o número de buscas do termo e o total de buscas realizadas no buscador Google.

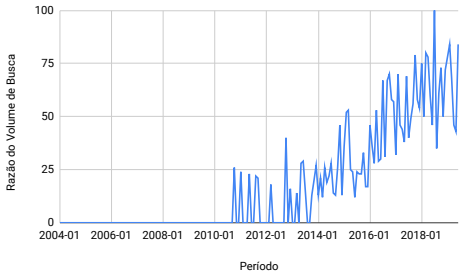


Figura 8 – Popularidade do Aprendizado de Máquina Profundo Multimodal. Fonte: (Google Inc., 2019).

A mesma pesquisa realizada no Pubmed demonstra o crescente interesse em pesquisas científicas que utilizam técnicas de aprendizado de máquina profundo multimodal em aplicações na área da saúde, como pode ser visto na Figura 9. Dentre os estudos encontrados no Pubmed pode-se citar aplicações para segmentação de nódulos cancerígenos (ZHAO et al., 2019) (MLYNARSKI et al., 2019), auxílio-diagnóstico de tumor cerebral (LAUKAMP et al., 2019) (JUN et al., 2018), auxílio-diagnóstico de acidente vascular cerebral (PRAVEEN et al., 2019) e auxílio-diagnóstico de Alzheimer (SINGANAMALLI; WANG; MADABHUSHI, 2017) (NGUYEN et al., 2019).

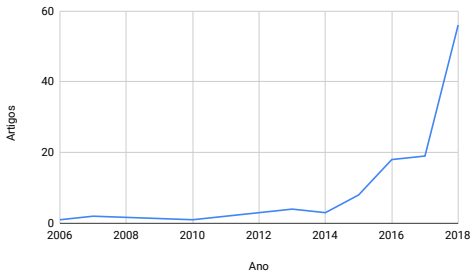


Figura 9 – Aprendizado de Máquina Profundo Multimodal na Área da Saúde. Fonte: (US National Library of Medicine, 2019).

A utilização de aprendizado de máquina profundo multimodal para auxílio-diagnóstico computadorizado ainda é incipiente na literatura quando comparada com outras técnicas

de inteligência artificial como o próprio aprendizado de máquina. A revisão de literatura conduzida em abril de 2018 objetivou identificar artigos que utilizam *técnicas de inteligência artificial* para auxílio-diagnóstico de hepatocarcinoma. Os parâmetros utilizados nesta busca foram:

- Palavras Chave: ((“machine learning” OR “deep learning” OR “artificial intelligence”) AND (“liver cancer” OR “hepatocellular carcinoma” OR “hepatoma” OR “liver neoplasm” OR “liver tumour”))
- Período de Publicação: artigos publicados a partir de 01/01/2015
- Filtros de Busca: Keywords found in Title, Abstract, e Keywords.
- Bases de Dados: Scopus, IEEE Xplore, Pubmed Central, Science Direct, Bireme, Nature e Web of Science.

A busca resultou em 291 artigos e após remoção de duplicados e aplicação de critérios de inclusão e exclusão restaram 46 artigos para análise. Os critérios de inclusão e exclusão utilizados estão descritos na Tabela 2.

Tabela 2 – Critérios de Inclusão e Inclusão da Revisão Sistemática de 2018

Inclusão	Exclusão
- Artigos Originais - Artigos Revisados por Pares - Artigos que abordam inteligência artificial e hepatocarcinoma - Artigos escritos em Inglês, Português e Espanhol	- Artigos de Revisão - Artigos duplicados - Artigos que não descrevem resultados - Artigos que abordam exclusivamente inteligência artificial ou hepatocarcinoma

Os artigos analisados demonstraram a prevalência do aprendizado profundo e de *Ensemble Learning* utilizando exames de imagem e atributos genéticos de maneira unimodal. A utilização de múltiplas modalidades de dados para auxílio-diagnóstico de hepatocarcinoma foi encontrada em 6 artigos nesta revisão:

- (MIOTTO; LI; DUDLEY, 2016) combinou informações extraídas do prontuário eletrônico do paciente como laudos radiológicos, atributos clínicos e informações sociodemográficas. A extração de características foi realizada com *Stacked Denoising AutoEncoders* (sDAE) e a camada de classificação foi implementada com florestas aleatórias para predição de 72 doenças. A predição de câncer de fígado obteve acurácia de 92.5%, sendo esta a doença cujo auxílio-diagnóstico computadorizado obteve o maior desempenho neste trabalho.

- (CHEN et al., 2017) combinou características extraídas de imagens de tomografia computadorizada como tamanho, perímetro e grau de deformação de nódulos e o laudo dos achados radiológico para classificação de lesões focais do fígado como benignas ou malignas. A classificação destas características foi testada em diferentes algoritmos de aprendizado de máquina como Redes Bayesianas e Máquinas de Vetores de Suporte obtendo acurácia máxima de 91.71% ao utilizar a abordagem proposta pelos autores denominada *Three-way Decision*.

- (ALI et al., 2017) combinou características extraídas por transformada Wavelet de imagens de tomografias computadorizadas e atributos clínicos para auxílio-diagnóstico de hepatocarcinoma. O algoritmo de aprendizado de máquina utilizado para classificação foi a Máquina de Vetor de Suporte e a acurácia máxima obtida neste trabalho foi de 88.46%.

- (RAMKUMAR et al., 2017) combinou atributos clínicos e informações comportamentais para auxílio-diagnóstico computadorizado de hepatocarcinoma utilizando redes bayesianas. A acurácia obtida para o conjunto de 23 pacientes de testes foi de 100%.

- (ABAJIAN et al., 2018) combinou atributos clínicos, sociodemográficos e exames de ressonância magnética para prognóstico de tratamento de quimioembolização em pacientes com hepatocarcinoma obtendo acurácia média de 78.00% utilizando florestas aleatórias.

- (WANG et al., 2018) combinou atributos clínicos e biomarcadores de câncer encontrados na urina para auxílio-diagnóstico computadorizado de hepatocarcinoma. Ambas as informações foram processadas utilizando diferentes abordagens de aprendizado de máquina e a Floresta Aleatória foi a técnica que obteve o maior desempenho neste trabalho, atingindo acurácia máxima de 94.70%.

O resultado da primeira revisão realizada não foi satisfatório devido à abrangência dos filtros de busca utilizados. Então no primeiro trimestre de 2019 optou-se por desenvolver uma nova revisão de literatura focando nas técnicas de aprendizado de máquina para auxílio-diagnóstico computadorizado de câncer (qualquer tipo de carcinoma). Os parâmetros de busca utilizados nesta revisão foram:

- Palavras Chave: multimodal AND “deep learning” AND (cancer OR tumor OR neoplasm)

- Período de Publicação: artigos publicados a partir de 01/01/2015

- Bases de Pesquisa: IEEE Xplore, Scopus, Pubmed Central, Science Direct, Bireme e Nature.

A busca resultou em 479 artigos e após remoção de duplicados e aplicação de critérios de inclusão e exclusão restaram 52 artigos para análise. Os critérios de inclusão e exclusão utilizados estão descritos na Tabela 3.

Tabela 3 – Critérios de Inclusão e Inclusão da Revisão de Literatura de 2019.

Critérios de Inclusão	Critérios de Exclusão
- Artigos originais - Artigos revisados por pares - Artigos escritos em Inglês	- Artigos de Revisão - Artigos Duplicados - Artigos sem resultados e métricas de desempenho
- Artigos que utilizam aprendizado de máquina profundo multimodal para auxílio-diagnóstico de câncer	- Artigos cujo objetivo engloba exclusivamente segmentação de câncer ou prognóstico de sobrevida

Nesta segunda revisão, predominaram trabalhos que tratam de tumores cerebrais e pulmonares. Não foi encontrado nenhum trabalho relacionado ao auxílio-diagnóstico computadorizado de hepatocarcinoma utilizando aprendizado de máquina profundo multimodal com os filtros acima.

Portanto, para aumentar a abrangência desta revisão, em agosto de 2019 foi realizada nova busca utilizando o Google Scholar com o intuito de encontrar artigos sobre auxílio-diagnóstico de hepatocarcinoma utilizando técnicas de *aprendizado de máquina* publicados a partir de 2015 que ainda não haviam aparecido nas revisões previamente realizadas. O motor de busca trouxe artigos já vistos anteriormente e também artigos focados exatamente no contexto deste trabalho e que não haviam aparecido nas revisões anteriores pois foram publicados em anais de eventos, prática muito comum na área da ciência da computação para divulgação de trabalhos científicos. Os artigos com foco em auxílio-diagnóstico computadorizado de hepatocarcinoma com aprendizado de máquina profundo multimodal encontrados foram:

- (BEN-COHEN et al., 2016) extrai características com técnicas de processamento de imagens aplicadas em tomografias computadorizadas para auxílio-diagnóstico computadorizado de hepatocarcinoma através da criação de representações esparsas armazenadas em dicionários K-SVD (AHARON; ELAD; BRUCKSTEIN, 2006). A acurácia máxima obtida com a abordagem LC-KSVD foi de 96% em uma base de imagens com 68 estudos de 20 pacientes.

- (QIAN et al., 2017), (MENG et al., 2018) e (GUO et al., 2018) utilizam aprendizado de máquina multimodal implementado com abordagens *multi-kernel* que processam insumos extraídos de múltiplas visões de ultrassom contrastado. A acurácia máxima obtida nestes trabalhos foi de $83.66\% \pm 2.30$ na abordagem proposta por (MENG et al., 2018) denominada MRBM-MEKL.

- (WANG et al., 2017) combina cortes 2D de imagens de ressonância magnética contrastada dos planos axial, sagital e coronal em um plano ortogonal que representa a região de interesse do achado radiológico. O volume criado é processado por uma rede convolucional profunda multimodal para extração de atributos que são posteriormente classificados utilizando máquina de vetor de suporte *multi-kernel* visando determinar a malignidade de um nódulo hepático. A acurácia máxima obtida neste trabalho foi de $99.23\% \pm 2.43\%$.

- (DOU; ZHOU, 2018) combina *patches* 16 x 16 extraídos de cortes 2D de imagens de ressonância magnética contrastada dos planos arterial, axial, sagital e coronal e *patches* 3D de tamanho 16 x 16 x 16 do mesmo achado radiológico em uma caixa delimitadora volumétrica processada por uma rede convolucional profunda multimodal visando determinar o grau de malignidade de um nódulo hepático. A acurácia máxima obtida neste exemplo de fusão antecipada é de $92.31\% \pm 5.96\%$.

- (DOU et al., 2018) extrai atributos de imagens de ressonância magnética contrastada utilizando redes convolucionais profundas e técnicas de processamento de imagem. Usando fusão de dados, estes atributos são combinados para determinar a malignidade de nódulos cancerígenos do fígado usando uma rede neural profunda. A acurácia máxima obtida neste trabalho foi de $93.16\% \pm 4.36\%$ processando as imagens como tensores 3D e utilizando fusão de todos os atributos extraídos.

- (DOU; ZHANG; ZHOU, 2018) utilizou diferentes fases de um estudo abdominal de ressonância magnética contrastada (pré-contraste, arterial e portal) para determinar a malignidade de nódulos cancerígenos de fígado através de fusão intermediária em uma rede convolucional profunda multimodal 3D. A acurácia máxima obtida com a fusão de todas as fases foi de $89.23\% \pm 6.92\%$.

- (TIAN et al., 2018) descreve a criação de uma ferramenta que elabora o laudo radiológico de pacientes para auxílio-diagnóstico de hepatocarcinoma a partir do processamento de estudos de tomografia computadorizada e máscaras de verdade absoluta utilizando redes neurais convolucionais e redes *Long short-term memory* (LSTM).

- (ZHOU et al., 2019) combina diferentes atributos de imagens de ressonância magnéticas difusas (DWI-MRI) utilizando fusão tardia em redes convolucionais profundas multimodal para enquadrar as entradas na classificação Edmondson-Steiner (ZHOU et al., 2017) obtendo acurácia máxima de 80%.

- (LIN et al., 2019) utiliza múltiplas imagens histopatológicas para realização de auxílio-diagnóstico de hepatocarcinoma usando fusão intermediária e uma rede convolucional profunda multimodal baseada na rede VGG16. A acurácia máxima obtida neste estudo foi de 94.1%.

- (KSIĄŻEK et al., 2019) testa diferentes abordagens de aprendizado de máquina baseadas em *kernel* e atinge acurácia máxima de 78% no auxílio-diagnóstico de hepatocarcinoma na base de testes utilizando atributos clínicos, comportamentais e resultados de exames laboratoriais.

Os artigos resumidos nesta seção demonstram múltiplas abordagens para construção de sistemas de auxílio-diagnóstico computadorizado de hepatocarcinoma utilizando algoritmos de aprendizado de máquina multimodal. Apesar da existência de trabalhos que já utilizam aprendizado de máquina profundo multimodal para auxílio-diagnóstico computadorizado de hepatocarcinoma, não foi encontrado na revisão de literatura realizada nenhum artigo que combine imagens de tomografia computadorizada, atributos clínicos, antropométricos e sociodemográficos com exames laboratoriais como insumo de entrada em arquiteturas convolucionais profundas.

4 Materiais e Métodos

O trabalho realizado constitui-se em pesquisa aplicada, com perfil exploratório, e abordagem quantitativa. Trata-se de experimentação de tecnologia computacional aplicada para aprendizado em profundidade. Este capítulo descreve em detalhes os passos percorridos desde a obtenção dos dados até o processo de experimentação propriamente dito.

4.1 Preparação das Bases de Dados

4.1.1 Aquisição dos Dados

O 'Arquivo de Imagens de Câncer' (em inglês *The Cancer Imaging Archive* - TCIA) é uma base de dados pública de exames radiológicos mantida pelo *Frederick National Laboratory* e patrocinada pelo *National Cancer Institute*, ambas ligadas ao Departamento de Saúde e Serviços Sociais do governo americano. O TCIA é composto de exames de imagem de múltiplas modalidades (ex: ressonância magnética, tomografia computadorizada, ultrassom) dos mais variados tipos de câncer (ex: fígado, estômago, rins, pulmão, cérebro, próstata) doados por instituições parceiras (CLARK et al., 2013).

O TCIA é um subprojeto do 'Atlas Genômico do Câncer' (em inglês *The Cancer Genome Atlas* - TCGA) (HUTTER; ZENKLUSEN, 2018), um projeto conjunto do *National Cancer Institute* (NCI) e do *National Human GenomeResearch Institute* (NHGRI) cujo objetivo é disponibilizar bases de dados de pacientes reais anonimizadas para fins científicos. Outro subprojeto do TCGA é o 'Dados Genômicos Comuns' (em inglês *Genomic Data Commons* - GDC) que objetiva fornecer dados clínicos, genômicos e sociodemográficos que podem ser relacionados aos exames de imagem do TCIA através do identificador do paciente.

Os dados são disponibilizados de forma totalmente anonimizada pelo TCGA, não sendo necessária nenhuma ação em relação à privacidade dos pacientes. As bases de dados do TCGA utilizadas neste trabalho foram:

- A 4^a versão da base *The Cancer Genome Atlas Liver Hepatocellular Carcinoma [TCGA-LIHC] collection* (TCGA-LIHC), publicada em 30/01/2017 e com 52.2 GB de imagens DICOM relacionadas a diagnósticos positivos de hepatocarcinoma. São 237 estudos de 97 pacientes submetidos ao TCIA pela Clínica Mayo, Universidade da Carolina do Norte, Hospital de Alberta e Hospital Lahey totalizando 125.397 imagens adquiridas através de ultrassonografias, tomografias computadorizadas e ressonâncias magnéticas. Os dados clínicos, biomarcadores tumorais e informações

sociodemográficas associados às imagens são publicados em arquivos separados por vírgula e possuem tamanho total de 112 KB (ERICKSON et al., 2017).

- A 2ª versão da base *The Cancer Genome Atlas Stomach Adenocarcinoma* (TCGA-STAD), publicada em 05/01/2016 e com 23.3 GB de imagens DICOM relacionadas a diagnósticos positivos de adenocarcinoma no estômago. São 46 estudos de 46 pacientes submetidos ao TCIA pelo Hospital do Câncer de Barretos totalizando 43.908 imagens de tomografia computadorizada. Os dados clínicos, biomarcadores tumorais e informações sociodemográficas associados às imagens são publicados em arquivos separados por vírgula e possuem tamanho total de 82 KB (LUCCHESI F. R., 2016).

- A 3ª versão da base *The Cancer Genome Atlas Cervical Kidney renal papillary cell carcinoma* (TCGA-KIRP), publicada em 30/01/2017 e com 9.6 GB de imagens DICOM relacionadas a diagnósticos positivos de carcinoma renal. São 47 estudos de 33 pacientes submetidos ao TCIA pelo Instituto Nacional de Câncer de Bethesda, Universidade da Carolina do Norte, Instituto do Câncer Roswell Park e Hospital Lahey totalizando 26.667 imagens adquiridas através de ultrassom, tomografia computadorizada e ressonância magnética. Os dados clínicos, biomarcadores tumorais e informações sociodemográficas associados às imagens são publicados em arquivos separados por vírgula e possuem tamanho total de 59 KB (LINEHAN M., 2016).

- A 4ª versão da base *Clinical Proteomic Tumor Analysis Consortium Pancreatic Ductal Adenocarcinoma* (CPTAC-PDA), publicada em 24/10/2018 e com 21.8 GB de imagens DICOM relacionadas a diagnósticos positivos de carcinoma do pâncreas. São 308 estudos de 44 pacientes submetidos ao TCIA pelo *Beaumont Health System*, Instituto de Oncologia Molecular da Polônia, *St. Joseph's Hospital*, Universidade de Calgary (Canadá) e *Cureline, Inc.* totalizando 40.010 imagens adquiridas através de ultrassom, tomografia computadorizada, ressonância magnética, radiografia digital e tomografia por emissão de pósitrons. Esta versão desta base não fornece dados clínicos, biomarcadores tumorais e informações sociodemográficas associados às imagens (National Cancer Institute Clinical Proteomic Tumor Analysis Consortium, 2018).

4.1.2 Pré-Processamento dos Dados

A qualidade dos dados utilizados para treinamento de arquiteturas de aprendizado de máquina profundo é um dos fatores que influenciam a convergência do aprendizado (LITJENS et al., 2017). Após a aquisição dos dados é necessária uma fase de análise para cada uma das entradas utilizadas para avaliar seu grau de qualidade e determinar quais passos de pré-processamento são necessários antes da fase de treinamento. As subseções

4.1.2.1 e 4.1.2.2 descrevem os passos de pré-processamento realizados em cada uma das modalidades de entrada utilizadas no algoritmo de aprendizado profundo multimodal para auxílio-diagnóstico de hepatocarcinoma proposto nesta dissertação.

4.1.2.1 Imagens

Os exames de imagem obtidos do TCIA são de múltiplas modalidades. Portanto, a primeira decisão tomada é em relação a modalidade de imagem utilizada na arquitetura proposta nesta dissertação.

A utilização de imagens de raios-X não é recomendada para detecção de hepatocarcinoma, pois neste tipo de exame não é possível exibir contrastes que permitem a identificação de nódulos cancerígenos (BOMFORD et al., 2012b). Imagens de ultrassom por sua vez são recomendadas durante a fase de vigilância do hepatocarcinoma, porém o enquadramento e a qualidade das imagens são dependentes da aparelhagem e do operador do exame. A qualidade visual obtida em imagens de ressonância magnética faz com que esta modalidade seja recomendada pelos protocolos de diagnóstico de hepatocarcinoma, porém o alto custo na aquisição deste tipo de imagem limita a aplicação deste tipo em sistemas de auxílio-diagnóstico de hepatocarcinoma (SHERLOCK; DOOLEY et al., 2012c). Sendo assim, a modalidade de imagem escolhida para o algoritmo de auxílio-diagnóstico de hepatocarcinoma descrito nesta dissertação é a tomografia computadorizada, que também é recomendada pelos protocolos de diagnóstico de hepatocarcinoma, fornece imagens de qualidade que permitem a visualização dos nódulos cancerígenos e possui custo financeiro inferior a ressonância magnética.

Os estudos de tomografia computadorizada encontrados nas bases obtidas possuem características diversas: múltiplos planos anatômicos, diversos protocolos de aquisição e tomógrafos de inúmeras marcas. Esta diversidade de fatores dificulta a utilização de todas as imagens para o treinamento e experimentação de uma arquitetura de aprendizado de máquina profundo multimodal para auxílio-diagnóstico de hepatocarcinoma. Portanto, após o *download* das imagens foi necessário extrair do conjunto original as imagens para o treinamento e experimentação da arquitetura proposta nesta dissertação. A extração das imagens de tomografia computadorizada foi realizada de forma automatizada utilizando um programa escrito em Python que analisa a modalidade da imagem gravada no cabeçalho DICOM e copia para uma pasta destino as imagens de tomografia computadorizada. O código-fonte desse programa está disponível no Apêndice A.1. As operações de leitura das imagens que estão no formato DICOM foi realizada com a biblioteca *pyDicom* e a biblioteca *numpy* foi utilizada para manipulação matemática da matriz 2D que representa uma imagem de tomografia computadorizada.

O próximo passo foi extrair os estudos que trazem imagens do fígado: estudos abdominais, abdominais totais e abdominais pélvicas. A extração ocorreu de forma semi-

automatizada utilizando primeiramente um programa Python que avalia o atributo *ImageType* do cabeçalho DICOM (vide Apêndice A.2). Após, uma revisão manual foi realizada com o visualizador gratuito *MicroDICOM DICOM Viewer* (MicroDicom Inc., 2019) para identificar possíveis divergências existentes no cabeçalho DICOM e selecionar dentre estes estudos somente cortes axiais que mostram o fígado independente do protocolo de aquisição utilizado. Os cortes selecionados foram catalogados em planilha eletrônica para posterior processamento e criação das bases de dados. Esta etapa foi necessária para deixar o foco do aprendizado na análise do fígado, pois as imagens serão processadas de forma 2D pelas técnicas de aprendizado de máquina profundo, isoladas dos demais cortes do estudo.

A Tabela 4 ilustra a quantidade de imagens manipulada em cada uma destas fases e a Tabela 5 descreve a base de imagens utilizada para treinamento, validação e testes da arquitetura de aprendizado de máquina profundo para auxílio-diagnóstico de hepatocarcinoma desenvolvida nesta dissertação.

Tabela 4 – Visão Geral dos Passos de Seleção das Imagens.

Passo	**Tamanho**	**Qtd Imagens**	**Qtd Pacientes**
Original	102,41 GB	219.403	210
Seleção de TC Axiais	57,8 GB	115.621	175
Seleção de Estudos Abdominais	54,99 GB	111.664	174
Seleção de Cortes que Mostram o Fígado	23,86 GB	46.832	174

Tabela 5 – Visão Geral da Base de Imagens.

Origem	**Tamanho**	**Qtd Imagens**	**Qtd Pacientes**	**Qtd Estudos**	**Classe**
TCGA-LIHC	10.2 GB	20,792	73	402	Positivo
TCGA-KIRP	1.82 GB	1,968	22	48	Negativo
TCGA-STAD	6.76 GB	13,741	46	184	Negativo
CPTAC-PDA	5.08 GB	10,331	33	132	Negativo
Totais	23.86 GB	46,832	174	766	

Após selecionar as imagens que serão utilizadas, é necessário pré-processá-las e convertê-las para um formato compreensível pelos *frameworks* de aprendizado de máquina. As imagens de tomografia computadorizadas são armazenadas na escala *Hounsfield Unit* (HU), que representa de maneira universal os coeficientes de atenuação que formam o contraste da imagem (BUZUG, 2011). A conversão de escalas conforme os valores do cabeçalho DICOM representam o primeiro passo deste pré-processamento das imagens, detalhado no trecho de código Python abaixo:

```
def read_dcm(inputdir, file):
    ds = pydicom.dcmread(inputdir + '/' + file)
    image = np.stack(ds.pixel_array)
```

```
image = image.astype(np.int16)
image[image == -2000] = 0
intercept = ds.RescaleIntercept
slope = ds.RescaleSlope
if slope != 1:
    image = slope * image.astype(np.float64)
    image = image.astype(np.int16)
image += np.int16(intercept)
```

A imagem retornada pela função acima não é nítida. A melhora da nitidez da imagem pode ser realizada utilizando algoritmos de equalização de histograma e filtragem de ruído. O algoritmo *Contrast Limited Adaptive Histogram Equalization* (CLAHE) (PIZER et al., 1987), utilizado em trabalhos encontrados na literatura para equalização de contraste em imagens médicas (AL-AMEEN et al., 2015) (MEN; DAI; LI, 2017) foi aplicado nas imagens. A filtragem de ruído foi realizada com o algoritmo *Chambolle Total Variation Denoising* (CHAMBOLLE, 2004), que obtém resultados visualmente mais nítidos em imagens de tomografia computadorizada quando comparados com outros algoritmos similares como *Wavelet* e *Bilateral Denoising*. A biblioteca *scikit-image* (WALT et al., 2014) foi utilizada nesta etapa do pré-processamento das imagens.

O resultado da aplicação destes algoritmos melhora visualmente as imagens como pode ser observado na Figura 10, porém esta melhora não implica num maior desempenho do algoritmo de aprendizado de máquina profundo, visto que estes passos podem ter removido informações invisíveis a olho nu mas que representam um padrão identificável pelas redes convolucionais. Portanto, serão testadas duas versões distintas das bases de imagem: a primeira possui imagens que passaram pelos passos descritos no parágrafo anterior e a segunda possui as imagens originais que não passaram pelo realce de nitidez e filtragem de ruído.

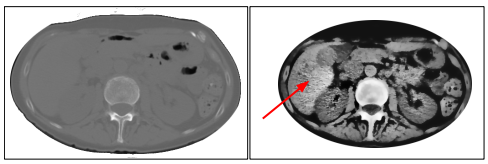


Figura 10 – Resultado Visual do Pré-Processamento das Imagens. Fonte: Do Autor

A compressão foi o último passo do pré-processamento das imagens. O formato de arquivos escolhido foi o PNG, que garante uma compressão sem perda na qualidade das imagens. Os arquivos no formato DICOM possuem em média 528.28 Kb, os arquivos no formato PNG sem realce de nitidez e filtragem de ruído possuem em média 98.05 Kb e os arquivos no formato PNG com realce de nitidez e filtragem de ruído possuem em média

58.98 Kb. A compressão e gravação dos arquivos foi realizada utilizando a biblioteca SciPy (JONES et al., 2019).

O nome de cada imagem no sistema de arquivos é composto pelo ID do paciente concatenado com um identificador universal gerado de forma aleatória através de um UUID v4 (LEACH; MEALLING; SALZ, 2005). O ID do paciente será utilizado na fusão de dados para relacionar os exames de imagem com os dados tabulares nas arquiteturas multimodais. A verificação da existência dos dados tabulares de todos os pacientes que compõem a base de imagens foi realizada previamente, através de um programa que verifica se o ID do paciente da imagem existe na base de dados tabulares. Caso algum ID do paciente não possua dados tabulares (como de fato ocorreu), o conjunto de imagens pertencente a este paciente é removido.

Os passos descritos nesta subseção podem ser resumidos através do pseudo algoritmo descrito na Figura 11. A implementação destes passos foi realizada na linguagem de programação Python e pode ser vista no Apêndice A.3.

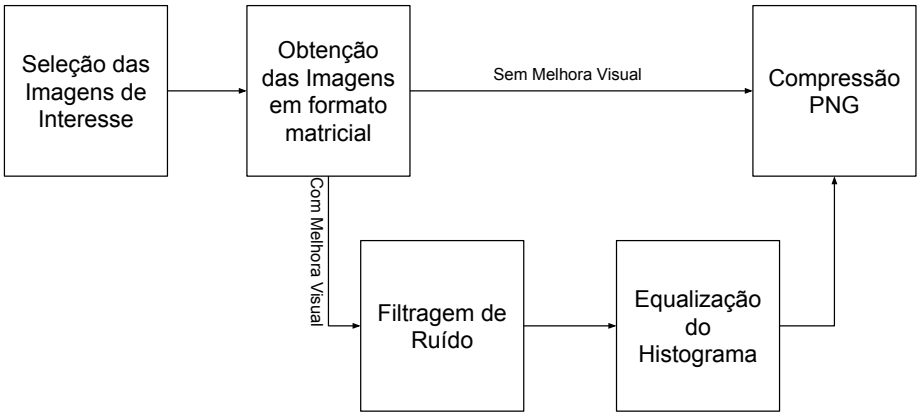


Figura 11 – Resumo do Pré-Processamento das Imagens. Fonte: Do Autor

4.1.2.2 Dados Tabulares

A segunda modalidade de dados utilizada no algoritmo de auxílio-diagnóstico de hepatocarcinoma proposto é composta de dados tabulares. Os dados tabulares disponibilizados pelo TCGA contém atributos sociodemográficos, dados antropométricos, fatores de risco relacionados ao hepatocarcinoma e resultados de exames laboratoriais, distribuídos entre atributos numéricos, ordinais, categóricos e alfanuméricos. São disponibilizados dados de 199 pacientes mas devido a seleção das imagens de interesse, somente 174 pacientes são efetivamente utilizadas nos experimentos.

A utilização destes dados em forma bruta não é possível, pois redes neurais trabalham somente com entradas numéricas. Além disso, existem diferenças entre os valores utilizados para representar o mesmo atributo em cada uma das fontes de dados e existem

atributos com valores não preenchidos. A solução destes problemas envolve a normalização dos dados e a aplicação de técnicas para imputação de dados faltantes (em inglês, *missing-data imputation*) (WALJEE et al., 2013) (KANG, 2013). O restante desta subseção descreve os passos realizados para o preenchimento dos dados faltantes e a normalização aplicada nos atributos.

Antes de iniciar o pré-processamento, é necessário selecionar os atributos relevantes para o diagnóstico de hepatocarcinoma dentre os atributos disponibilizados pelo TCGA. O processo de seleção foi guiado com base no protocolo de diagnóstico recomendado pela Associação Europeia para Estudo do Fígado (EASL) (GALLE et al., 2018). Uma visão geral dos 20 atributos escolhidos pode ser vista na Tabela 6.

Tabela 6 – Atributos dos Dados Tabulares

Nome	**Grupo**	**Descrição**
ID do Paciente	Dados Básicos	Identificador do paciente na base de dados.
Gênero	Dados Sociodemográficos	Sexo do paciente no momento do nascimento. Os valores utilizados foram 0 = Feminino, 1 = Masculino.
Idade	Dados Sociodemográficos	Idade do paciente em anos no momento do diagnóstico.
Raça	Dados Sociodemográficos	Raça do paciente. Os valores utilizados foram 0 = Branco, 1 = Afrodescendente, 2 = Asiático.
Etnia	Dados Sociodemográficos	Origem étnica do paciente. Os valores utilizados foram 0 = Não hispânico nem latino, 1 = Latino.
Altura	Dados Antropométricos	Altura do paciente em centímetros.
Peso	Dados Antropométricos	Peso do paciente em quilogramas.
Outras Malignidades	Fatores de Risco	Indicativo de outros carcinomas. Os valores utilizados foram 0 = Não, 1 = Sim.
Câncer na Família	Fatores de Risco	Indicativo de câncer na família. Os valores utilizados foram 0 = Não, 1 = Sim.
Qt. Câncer na Família	Fatores de Risco	Quantitativo de parentes ascendentes com câncer.
Álcool	Fatores de Risco	Indicativo de consumo de álcool. Os valores utilizados foram 0 = Não, 1 = Sim.

Continuação da Tabela 6

Nome	Grupo	Descrição
Hemocromatose	Fatores de Risco	Indicativo de Hemocromatose. Os valores utilizados foram 0 = Não, 1 = Sim.
Hepatite	Fatores de Risco	Indicativo de Hepatite [A-E]. Os valores utilizados foram 0 = Não, 1 = Sim.
Esteatose	Fatores de Risco	Indicativo de Esteatose Não-Alcoólica. Os valores utilizados foram 0 = Não, 1 = Sim.
Outras Doenças	Fatores de Risco	Indicativo de Outras Doenças no Fígado. Os valores utilizados foram 0 = Não, 1 = Sim.
Alfafetoproteína	Biomarcador Tumoral	Dosagem sérica da Alfafetoproteína medida em ng/mL.
Plaquetas	Exames Laboratoriais	Quantidade de Plaquetas medida em $1 \times 1000/mm^3$.
Tempo de Protrombina	Exames Laboratoriais	Resultado do Tempo de Protrombina medido conforme a Razão Normalizada Internacional (do inglês, *International Normalized Ratio* (INR).
Albumina	Exames Laboratoriais	Dosagem sérica da Albumina medida em g/dL.
Bilirrubina	Exames Laboratoriais	Dosagem sérica de Bilirrubinas medida em mg/dL.
Creatinina	Exames Laboratoriais	Dosagem sérica de Creatinina medida em mg/dL.

O Identificador (ID) do paciente serve para relacionar os dados tabulares com os exames de imagem no momento da fusão de dados. Os atributos que pertencem ao grupo 'Dados Sociodemográficos' servem para identificação do grupo populacional dos pacientes. A utilização destes atributos deve-se à preponderância da ocorrência de hepatocarcinoma em populações asiáticas (especialmente China e Ásia Oriental) e populações da África Subsaariana (BOSCH et al., 2005). Os dados antropométricos mostram uma visão geral do estado nutricional do paciente, que pode ser afetado negativamente por doenças do fígado como a cirrose hepática (O'BRIEN; WILLIAMS, 2008).

Os atributos componentes do conjunto 'Fatores de Risco' contém dados relacionadas a doenças e condições cuja pré-existência possui associação com o desenvolvimento de hepatocarcinoma, independente do grupo étnico do paciente. Analisando a literatura, percebe-se que a lista de atributos fornecidos pelo TCGA não engloba todas as causas de

hepatocarcinoma. Dentre as causas não cirróticas do hepatocarcinoma não disponibilizadas pode-se citar os Adenomas Hepáticos, a Tirosinemia Hereditária, a Cirrose Alcoólica, a Cirrose Primária Biliar e a Hepatite Autoimune (SHERLOCK; DOOLEY et al., 2012b).

A Alfafetoproteína é uma proteína sintetizada pelo fígado e que possui baixa dosagem sérica em adultos sadios. O aumento da dosagem sérica desta proteína pode indicar uma disfunção hepática. A Alfafetoproteína costuma ser utilizada como biomarcador para o hepatocarcinoma, apesar da baixa acurácia e precisão ($\leq 60\%$) obtida quando esta informação é utilizada de forma isolada para o diagnóstico (EL-SERAG et al., 2014). Atualmente, a utilização deste biomarcador só é recomendada durante a fase de vigilância da doença em conjunto com exames de imagem (ex: ultrassom) (GALLE et al., 2018) (HEIMBACH et al., 2018) (MARRERO et al., 2018).

O grupo 'Exames Laboratoriais' contém atributos que fornecem uma visão geral do estado de saúde do paciente em relação a disfunções hepáticas (LIU et al., 2016):

- Albumina: proteína plasmática produzida pelos hepatócitos que serve como meio de transporte de substâncias na corrente sanguínea e age como um regulador da pressão oncótica. Devido a sua origem, pode ser utilizada para monitoramento do funcionamento do fígado (WEAVING; BATSTONE; JONES, 2016).
- Bilirrubina: componente derivado da degradação da hemoglobina. O excesso desta substância no sangue pode indicar problemas hepáticos e renais (STICOVA; JIRSA, 2013).
- Creatinina: produto catabólico da fosfatocreatinina. Sua dosagem sérica pode ser utilizada para monitoramento das funções renais e hepáticas (LIM et al., 2010).
- Plaquetas: são componentes do sangue responsáveis pelo controle da hemóstase. A alteração na contagem de plaquetas pode ser indício de doenças hepáticas que podem preceder um hepatocarcinoma (HUGENHOLTZ; PORTE; LISMAN, 2009).
- Protrombina: substância produzida pelo fígado que ajuda na coagulação do sangue. O Tempo de Protrombina pode ser utilizado para monitorar o funcionamento do fígado (TRIPODI; MANNUCCI, 2011).

Após a escolha dos atributos tabulares utilizados no algoritmo para auxílio-diagnóstico computadorizado de hepatocarcinoma proposto, foi necessário unificar os dados obtidos de cada base. A unificação facilita o processo normalização e o preenchimento dos valores faltantes em cada um dos atributos. Os dados foram unificados primeiramente em planilha eletrônica para normalização das unidades de medida dos valores numéricos e para categorização ordinal dos valores disponibilizados de forma alfanumérica, necessária somente nos atributos componentes do grupo 'Fatores de Risco'.

A partir da criação desta planilha, todas as ações foram realizadas programaticamente utilizando uma versão destes dados no formato separado por vírgulas (CSV). A Tabela 7 descreve o estado inicial dos dados tabulares em relação ao percentual de valores faltantes em atributos.

Tabela 7 – Percentual de Registros Faltantes na Base Tabular

Atributo	**TCGA-LIHC** %	**TCGA-STAD** %	**TCGA-KIRP** %	**CPTAC-PDA** %
ID do Paciente	0	0	0	0
Gênero	0	0	0	100
Idade	0	0	0	100
Raça	3	0	0	100
Etnia	9	0	0	100
Altura	7	100	12	100
Peso	1	100	9	100
Outras Malignidades	0	0	0	100
Câncer na Família	1	0	100	100
Qt. Câncer na Família	36	86	100	100
Álcool	4	100	100	100
Hemocromatose	4	100	100	100
Hepatite	4	100	100	100
Esteatose	4	100	100	100
Outras Doenças	0	100	100	100
Alfafetoproteína	12	100	100	100
Plaquetas	1	100	0	100
Tempo de Protrombina	3	100	100	100
Albumina	13	100	100	100
Bilirrubina	4	100	100	100
Creatinina	1	100	100	100

O preenchimento de valores faltantes do atributo Gênero foi realizado utilizando geração de números pseudo-aleatórios ponderada guiado pelas probabilidades de gêneros versus nascimentos descritas em (GRECH; SAVONA-VENTURA; VASSALLO-AGIUS, 2002). O código-fonte do programa que realiza este preenchimento pode ser visto no Apêndice A.4.

Os dados antropométricos faltantes foram preenchidos utilizando números pseudo-aleatórios gerados a partir de uma distribuição normal baseada nas informações coletadas

do programa *National Health and Nutrition Examination Survey* (NHANES) (FRYAR et al., 2016). O código-fonte do programa utilizado para este preenchimento pode ser visto no Apêndice A.5

Os valores faltantes dos atributos Raça e Etnia foi preenchido utilizando geração de números pseudo-aleatórios ponderados tomando como base probabilística as informações descritas no Censo dos Estados Unidos de 2010 (HUMES; JONES; RAMIREZ, 2011). O código-fonte do programa que realiza este preenchimento pode ser visto no Apêndice A.8

Dados faltantes dos atributos relacionados aos fatores de risco (Outra Malignidade, Câncer na Família, Qtd. de Câncer na Família, Álcool, Hemocromatose, Hepatite, Esteatose e Outras Doenças) foram preenchidos através da geração de números pseudo-aleatórios ponderados utilizando como base probabilística os valores descritos em artigos científicos revisados por pares (ARAGON; YOUNOSSI, 2010) (KORAM et al., 2007) (PEACOCK et al., 2018) (DUFOUR et al., 2000) (BOSCH et al., 2005) (WAGHRAY; MURALI; MENON, 2015) (IMBERT-BISMUT et al., 2001) (EL-SERAG et al., 2014). O código-fonte do programa que realiza esta operação pode ser visto no Apêndice A.6.

O preenchimento dos dados faltantes relacionados aos resultados de exames laboratoriais foi realizado de forma separada por classe. Para os dados de pacientes portadores de hepatocarcinoma, os resultados de testes laboratoriais faltantes foram preenchidos utilizando uma rede neural artificial que prevê o resultado de cada um dos exames de forma exclusiva baseada em atributos como sexo, idade e o resultado dos demais exames laboratoriais. Esta abordagem já foi aplicada com sucesso em outros trabalhos da literatura (SILVA-RAMÍREZ et al., 2011). A arquitetura da rede utilizada pode ser vista na Figura 12. Esta rede neural foi treinada com uma base de dados de 165 pacientes anonimizados denominada *HCC Survival Data Set*, criada pela Universidade de Coimbra (SANTOS et al., 2015) e obtida através do repositório de aprendizado de máquina da Universidade da Califórnia (DUA; GRAFF, 2017). O código-fonte do programa que realiza esta operação pode ser visto no Apêndice A.7.

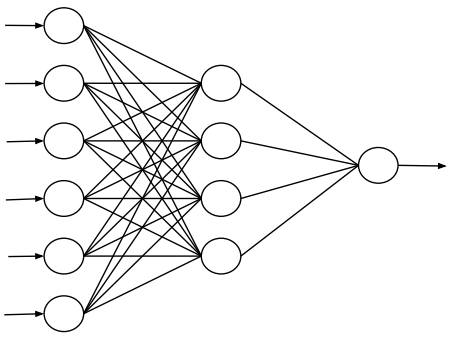


Figura 12 – Rede Neural para Preenchimento de Resultados Laboratoriais. Fonte: Do Autor

Para os dados de pacientes que não possuem hepatocarcinoma, os valores dos testes laboratoriais faltantes foram preenchidos utilizando geração de números pseudo-aleatórios em uma distribuição normal (MALIK et al., 2011) utilizando como base os valores de referência descritos na Tabela 8. O código-fonte do programa que realiza esta operação pode ser visto no Apêndice A.9.

Tabela 8 – Limiares de Referência para Exames Laboratoriais.

Exame	**Valor Médio**	**Sexo**	**Faixa Etária**	**Referência**
Alfa-fetoproteína	3.04 ± 1.9	M/F	[0-99]	(BALL; ROSE; ALPERT, 1992)
Plaquetas	290 ± 50	M	[0-40]	(BALDUINI; NORIS, 2014)
Plaquetas	240 ± 30	M	[40-60]	(BALDUINI; NORIS, 2014)
Plaquetas	240 ± 50	M	[60-99]	(BALDUINI; NORIS, 2014)
Plaquetas	290 ± 40	F	[0-40]	(BALDUINI; NORIS, 2014)
Plaquetas	690 ± 20	F	[40-60]	(BALDUINI; NORIS, 2014)
Plaquetas	240 ± 10	F	[60-99]	(BALDUINI; NORIS, 2014)
Albumina	4.6 ± 0.3	M	[0-20]	(WEAVING; BATSTONE; JONES, 2016)
Albumina	4.6 ± 0.2	M	[20-40]	(WEAVING; BATSTONE; JONES, 2016)
Albumina	4.4 ± 0.2	M	[40-60]	(WEAVING; BATSTONE; JONES, 2016)
Albumina	4.3 ± 0.3	M	[60-99]	(WEAVING; BATSTONE; JONES, 2016)
Albumina	4.4 ± 0.3	F	[0-20]	(WEAVING; BATSTONE; JONES, 2016)

Continuação da Tabela 8

Exame	**Valor Médio**	**Sexo**	**Faixa Etária**	**Referência**
Albumina	4.3 ± 0.2	F	[20-40]	(WEAVING; BATSTONE; JONES, 2016)
Albumina	4.2 ± 0.2	F	[40-60]	(WEAVING; BATSTONE; JONES, 2016)
Albumina	4.1 ± 0.2	F	[60-99]	(WEAVING; BATSTONE; JONES, 2016)
Bilirrubinas	6.2 ± 1.7	M	[0-20]	(ROSENTHAL; PINCUS; FINK, 1984)
Bilirrubinas	5.6 ± 1.5	M	[20-40]	(ROSENTHAL; PINCUS; FINK, 1984)
Bilirrubinas	5.3 ± 1.3	M	[40-60]	(ROSENTHAL; PINCUS; FINK, 1984)
Bilirrubinas	5.1 ± 1.7	M	[60-99]	(ROSENTHAL; PINCUS; FINK, 1984)
Bilirrubinas	4.4 ± 1.1	F	[0-20]	(ROSENTHAL; PINCUS; FINK, 1984)
Bilirrubinas	4.4 ± 1.6	F	[20-40]	(ROSENTHAL; PINCUS; FINK, 1984)
Bilirrubinas	4.0 ± 1.1	F	[40-60]	(ROSENTHAL; PINCUS; FINK, 1984)
Bilirrubinas	4.1 ± 1.2	F	[60-99]	(ROSENTHAL; PINCUS; FINK, 1984)
Creatinina	1.1 ± 0.2	M	[0-99]	(BURTIS; BRUNS, 2014)

Continuação da Tabela 8

Exame	**Valor Médio**	**Sexo**	**Faixa Etária**	**Referência**
Creatinina	0.9 ± 0.3	F	[0-99]	(BURTIS; BRUNS, 2014)
Tempo de Protrombina	1.0 ± 0.2	M/F	[0-99]	(PROVAN et al., 2009)

A utilização de uma distribuição normal como base para geração de valores pseudo-aleatórios para preenchimento dos atributos faltantes foi a opção escolhida pois estas alterações, quando aplicadas em uma pequena parcela dos registros, não altera globalmente a distribuição do conjunto (YUAN; BENTLER, 2010). A criação de dados sintéticos para a base CPTAC-PDA foi necessária para aproveitamento das imagens e por tratar-se de somente 17% dos pacientes, também não deve influenciar o resultado dos experimentos. O resultado final do pré-processamento de dados tabulares está disponível em <https://github.com/amenegotto/pyLiver/blob/master/csv/clinical_data.csv>.

4.1.3 Divisão das Bases de Dados

Após o pré-processamento das imagens e dos dados tabulares, duas bases de dados de exames de imagens foram criadas. A primeira possui imagens DICOM convertidas para PNG sem realce da nitidez e sem filtragem de ruído. A segunda possui imagens DICOM que passaram pelo processo de realce da nitidez e filtragem de ruído convertidas para o formato PNG. A criação e utilização destas duas bases de imagens servem para validar se as técnicas de realce de nitidez e filtragem de ruído efetivamente aumentam o desempenho do algoritmo de auxílio-diagnóstico de hepatocarcinoma proposto.

As bases de imagens criadas necessitam ser particionadas para realização do processo de treinamento e experimentação. A estratégia de particionamento dos dados utilizada nos experimentos é a validação *Holdout*, pois métodos de validação cruzada iterativos, apesar de comumente fornecerem um incremento no desempenho obtido, aumentam a duração e o custo computacional do processo de treinamento tornando-se inviáveis quando aplicados em projetos de curta duração e com grandes massas de dados (YADAV; SHUKLA, 2016).

Cada base de dados foi subdividida em conjuntos de treinamento, validação e testes. A base de treinamento é utilizada para extração de padrões e ao término de cada época, a base de validação é utilizada para verificar se a rede neural convolucional está convergindo, ou seja, se a rede está sendo capaz de aplicar o conhecimento adquirido durante o treinamento num novo conjunto de dados. Após o encerramento das épocas de treinamento, a base de testes é utilizada para medir o desempenho efetivo do algoritmo de aprendizado de máquina profundo desenvolvido em dados inéditos (CHOLLET, 2017c). A

divisão das bases foi realizada numa proporção 70% / 20% / 10%, comumente utilizada em trabalhos que aplicam técnicas de aprendizado de máquina profundo em bases de dados com grande quantidade e variabilidade de exemplos em cada classe (XUE et al., 2018) (EATON-ROSEN et al., 2018) (VANG; CHEN; XIE, 2018).

As bases de dados originais não são balanceadas, pois existe um número maior de imagens da classe negativa. Apesar de esta situação refletir a realidade, existe um consenso na literatura de que a utilização de bases desbalanceadas sem uma metodologia específica para tratar a falta de balanceamento das classes pode causar problemas no processo de treinamento de algoritmos de aprendizado de máquina profundo ocasionando um modelo de menor desempenho (BUDA; MAKI; MAZUROWSKI, 2018). Desta forma, optou-se por criar bases balanceadas com exatamente 20.792 imagens da classe positiva (com hepatocarcinoma) e 20.792 imagens da classe negativa (sem hepatocarcinoma), totalizando 41.584 imagens. O processo de remoção das imagens para equalização da quantidade de imagens em cada classe é conhecido como *undersampling* (JUNSOMBOON; PHIENTHRAKUL, 2017). Neste trabalho, a remoção de imagens da classe negativa foi realizada de forma proporcional entre as bases TCGA-STAD, TCGA-KIRP e CPTAC-PDA, ou seja, foi removido o mínimo de imagens em cada base para balancear a base de imagens. A escolha das imagens para remoção foi realizada de forma aleatória com base na ordem dos arquivos retornada pelo comando de listagem de arquivos do sistema operacional Linux.

A Tabela 9 ilustra a quantidade de imagens que compõe cada uma das partições.

Tabela 9 – Visão Geral dos Conjuntos de Dados.

Origem	**Treinamento**	**Validação**	**Testes**	**Classe**
TCGA-LIHC	14.552	4.158	2.082	Positivo
TCGA-KIRP	1.100	316	160	Negativo
TCGA-STAD	7.680	2.193	1.096	Negativo
CPTAC-PDA	5.772	1.649	826	Negativo
Totais	29.104	8.316	4.164	

A criação e a divisão das bases de dados de imagem foram realizadas programaticamente. O pseudo-código que ilustra o mecanismo utilizado para este fim pode ser visto no Algoritmo 1. A Figura 13 ilustra o tamanho de cada uma das bases de dados de imagens e suas respectivas partições. Os dados tabulares possuem tamanho de 22 KB e foram

utilizados sob demanda, conforme detalhado na Subseção 4.2.3.

Algoritmo 1: Pseudo-Código para Construção e Divisão das Bases de Imagens

Entradas : Caminho das imagens DICOM no Sistema Operacional (input_path), Realizar Pré-Processamento(is_pp)

Resultado : Bases de imagens prontas para uso

```
Criar Diretórios Destino;
Criar Arquivo CSV para Listagem Destino;
lista_arquivos_dicom = busca_images_dicom(input_path);
for f in lista_arquivos_dicom: do
    Abrir f;
    if is_pp then
        Aplicar CLAHE;
        Aplicar TV Denoising;
    else
        ;
    end
    Determinar Diretório;
    Criar Nome Arquivo;
    Salvar Imagem PNG;
    Adicionar Metadados da Imagem no CSV;
end
Calcular Quantidade de Imagens em Cada Base;
Zerar Contadores de Imagem por Subconjunto;
for i in csv_destino: do
    Ler metadados i;
    Determinar Caminho Destino;
    Mover i para Caminho Destino;
    Atualizar Contadores de Imagem por Subconjunto;
end
```

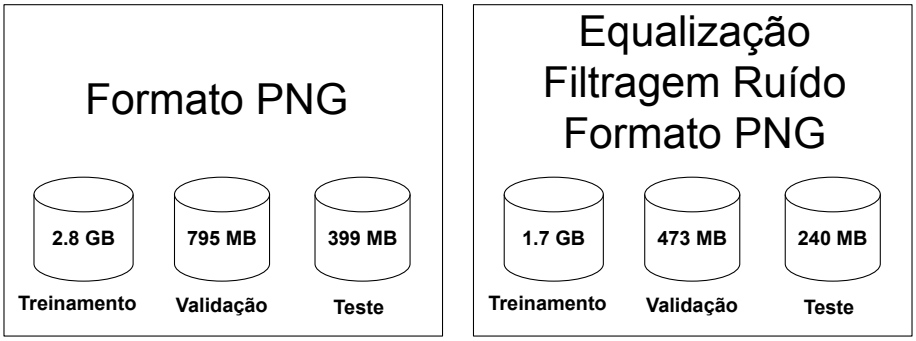


Figura 13 – Divisão das Bases de Imagens. Fonte: Do Autor

4.1.4 Armazenamento

A utilização das bases de imagens criadas na Seção 4.1.3 para treinamento e experimentação do algoritmo de aprendizado de máquina profundo para auxílio-diagnóstico de hepatocarcinoma requer acesso local (direto), visto que o tamanho das bases de imagem não permite sua utilização através de uma rede de computadores de forma performática.

Após as etapas descritas de pré-processamento e divisão dos dados, as bases de imagens foram comprimidas utilizando o comando *gzip* e transferidas para *buckets* privados do serviço Amazon S3, que permitem o *download* das bases de imagens em menos de 10 segundos a partir de máquinas virtuais criadas com os serviços de computação em nuvem fornecidos pela Amazon. O serviço Amazon S3 custa atualmente US$ 0.023/GB por mês e foi utilizado como *backup* durante todo o processo de criação e experimentação do algoritmo de aprendizado de máquina profundo para auxílio-diagnóstico computadorizado de hepatocarcinoma.

O meio de armazenamento primário utilizado para criação e experimentação da arquitetura proposta foi o Amazon *Elastic Block Store* (EBS). Através deste serviço é possível criar discos virtuais que podem ser anexados de forma transparente em servidores alocados no serviço de computação em nuvem da Amazon (EC2), facilitando assim a montagem do ambiente operacional utilizado e garantindo um acesso performático aos arquivos. Para as tarefas deste projeto, foi criado um disco virtual de 40 GB que possui os códigos-fontes, as bases de dados e os artefatos resultantes da execução dos experimentos. O custo atual do Amazon EBS é de US$ 0.10/GB por mês.

4.2 Aprendizado de Máquina Profundo

4.2.1 Técnicas de Regularização

A aplicação de técnicas de regularização em algoritmos de aprendizado de máquina profundo objetiva minimizar os erros de generalização sem afetar o processo de treinamento deste algoritmo (GOODFELLOW; BENGIO; COURVILLE, 2016e). Estas técnicas foram desenvolvidas em grande parte durante a evolução das redes neurais ocorrida no final do século XX e são utilizadas desde então em diferentes tipos de redes conexionistas como redes neurais convolucionais e redes recorrentes.

As técnicas descritas nas subseções que seguem foram utilizadas em todas as arquiteturas descritas neste capítulo. As exceções em relação à parametrização destas técnicas estão descritas de forma explícita nas respectivas descrições das arquiteturas.

4.2.1.1 Data Augmentation

A capacidade de generalização de uma rede convolucional está diretamente relacionada à quantidade e variedade de exemplos apresentados durante sua fase de treinamento, pois nesta etapa são extraídos os padrões e nuances pertinentes a cada uma das classes. A disponibilidade de infinitos exemplos em problemas de classificação de imagens é uma utopia principalmente quando trabalha-se com domínios específicos como, por exemplo, auxílio-diagnóstico de hepatocarcinoma (FRID-ADAR et al., 2018).

A utilização de redes convolucionais profundas com um conjunto limitado de imagens para treinamento requer a aplicação de uma técnica conhecida como *Data Augmentation* cujo objetivo é aumentar o conjunto de dados disponíveis para treinamento e validação sem perder a correlação entre os exemplos criados e os exemplos originais (GOODFELLOW; BENGIO; COURVILLE, 2016e). As abordagens mais comumente encontradas para a aplicação desta técnica são a criação de novas imagens e a inserção de ruído.

As operações de *Data Augmentation* utilizadas nesta dissertação para criação de novas imagens a partir do conjunto de imagens original foram desenvolvidas utilizando o próprio ferramental disponibilizado pelo Keras para este fim (CHOLLET, 2017d). São elas:

- Rotação: rotação pseudo-aleatória do eixo X da imagem em até 15°;
- Translação Horizontal: deslocamento horizontal pseudo-aleatório da imagem dentro de seu eixo X em até 20%;
- Translação Vertical: deslocamento vertical pseudo-aleatório da imagem dentro de seu eixo Y em até 20%;

- Cisalhamento (do inglês, *Shear*): Deformação pseudo-aleatória da imagem em até 10°;
- *Zoom*: Aumento ou decremento pseudo-aleatório do *zoom* em até 20%;
- Redimensionamento: redimensionamento pseudo-aleatório da imagem com escala 1/255.

A aplicação das operações de *Data Augmentation* deve levar em consideração o domínio das imagens. Em imagens de tomografia computadorizada para auxílio-diagnóstico de hepatocarcinoma com cortes axiais, existem variações angulares de rotação, ruído e diferenças de posicionamento dependentes do contexto (equipamento, operador, paciente). Entretanto, não existirão exemplos de imagens com rotação superior a 90°. Portanto, a escolha dos valores para parametrização dessas operações deve ser realizada com cautela para não prejudicar o treinamento com imagens deformadas ou incompatíveis ao conjunto de imagens originais.

As redes neurais convolucionais descritas neste capítulo utilizam *Data Augmentation* durante as etapas de treinamento e validação em todas arquiteturas experimentadas.

4.2.1.2 L^2 *Norm*

A regularização através da normalização matemática dos pesos de cada neurônio ajuda a mitigar a ocorrência de sobreajuste através da diminuição da entropia da distribuição destes valores, obtida através da aplicação de uma penalidade ao valor que mede a distância atual da solução ótima (*loss*) durante a fase de treinamento da rede. A normalização escolhida para as redes convolucionais trabalhadas nesta dissertação é conhecida como L^2Norm ou *Weight Decay*. (CHOLLET, 2017e).

A penalidade (valor do decréscimo) aplicada nas camadas convolucionais e densas das redes convolucionais profundas descritas neste capítulo é de 0,0005, valor comumente utilizado em redes convolucionais profundas estado da arte (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) (SZEGEDY et al., 2016) (CHOLLET, 2017f).

4.2.1.3 *Early Stopping*

A 'Parada Antecipada' (do inglês, *Early Stopping*) é uma técnica de regularização que monitora o desempenho da rede neural convolucional durante a etapa de validação de cada época, observando o histórico de uma métrica previamente escolhida. Com base na análise do histórico de valores desta métrica e na parametrização realizada, o processo de treinamento é cancelado quando não existe mais progresso no aprendizado (GOODFELLOW; BENGIO; COURVILLE, 2016e).

A coleta das métricas envolve também o armazenamento dos pesos dos neurônios. Durante a interrupção do treinamento, os pesos que obtiverem maior desempenho conforme a métrica escolhida são recarregados e reutilizados na etapa de teste da rede convolucional, que ocorre ao final das ciclos de treinamento e validação e de onde são efetivamente extraídas as métricas de desempenho para comparação das redes convolucionais.

A métrica escolhida para aplicação do *Early Stopping* é a Acurácia. A quantidade de épocas sem melhora efetiva da Acurácia na fase de validação varia conforme a classe da rede convolucional e o tipo de experimento realizado. A Subseção 4.3.4 descreve os parâmetros de *Early Stopping* utilizados em cada uma das arquiteturas e fases dos experimentos.

4.2.2 Arquiteturas Unimodais

As redes convolucionais profundas descritas nesta subseção utilizam exclusivamente imagens de tomografias computadorizada como insumo de entrada para auxílio-diagnóstico de hepatocarcinoma. A implementação e experimentação destas redes possibilita a extração das métricas que servirão como ponto de partida para o cálculo do impacto da adição de múltiplas modalidades de dados em redes convolucionais profundas para auxílio-diagnóstico computadorizado de hepatocarcinoma.

Estas arquiteturas podem ser agrupadas em duas classes distintas. A primeira classe contém duas redes convolucionais projetadas pelos autores desta pesquisa denominadas respectivamente RNCPU-HCC e RNCPU-HCC Light. A segunda classe contém três redes convolucionais estado da arte adaptadas para auxílio-diagnóstico de hepatocarcinoma. O restante desta subseção detalha as arquiteturas destas redes convolucionais.

4.2.2.1 RNCPU-HCC

A rede neural convolucional profunda RNCPU-HCC foi projetada pelos autores desta pesquisa para realização de auxílio-diagnóstico de hepatocarcinoma utilizando imagens de tomografia computadorizada como insumo de entrada.

A dimensão das imagens utilizadas como entrada desta rede é de 96×96 pixels com três canais de cores: *Red*, *Green* e *Blue* (RGB). Os pesos dos neurônios das camadas convolucionais e densas desta rede são inicializados com o algoritmo '*He*' (HE et al., 2015). Este algoritmo de inicialização de pesos é comumente utilizado em arquiteturas profundas sem treinamento prévio visando diminuir o tempo de convergência destas redes.

A quantidade de canais utilizada nas camadas convolucionais reflete diretamente a quantidade de características extraídas durante a aplicação destes filtros. Nos diagramas que ilustram este capítulo, esta informação pode ser vista nas camadas 'Conv', cuja nomenclatura possui o formato *Conv[tamanho do filtro convolucional]-[quantidade de canais]*. Nesta arquitetura, o componente para reduzir a dimensionalidade dos parâmetros

obtidos na saída das camadas convolucionais é o *MaxPooling* com filtro de tamanho 3×3 pixels e salto (do inglês, *stride*) de 1 pixel.

O algoritmo de otimização do gradiente descendente é o componente responsável pela busca da solução ótima, ou seja, descobrir quais são os valores paramétricos necessários para que a rede convolucional profunda atinja a melhor capacidade de generalização usando os insumos fornecidos durante o treinamento (GOODFELLOW; BENGIO; COURVILLE, 2016f). O algoritmo de otimização Adam (KINGMA; BA, 2015) é utilizado na rede convolucional RNCPU-HCC com taxa de aprendizado (do inglês, *Learning Rate*) aplicada de 0,00001. A utilização de um valor pequeno na taxa de aprendizado ajuda a mitigar o sobreajuste em redes convolucionais profundas, pois evita que a rede memorize o conjunto de treinamento sem aprender os padrões necessários para classificação de novas imagens.

A função de ativação utilizada em todas as camadas com exceção da camada de saída é a *Rectified Linear Unit* (ReLU) (NAIR; HINTON, 2010), que possui baixo custo computacional e excelente desempenho em redes neurais convolucionais profundas (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). A camada de saída da rede utiliza a função de ativação sigmoide, pois esta apresenta desempenho superior às demais funções de ativação em redes neurais profundas com saídas dicotômicas (SIBI; JONES; SIDDARTH, 2013).

A utilização da técnica denominada *Batch Normalization* (IOFFE; SZEGEDY, 2015) entre as camadas convolucionais e as funções de ativação objetiva diminuir o tempo de convergência do treinamento através da normalização das entradas dessas camadas a cada lote de imagens processadas. A aplicação de *Dropout* após as funções de ativação, técnica que aleatoriamente zera os pesos dos neurônios, também objetiva mitigar o sobreajuste e aumentar a capacidade de generalização da rede convolucional (GOODFELLOW; BENGIO; COURVILLE, 2016g).

O código-fonte que implementa a rede RNCPU-HCC pode ser visto no Apêndice A.10 e a Figura 14 ilustra um diagrama esquemático da arquitetura desta rede convolucional.

Figura 14 – Arquitetura RNCPU-HCC Unimodal. Fonte: Do Autor

4.2.2.2 RNCPU-HCC Light

A rede convolucional profunda RNCPU-HCC Light também foi projetada pelos autores desta pesquisa e portanto não possui nenhuma forma de conhecimento prévio. Esta rede é uma versão reduzida da arquitetura RNCPU-HCC projetada com o objetivo de diminuir o custo computacional e o tempo de treinamento desta rede convolucional sem perda da capacidade de generalização.

Todos os componentes e hiperparâmetros utilizados na rede convolucional RNCPU-HCC Light derivam da rede RNCPU-HCC. Apesar disso, a principal diferença destas redes convolucionais é sua arquitetura. A rede RNCPU-HCC possui uma quantidade menor de camadas convolucionais e um número maior de vetores de características em cada camada, fatores que aumentam o custo computacional e os cuidados para evitar o sobreajuste. A rede RNCPU-HCC Light por sua vez possui uma quantidade maior de camadas convolucionais com um número menor de vetores de características em cada camada.

O código-fonte que implementa a rede convolucional RNCPU-HCC Light pode ser visto no Apêndice A.11 e a Figura 15 ilustra um diagrama esquemático da arquitetura desta rede.

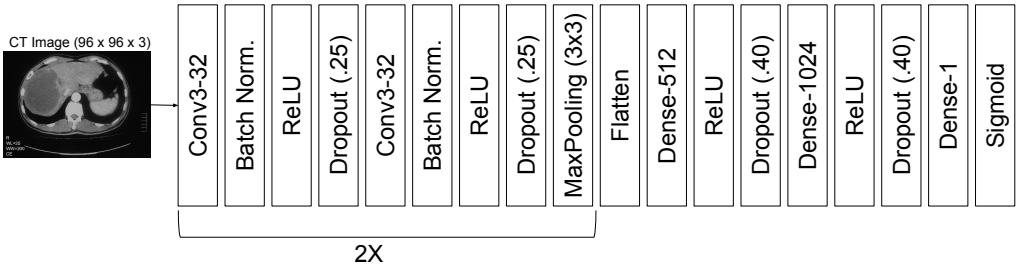


Figura 15 – Arquitetura RNCPU-HCC Light Unimodal. Fonte: Do Autor

4.2.2.3 InceptionV3

A rede convolucional profunda InceptionV3 foi criada pelo Google em 2014 (SZEGEDY et al., 2015). Esta rede já foi utilizada de forma promissora em problemas de classificação de imagens médicas e ganhou a atenção do público quando ultrapassou a precisão de dermatologistas humanos no auxílio-diagnóstico computadorizado de melanomas (ESTEVA et al., 2017). A utilização desta rede convolucional em problemas de classificação de imagens de domínios diferentes ao originalmente proposto demanda a modificação da camada densa e uma etapa de treinamento que refina o conhecimento previamente adquirido dessa rede para o contexto trabalhado.

O *framework* Keras traz uma implementação da rede convolucional InceptionV3 com os pesos do treinamento obtido com a base de imagens ImageNet. É possível utilizar esta implementação que já possui o conhecimento básico para reconhecimento de milhares

de objetos sem incluir a camada densa original e adaptá-la para classificação de imagens de outros domínios através da adição de um novo classificador. Recomenda-se realizar a adaptação do novo classificador em duas etapas de treinamento distintas: a primeira serve para ajustar os pesos da nova camada densa e a segunda serve para ajustar os pesos das camadas convolucionais já existentes.

A rede InceptionV3 utiliza imagens com dimensão de 299×299 pixels e três canais de cores (RGB) como insumo de entrada. O algoritmo de otimização do gradiente descendente utilizado é o *Stochastic Gradient Descendent* (SGD) (GOODFELLOW; BENGIO; COURVILLE, 2016f) com taxa de aprendizado de 0,0001 e *Momentum* de 0,9 visando pular pontos máximos locais (SUTSKEVER et al., 2013).

A função de ativação Softmax é utilizada na camada de saída da rede InceptionV3. Esta é a mesma função utilizada originalmente e recomenda-se não modificá-la em processos de transferência de conhecimento para não interferir nos padrões aprendidos em treinamentos prévios. A única diferença em relação à versão original é que somente duas classes são reconhecidas, hepatocarcinoma positivo ou negativo.

A arquitetura InceptionV3 adaptada para auxílio-diagnóstico de hepatocarcinoma exclusivamente com imagens de tomografia computadorizada pode ser vista na Figura 16 e o código-fonte desta implementação está disponível no Apêndice A.12. Na Figura 16, as funções de ativação ReLU das camadas convolucionais e as camadas de *Dropout* foram suprimidas por brevidade.

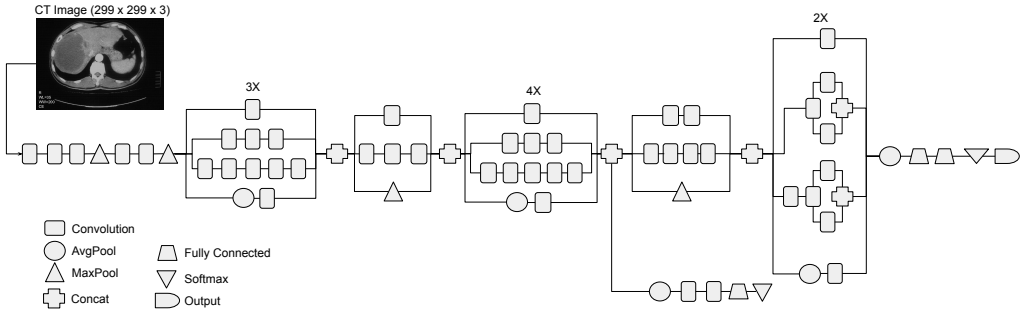


Figura 16 – Arquitetura InceptionV3 Unimodal. Fonte: Do Autor

4.2.2.4 Xception

A rede convolucional profunda Xception foi criada pelo Google em 2017 (CHOLLET, 2017f). A arquitetura da rede Xception é derivada da rede InceptionV3 e ambas possuem um custo computacional semelhante. A criação desta rede foi motivada por melhorias na utilização dos parâmetros da rede através da aplicação de camadas convolucionais paralelas e conexões residuais.

O criador e principal mantenedor do *framework* Keras, François Chollet, é funcionário do Google e autor desta rede convolucional profunda. A adaptação da rede Xception

para imagens de outros domínios é realizada da mesma maneira descrita na Subseção 4.2.2.3, passando pela adição de uma nova camada de classificação e duas etapas de treinamento.

A rede Xception também utiliza imagens com dimensão de $299 \times 299 \times 3$ e o algoritmo de otimização SGD. Os valores de taxa de aprendizado e *Momentum* também são idênticos aos valores da rede InceptionV3 descritos anteriormente.

A função de ativação Softmax é utilizada na camada de saída da rede Xception. Esta é a mesma função utilizada originalmente e recomenda-se não modificá-la em processos de transferência de conhecimento para não interferir nos padrões aprendidos em treinamentos prévios. A única diferença em relação à versão original é que somente duas classes são reconhecidas, hepatocarcinoma positivo ou negativo.

A arquitetura da rede convolucional Xception adaptada para auxílio-diagnóstico de hepatocarcinoma exclusivamente com imagens de tomografia computadorizada pode ser vista na Figura 17 e o código-fonte desenvolvido está disponível no Apêndice A.13. Nesta figura, as funções de ativação ReLU das camadas convolucionais e as camadas *Dropout* foram suprimidas por brevidade.

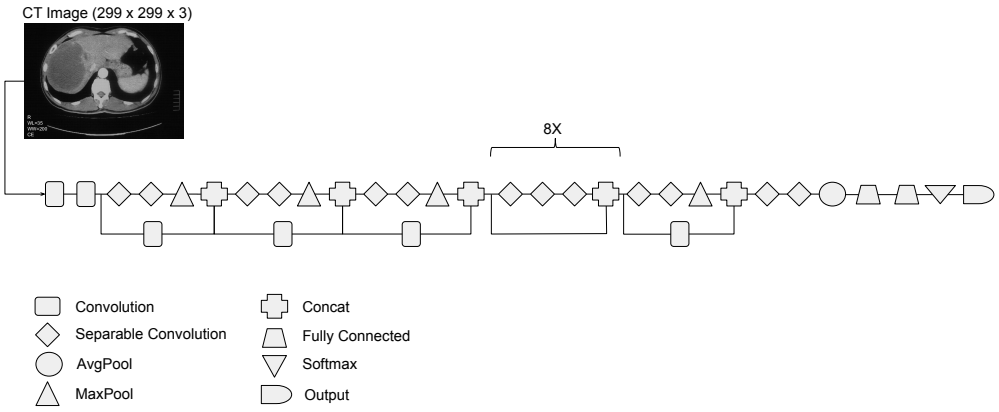


Figura 17 – Arquitetura Xception Unimodal. Fonte: Do Autor

4.2.2.5 VGG19

A rede convolucional profunda VGG19 é uma versão da rede VGGNet criada pelo *Visual Geometry Group* (VGG) da Universidade de Oxford (SIMONYAN; ZISSERMAN, 2015) em 2014. Esta rede é composta por 19 camadas sendo 16 camadas convolucionais e 3 camadas densas.

A adaptação da rede convolucional VGG19 é realizada da mesma maneira descrita nas subseções anteriores. A diferença é que a recomendação para readaptação dos pesos da camada de classificação adicionada seja realizada em uma etapa, fator que diminui o custo

computacional de modificação desta rede convolucional para imagens de outros domínios quando comparado com as redes InceptionV3 e Xception.

A rede VGG19 utiliza imagens com dimensão de 224×224 e três canais de cores (RGB). O algoritmo de otimização utilizado nesta arquitetura é o SGD com taxa de aprendizado = 0,002, por tratar-se de uma arquitetura de menor profundidade que as demais. Nesta rede não foi utilizado *Momentum*.

A função de ativação Softmax é utilizada na camada de saída da rede VGG19. Esta é a mesma função utilizada originalmente e recomenda-se não modificá-la em processos de transferência de conhecimento para não interferir nos padrões aprendidos em treinamentos prévios. A única diferença em relação à versão original é que somente duas classes são reconhecidas, hepatocarcinoma positivo ou negativo.

A arquitetura VGG19 adaptada para auxílio-diagnóstico de hepatocarcinoma utilizando exclusivamente imagens de tomografia computadorizada pode ser vista na Figura 18, que não possui as funções de ativação ReLU das camadas convolucionais por brevidade. O código-fonte desta arquitetura está disponível no Apêndice A.14.

Figura 18 – Arquitetura VGG19 Unimodal. Fonte: Do Autor

4.2.3 Arquiteturas Multimodais

As arquiteturas das redes convolucionais profundas multimodais projetadas neste trabalho são adaptações das arquiteturas unimodais descritas na Subseção 4.2.2, ou seja, utilizam os mesmos dados de entrada, algoritmos de otimização, hiperparâmetros e técnicas para mitigação de sobreajuste. Desta forma, é possível quantificar a diferença de desempenho obtido somente pelo acréscimo das múltiplas modalidades de dados para auxílio-diagnóstico de hepatocarcinoma utilizando redes convolucionais profundas.

O *framework* Keras possibilita a utilização de múltiplas entradas em uma rede neural convolucional profunda através de duas abordagens: na primeira, todos os dados devem ser alocados em memória antes do processamento e na segunda o próprio *framework* Keras utilizando uma classe customizada consegue carregá-los sob demanda. Após provas de conceito, optou-se pela utilização da segunda abordagem nesta dissertação devido ao melhor desempenho computacional obtido através da paralelização das operações e a economia de recursos de *hardware* para execução dos experimentos, especialmente memória de acesso aleatório (do inglês, *Random Access Memory* - RAM). Maiores detalhes sobre a implementação da classe customizada são fornecidos na Subseção 4.3.3.

As imagens de tomografia computadorizada e os 20 atributos tabulares utilizados como insumo de entrada para as arquiteturas profundas multimodais estão descritos em detalhe nas Subseções 4.1.2.1 e 4.1.2.2. A implementação das arquiteturas supracitadas de forma multimodal utiliza duas abordagens de fusão de dados para determinar qual possui maior precisão no auxílio-diagnóstico de hepatocarcinoma. O chaveamento entre as abordagens de fusão de dados é realizado através de parametrização no código-fonte realizada antes da execução dos experimentos.

As arquiteturas multimodais com fusão de dados intermediária processam as imagens de forma exclusiva até a saída das camadas de extração de características. Nas arquiteturas RNCPU-HCC, RNCPU-HCC Light e VGG19 as características são convertidas para uma matriz linha pela camada *Flatten* e nas arquiteturas InceptionV3 e Xception esta tarefa é feita pela camada *Global Average Pooling*. A partir deste ponto as imagens estão no mesmo formato de representação dos dados tabulares e as múltiplas modalidades de dados podem ser concatenadas e utilizadas pela camada densa para classificação. As Figuras 19, 20, 21, 22 e 23 ilustram as arquiteturas das redes convolucionais profundas multimodais com fusão intermediária experimentadas nesta dissertação.

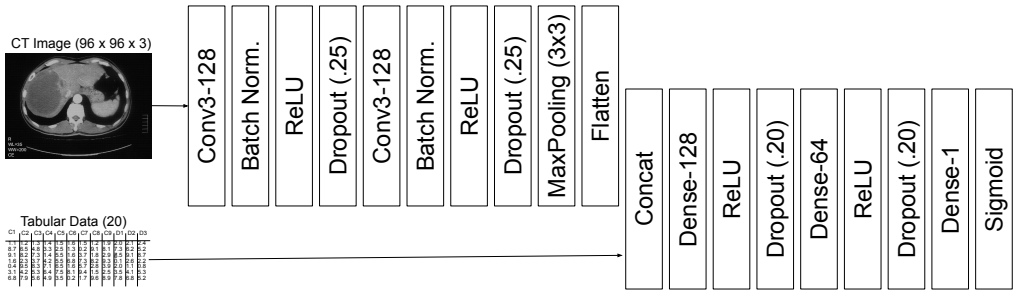


Figura 19 – Arquitetura RNCPU-HCC Multimodal com Fusão Intermediária. Fonte: Do Autor

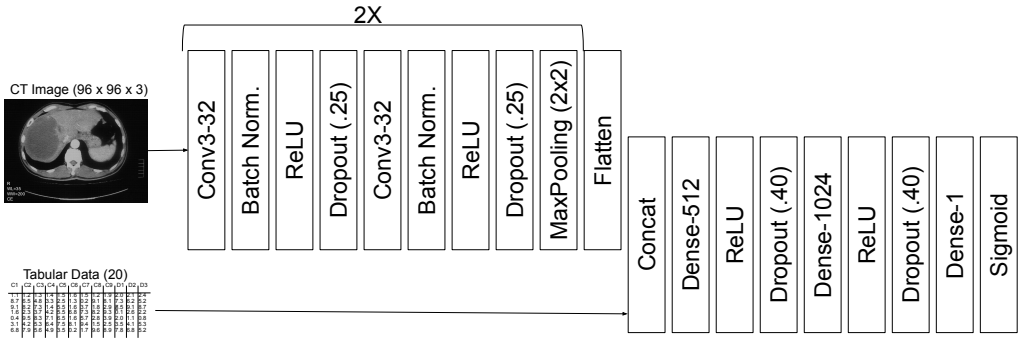


Figura 20 – Arquitetura RNCPU-HCC Light Multimodal com Fusão Intermediária. Fonte: Do Autor

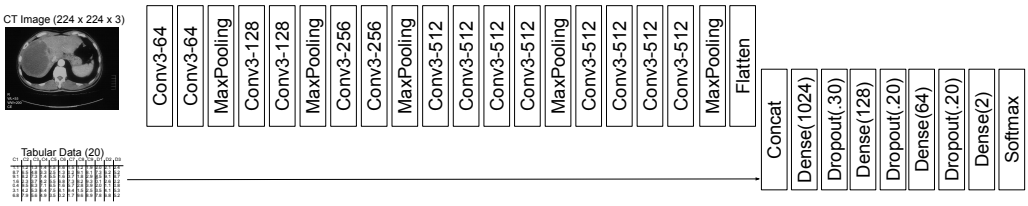


Figura 21 – Arquitetura VGG19 Multimodal com Fusão Intermediária. Fonte: Do Autor

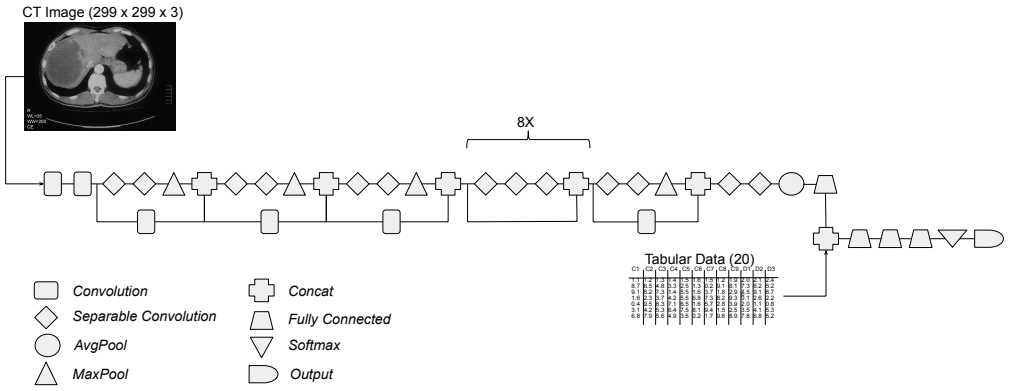


Figura 22 – Arquitetura Xception Multimodal com Fusão Intermediária. Fonte: Do Autor

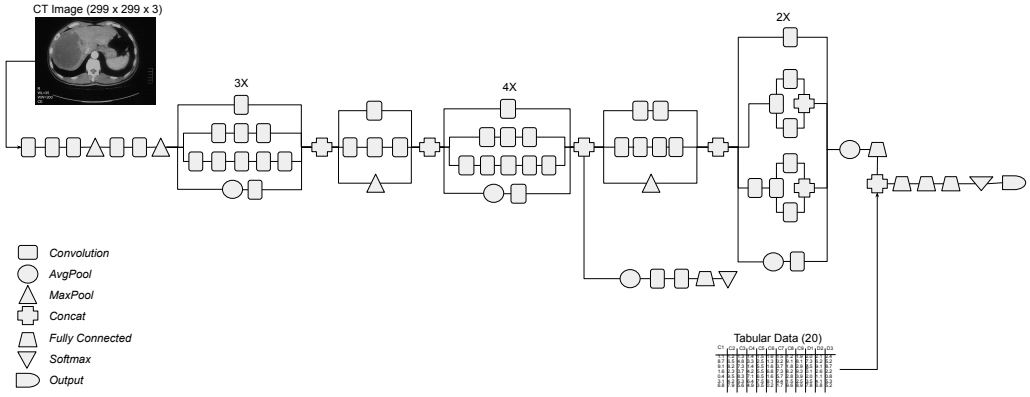


Figura 23 – Arquitetura InceptionV3 Multimodal com Fusão Intermediária. Fonte: Do Autor

A fusão de dados tardia implica que o processamento das imagens e atributos tabulares ocorra de maneira isolada e o resultado obtido em cada um dos processamentos seja concatenado para enfim determinar a possível existência de um hepatocarcinoma nos insumos de entrada da rede convolucional. As camadas densas utilizadas nestas redes convolucionais variam de acordo com a quantidade de parâmetros entregues pelas camadas de extração de características. As Figuras 24, 25, 26, 27 e 28 ilustram as arquiteturas das redes convolucionais profundas multimodais com fusão tardia experimentadas nesta dissertação.

O código-fonte das arquiteturas multimodais experimentadas nesta dissertação está disponível nos Anexos A.15, A.16, A.17, A.18 e A.19.

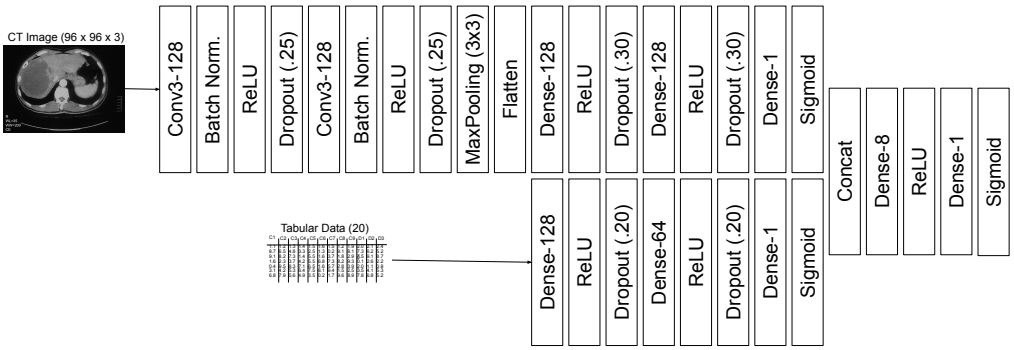


Figura 24 – Arquitetura RNCPU-HCC Multimodal com Fusão Tardia. Fonte: Do Autor

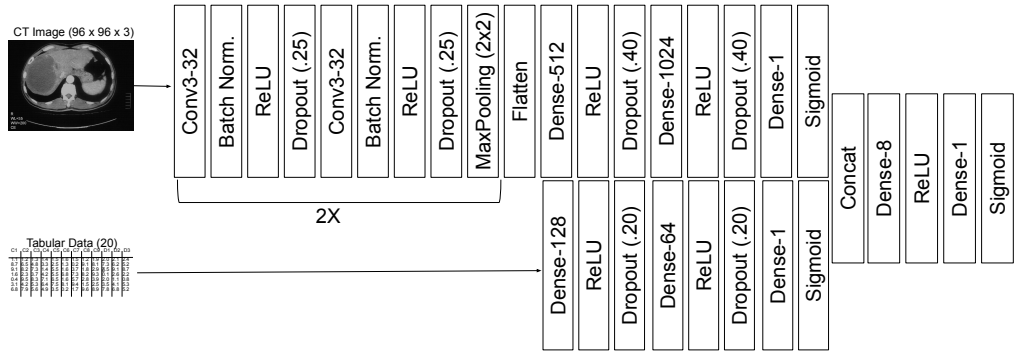


Figura 25 – Arquitetura RNCPU-HCC Light Multimodal com Fusão Tardia. Fonte: Do Autor

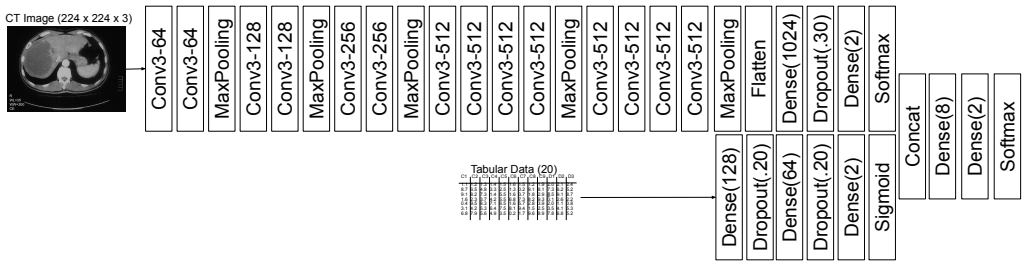


Figura 26 – Arquitetura VGG19 Multimodal com Fusão Tardia. Fonte: Do Autor

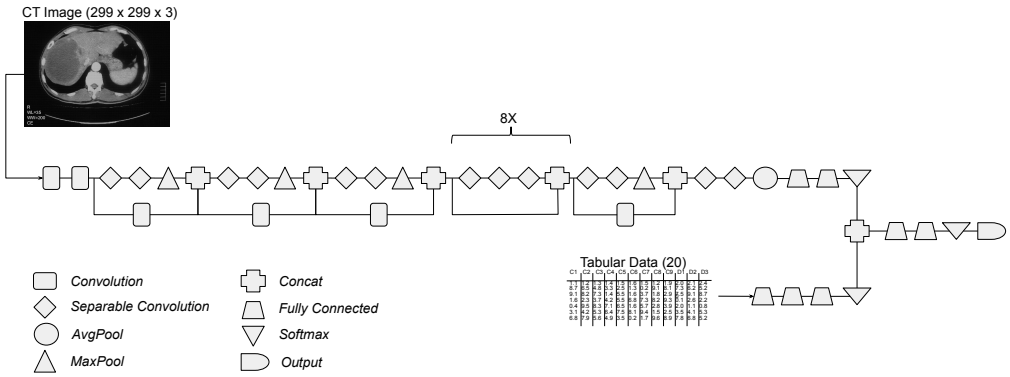


Figura 27 – Arquitetura Xception Multimodal com Fusão Tardia. Fonte: Do Autor

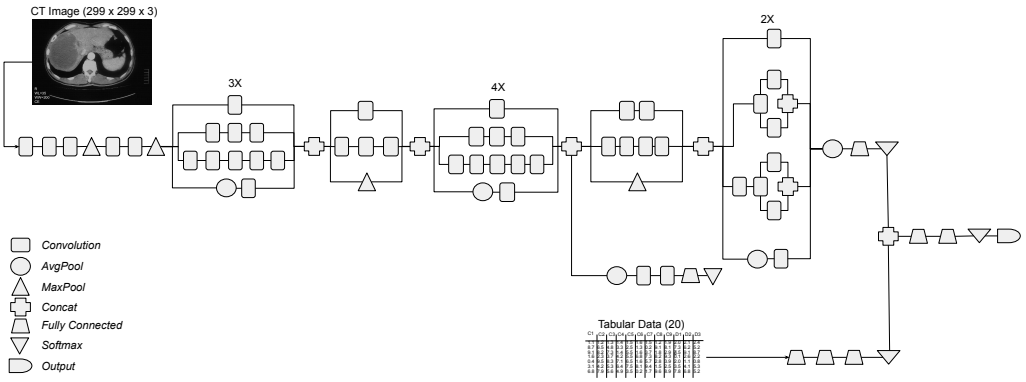


Figura 28 – Arquitetura InceptionV3 Multimodal com Fusão Tardia. Fonte: Do Autor

4.3 Execução dos Experimentos

4.3.1 Ambiente Operacional

Os equipamentos utilizados para execução dos experimentos são máquinas virtuais contratadas do serviço Amazon EC2. O sistema operacional escolhido é o Linux utilizando a distribuição Ubuntu 18.04. Os componentes para interação com *General Processing Units* (GPUs) NVIDIA são pré-configurados e o *framework* Keras, ao identificar estes componentes, também configura-se automaticamente para utilizar a GPU para realização das operações matemáticas matriciais.

A imagem de disco utilizada para criação das instâncias contratadas denomina-se *Deep Learning AMI* versão 23. Esta imagem traz o ferramental necessário para desenvolvimento de soluções de aprendizado de máquina utilizando MXNet-1.4, TensorFlow-1.14, PyTorch-1.1, Keras-2.2, Chainer-6.1, Caffe/2-0.8, Theano-1.0 e CNTK-2.7.

A implementação dos artefatos de *software* criados nesta dissertação foi realizada com a linguagem de programação Python versão 3.6. O *framework* Keras-2.2 operando

sobre o TensorFlow-1.14 foi utilizado para a implementação e experimentação das redes convolucionais profundas.

Durante a fase de implementação e provas de conceito foram utilizadas instâncias EC2 do tipo *p2.xlarge*, equipadas com 1.75 ECUs, 4 vCPUs 2.7 GHz Intel E5-2686v4, 61 GB RAM, GPU NVIDIA Tesla K80 com 12 GB, 1 disco SSD de 75 GB para o sistema operacional e 1 disco SSD de 40 GB (EBS) para as bases de dados, códigos-fonte e resultados dos experimentos. Atualmente, o custo nominal destas instâncias é de US$ 0.90/hora, mas é possível obter uma redução de preço utilizando instâncias criadas com *Spot Requests*, que possuem custo variável na faixa entre US$ 0.20 - US$ 0.30 / hora.

Os experimentos descritos na Subseção 4.3.4 foram realizados com instâncias EC2 do tipo *p3.2xlarge*, equipadas com 23.5 ECUs, 8 vCPUs 2.7 GHz Intel Xeon E5-2686 v4, 61 GB RAM, GPU NVIDIA Tesla V100 com 16 GB, 1 disco SSD de 75 GB para o sistema operacional e 1 disco SSD de 40 GB (EBS) para as bases de dados, códigos-fonte e resultados dos experimentos. O custo nominal destas instâncias atualmente é de US$ 3.06/hora, mas é possível obter uma redução do valor pago utilizando instâncias criadas com *Spot Requests*, que possuem custo variável entre US$ 0.70 - US$ 1.40 / hora.

4.3.2 Métricas de Desempenho

O progresso do aprendizado e o grau de generalização de um algoritmo de classificação desenvolvido com técnicas de aprendizado de máquina podem ser medidos através de métricas de desempenho próprias para este tipo de problema (THARWAT, 2018).

A métrica definida para balizar o progresso do aprendizado durante as fases de treinamento e validação das redes convolucionais profundas para auxílio-diagnóstico de hepatocarcinoma desenvolvidas nesta dissertação é a Acurácia (do inglês, *Accuracy*), cuja medida revela a razão entre as classificações realizadas corretamente e o total de classificações realizadas. A fórmula utilizada pelo *framework* Keras para medir a acurácia de um modelo pode ser vista na Equação 4.1. Os valores obtidos nestas fases não foram utilizados para comparar as arquiteturas experimentadas nesta dissertação.

$$acur = \frac{(VP + VN)}{(VP + VN + FP + FN)} \tag{4.1}$$

As métricas utilizadas para comparar as diferentes abordagens arquiteturais no auxílio-diagnóstico de hepatocarcinoma a fim de identificar a abordagem com maior desempenho neste tipo de tarefa foram coletadas durante a fase de testes dos experimentos. O restante desta seção descreve resumidamente as métricas coletadas para esse fim.

A matriz de confusão de um problema de classificação binário é uma matriz quadrada 2×2 que contém o resultado das predições realizadas durante a fase de testes. As

ocorrências de Verdadeiros Positivos (VP), Verdadeiros Negativos (VN), Falsos Positivos (FP) e Falsos Negativos (FN) são dispostas conforme a Figura 29:

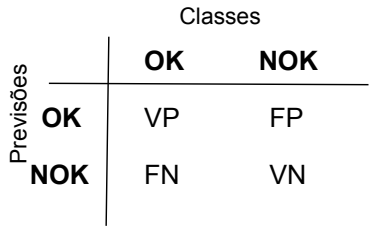


Figura 29 – Matriz de Confusão para Problemas de Classificação Binário. Fonte: Do Autor

A Acurácia, previamente definida na Equação 4.1, também foi coletada na fase de testes dos experimentos para comparar as arquiteturas experimentadas nesta dissertação.

A Precisão (do inglês, *Precision*), também conhecida como Valor Preditivo Positivo (VPP), mede a proporção de amostras positivas classificadas corretamente em relação ao total de classificações realizada. A fórmula utilizada para cálculo da precisão é:

$$prec = \frac{(VP)}{(VP + FP)} \tag{4.2}$$

A revocação (do inglês, *Recall*), também conhecida como sensibilidade, mede a proporção de casos positivos classificados corretamente em relação ao total de amostras positivas existentes e é calculada pela fórmula a seguir:

$$revoc = \frac{(VP)}{(VP + FN)} \tag{4.3}$$

O F-Score mede a qualidade do modelo combinando a precisão e a revocação em uma única métrica. O valor F-Score de um modelo é calculado pela fórmula:

$$F - Score = \frac{(2 * prec * revoc)}{(prec + revoc)} \tag{4.4}$$

O Valor Kappa mede o grau de concordância entre classificadores distintos trabalhando com o mesmo conjunto de dados e é calculado pela fórmula

$$\kappa = \frac{(p_o - p_e)}{(1 - p_e)} \tag{4.5}$$

O *AUC Score* mede a área sob a curva ROC (*Receiver Operating Characteristic*). O cálculo do *AUC Score* pode ser realizado com duas abordagens. A abordagem utilizada pelo *framework* Scikit-Learn (PEDREGOSA et al., 2011) é a construção da curva ROC

utilizando trapézios de forma que a área da curva ROC é dada pela soma da área destes trapézios.

Com exceção da Acurácia, todas as demais métricas são calculadas utilizando funções do *framework* Scikit-Learn versão 0.21.3.

4.3.3 Artefatos Construídos

Os artefatos de *software* construídos nesta dissertação objetivam automatizar e facilitar a realização de tarefas. Ao longo do texto, além da implementação das redes convolucionais profundas, diversos artefatos que realizam tarefas específicas já foram citados. O restante desta subseção descreve os artefatos utilizados essencialmente para execução dos experimentos de maneira ágil e segura, minimizando a interferência humana na execução e coleta das métricas.

4.3.3.1 Padrão de Construção dos Experimentos

O *framework* Keras facilita a implementação de redes neurais convolucionais profundas pois abstrai conceitos e operações matemáticas envolvidas na construção deste tipo de rede conexionista. Apesar das facilidades fornecidas, a execução e a coleta de métricas de repetidos experimentos torna-se uma tarefa morosa e propensa à erros. A construção de funções genéricas visando reuso de código-fonte e uma arquitetura que permita a rápida parametrização dos experimentos objetiva agilizar a execução dos experimentos e mitigar os erros de coleta.

Dentre os artefatos construídos, a classe que representa a execução de um experimento denomina-se ExecutionAttribute. Os atributos desta classe armazenam a arquitetura da rede convolucional, a parametrização dos experimentos, o estado atual do experimento e os resultados obtidos. Maiores detalhes sobre os atributos dessa classe podem ser vistos no código-fonte disponibilizado no Apêndice A.20.

A construção do código-fonte dos experimentos segue um padrão básico definido

pelos autores deste projeto e descrito no Algoritmo 2 em pseudocódigo.

Algoritmo 2: Padrão para Construção de Redes Convolucionais Multimodais

Entradas : Imagem de Tomografia Computadorizada; Atributos Antropométricos, Clínicos e Sociodemográficos
Resultado : Rede Convolucional Multimodal Treinada para Auxílio-Diagnóstico de Hepatocarcinoma

```
Importar Bibliotecas;
Configurar Ambiente;
Parametrizar Experimentos;
for i in range(0, CYCLES): do
    Definir Base da Arquitetura;
    if INTERMEDIATE_FUSION then
        Realizar Complemento Arquitetura;
    else
        if LATE_FUSION then
            Realizar Complemento Arquitetura;
        else
            Mostrar Mensagem de Erro;
        end
    end
    Gerar Diagrama da Arquitetura (PNG);
    Salvar Modelo;
    Configurar e Compilar Modelo;
    Criar e Parametrizar Geradores;
    Configurar Early Stop;
    Configurar Callback de Checkpoint;
    Realizar Treinamento;
    Recarregar Melhores Pesos;
    Realizar Testes;
    Gerar Relatórios;
    Limpar Ambiente;
end
Copiar Arquivos para S3;
```

4.3.3.2 Relatórios

Os relatórios gerados durante as execuções dos experimentos contêm dados coletados em todas as fases do processo, começando pela parametrização dos experimentos, seguindo durante a construção e treinamento das redes convolucionais profundas e terminando com a coleta de métricas na fase de testes. Através destes artefatos é possível determinar com exatidão em todos os experimentos a parametrização utilizada e visualizar: 1) a arquitetura da rede convolucional utilizada; 2) o desempenho das fases de treinamento e validação (acurácia e *loss*); 3) a matriz de confusão; 4) as métricas das predições realizadas na fase de testes da rede.

Os arquivos gerados durante os ciclos que compõem um experimento (modelo, melhores pesos e relatórios) são armazenados localmente no disco EBS da Amazon EC2 e para fins de *backup* são copiados para um *bucket* Amazon S3. O código-fonte da função genérica que implementa os testes e a coleta de métricas pode ser visto no Apêndice A.21. As funções implementadas neste arquivo foram utilizadas em todos os experimentos

realizados para automatizar o processo e unificar a coleta das métricas.

4.3.3.3 MultimodalGenerator

O *framework* Keras utiliza um tipo de classe conhecido como 'gerador' (do inglês, *Generator*) que responsabiliza-se pela entrega de insumos de entrada em lotes sob demanda para que não seja preciso carregar toda a base de dados na memória RAM durante o processo de treinamento, validação e testes de um algoritmo de aprendizado de máquina. Além da economia de memória, os geradores permitem a paralelização do treinamento da rede conforme o número de processadores e *cores* disponíveis no *hardware* utilizado de forma implícita diminuindo a duração destas tarefas (CHOLLET, 2017d).

A implementação de redes convolucionais profundas multimodais no *framework* Keras exige a criação de um gerador customizado visto que os geradores fornecidos pelo Keras suportam apenas entradas unimodais. O gerador construído para carregar imagens e dados tabulares sob demanda para este projeto denomina-se 'MultimodalGenerator'. Além das tarefas básicas, este gerador também implementa a entrega de imagens com *Data Augmentation* e faz o relacionamento entre as múltiplas modalidades de dados através do ID do Paciente. O gerador criado é totalmente parametrizável e a opção de *Data Augmentation* foi implementada utilizando as funções fornecidas pelo próprio Keras, de forma que os geradores sejam utilizados pelas arquiteturas unimodais e multimodais com a mesma parametrização para construção de novas imagens. O código-fonte do 'MultimodalGenerator' está disponível no Apêndice A.22.

4.3.4 Descrição dos Experimentos

A experimentação das redes convolucionais profundas descritas nas Subseções 4.2.2 e 4.2.3 foi realizada em duas etapas distintas. A primeira etapa teve por objetivo identificar as arquiteturas profundas unimodais e multimodais com melhor desempenho para auxílio-diagnóstico de hepatocarcinoma. Durante a segunda etapa foram realizados testes com as arquiteturas com maior desempenho na primeira etapa para extração de métricas visando responder a questão de pesquisa deste projeto. As Subseções 4.3.4.1 e 4.3.4.2 descrevem os detalhes dos experimentos realizados em cada uma destas fases.

4.3.4.1 Fase 1: Experimentos Preliminares

A primeira fase de experimentos buscou determinar qual arquitetura obtém maior desempenho no auxílio-diagnóstico de hepatocarcinoma dentre as arquiteturas propostas nesta dissertação. Para isto, foram realizadas múltiplas rodadas de experimentos utilizando as redes unimodais, as redes multimodais com fusão intermediária e as redes multimodais com fusão tardia.

A validação do pré-processamento das imagens também foi testada nesta fase, portanto as três abordagens arquiteturais foram experimentadas utilizando as duas bases de imagens criadas conforme descritos na Seção 4.1.3. Em resumo, a primeira contém imagens que passaram somente pelo processo de compressão através da conversão de formatos (DICOM => PNG) e a segunda contém as imagens que passaram pelo processo de realce de nitidez, filtragem de ruídos e compressão.

A Tabela 10 descreve os parâmetros utilizados nos experimentos desta fase. Foram realizados experimentos com as 5 arquiteturas utilizando as três formas de fusão de dados com as duas bases de imagens criadas, totalizando 30 experimentos.

Tabela 10 – Parametrização dos Experimentos Preliminares.

Arquitetura	**Qtd Épocas**	***Batch Size***	***Patience***	**Ciclos**
RNCPU-HCC	100	128	20	20
RNCPU-HCC Light	100	128	20	20
Xception	50	128	10	5
InceptionV3	50	128	10	5
VGG19	50	128	10	5

As redes convolucionais profundas sem treinamento prévio necessitam maior tempo de treinamento e uma quantidade maior de execuções para demonstração da robustez e estabilidade dos resultados obtidos. Além destes fatores, parametrizou-se o *Early Stopping* (parâmetro *Patience* da Tabela 10) com uma quantidade maior de épocas neste tipo de rede visto que a ausência de treinamento prévio demanda um tempo maior para ajuste dos pesos e estabilização da rede.

As redes convolucionais profundas do atual estado da arte modificadas para auxílio-diagnóstico de hepatocarcinoma são mais estáveis e portanto necessitam de um número menor de épocas de treinamento e de experimentos para demonstração da robustez e do poder que essas arquiteturas possuem. A diferença destes casos é que o treinamento das redes InceptionV3 e Xception é feito em duas etapas, fato que eleva a duração dos experimentos neste tipo de arquitetura.

4.3.4.2 Fase 2: Experimentos de Verificação

A realização dos experimentos nesta segunda fase tem por objetivo testar os limites das redes convolucionais profundas para auxílio-diagnóstico de hepatocarcinoma propostas nesta dissertação. Durante os experimentos desta etapa são extraídas métricas para determinar se as abordagens multimodais com exames de imagem, resultados de exames laboratoriais, atributos antropométricos e dados sociodemográficos obtém desempenho maior que as abordagens unimodais que utilizam somente os exames de imagem como insumo de entrada no auxílio-diagnóstico de hepatocarcinoma.

Os experimentos desta fase são realizados através de um único ciclo de treinamento/validação com 500 épocas, *Batch Size* de 128 exemplos e *Patience* de 50 épocas seguido da fase de testes e coleta de métricas. As três melhores arquiteturas multimodais foram comparadas com suas respectivas versões unimodais totalizando 6 experimentos que ao final entregam os subsídios necessários para a resposta da questão de pesquisa deste projeto.

5 Resultados

5.1 Experimentos Preliminares

Os resultados obtidos com as diferentes abordagens de construção de redes convolucionais profundas na primeira fase dos experimentos estão descritos nas subseções que seguem. Estes resultados serviram de subsídio para as decisões tomadas em relação às arquiteturas e estratégias de fusão de dados utilizadas na segunda fase de experimentos e que resultaram no algoritmo de auxílio-diagnóstico de hepatocarcinoma proposto.

5.1.1 Arquiteturas Unimodais

As redes convolucionais profundas unimodais foram experimentadas conforme descrito na Subseção 4.3.4. Os resultados obtidos nestes experimentos servem como parâmetro inicial de comparação para determinar o impacto da aplicação de abordagens multimodais no auxílio-diagnóstico de hepatocarcinoma utilizando aprendizado de máquina profundo.

As Tabelas 11, 12, 13, 14 e 15 descrevem os resultados dos experimentos preliminares realizados com as arquiteturas unimodais. Nas células das tabelas, os valores listados a esquerda do ponto-e-vírgula referem-se aos experimentos com imagens de tomografia computadorizada que não passaram por filtragem de ruído e realce de nitidez e os valores situados a direita do ponto-e-vírgula referem-se aos experimentos com imagens que passaram pela filtragem de ruído e realce de nitidez.

Tabela 11 – Resultados da Arquitetura RNCPU-HCC Unimodal.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.554; 0.536	0.037; 0.038	0.465; 0.473	0.607; 0.597	0.552; 0.546
Precisão	0.562; 0.536	0.042; 0.044	0.465; 0.442	0.627; 0.599	0.567; 0.551
Revocação	0.554; 0.536	0.037; 0.038	0.465; 0.473	0.607; 0.597	0.552; 0.546
F-Score	0.539; 0.520	0.038; 0.055	0.461; 0.382	0.592; 0.595	0.552; 0.546
Valor Kappa	0.108; 0.072	0.0743; 0.077	-0.0682; -0.053	0.215; 0.195	0.104; 0.092

Tabela 12 – Resultados da Arquitetura RNCPU-HCC Light Unimodal.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.539 ; 0.482	0.037 ; 0.028	0.494 ; 0.417	0.620 ; 0.526	0.545 ; 0.486
Precisão	0.561 ; 0.466	0.052 ; 0.044	0.494 ; 0.379	0.692 ; 0.530	0.563 ; 0.474
Revocação	0.539 ; 0.482	0.037 ; 0.028	0.494 ; 0.417	0.620 ; 0.526	0.545 ; 0.486
F-Score	0.489 ; 0.436	0.063 ; 0.049	0.381 ; 0.337	0.581 ; 0.512	0.545 ; 0.486
Valor Kappa	0.078 ; -0.03	0.074 ; 0.057	-0.01 ; -0.16	0.241 ; 0.053	0.090 ; -0.02

Tabela 13 – Resultados da Arquitetura Xception Unimodal.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.507 ; 0.549	0.043 ; 0.033	0.464 ; 0.516	0.566 ; 0.600	0.496 ; 0.548
Precisão	0.506 ; 0.575	0.050 ; 0.041	0.460 ; 0.522	0.577 ; 0.629	0.480 ; 0.561
Revocação	0.507 ; 0.549	0.043 ; 0.033	0.464 ; 0.516	0.566 ; 0.600	0.496 ; 0.548
F-Score	0.472 ; 0.511	0.073 ; 0.075	0.367 ; 0.400	0.550 ; 0.595	0.496 ; 0.548
Valor Kappa	0.015 ; 0.099	0.087 ; 0.066	-0.07 ; 0.032	0.133 ; 0.200	-0.00 ; 0.097

Tabela 14 – Resultados da Arquitetura VGG19 Unimodal.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.603 ; 0.477	0.027 ; 0.030	0.560 ; 0.444	0.634 ; 0.520	0.605 ; 0.473
Precisão	0.604 ; 0.473	0.026 ; 0.033	0.561 ; 0.441	0.634 ; 0.520	0.607 ; 0.467
Revocação	0.603 ; 0.477	0.027 ; 0.030	0.560 ; 0.444	0.634 ; 0.520	0.605 ; 0.473
F-Score	0.601 ; 0.464	0.028 ; 0.039	0.557 ; 0.427	0.634 ; 0.519	0.605 ; 0.473
Valor Kappa	0.206 ; -0.04	0.054 ; 0.061	0.120 ; -0.11	0.269 ; 0.040	0.210 ; -0.05

Tabela 15 – Resultados da Arquitetura InceptionV3 Unimodal.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.613 ; 0.524	0.015 ; 0.011	0.595 ; 0.515	0.630 ; 0.543	0.621 ; 0.522
Precisão	0.670 ; 0.525	0.016 ; 0.012	0.647 ; 0.516	0.690 ; 0.546	0.667 ; 0.523
Revocação	0.613 ; 0.524	0.015 ; 0.011	0.595 ; 0.515	0.630 ; 0.543	0.621 ; 0.522
F-Score	0.579 ; 0.516	0.030 ; 0.012	0.544 ; 0.504	0.604 ; 0.536	0.621 ; 0.522
Valor Kappa	0.227 ; 0.048	0.030 ; 0.022	0.191 ; 0.030	0.260 ; 0.087	0.243 ; 0.045

A análise isolada do desempenho das arquiteturas unimodais não contribui para os objetivos desta pesquisa, mas é possível observar que a aplicação da filtragem de ruído e realce de nitidez não é necessária no algoritmo final visto que somente a arquitetura Xception obteve melhora de desempenho com a aplicação do pré-processamento. Entretanto, os resultados obtidos pelas abordagens multimodais ainda nos experimentos preliminares podem invalidar esta afirmação.

5.1.2 Arquiteturas Multimodais

As redes convolucionais profundas multimodais foram experimentadas conforme descrito na Seção 4.3.4. O resultado destes experimentos embasou a tomada de decisão em relação a arquitetura e estratégia de fusão de dados escolhidas para compor a segunda etapa de experimentos.

5.1.2.1 Fusão Intermediária

As Tabelas 16, 17, 18, 19 e 20 descrevem as métricas coletadas durante os experimentos preliminares realizados com as arquiteturas multimodais utilizando fusão de

dados intermediária. Nas células das tabelas, os valores listados a esquerda do ponto-e-vírgula referem-se aos experimentos com imagens de tomografia computadorizada que não passaram por filtragem de ruído e realce de nitidez e os valores listados a direita do ponto-e-vírgula referem-se aos experimentos com imagens que passaram pela filtragem de ruído e realce de nitidez.

Tabela 16 – Resultados da Arquitetura RNCPU-HCC com Fusão Intermediária.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.621 ; 0.591	0.056 ; 0.053	0.531 ; 0.469	0.762 ; 0.675	0.624 ; 0.603
Precisão	0.646 ; 0.607	0.074 ; 0.064	0.531 ; 0.468	0.782 ; 0.689	0.642 ; 0.630
Revocação	0.621 ; 0.591	0.056 ; 0.053	0.531 ; 0.469	0.762 ; 0.675	0.624 ; 0.603
F-Score	0.609 ; 0.580	0.055 ; 0.052	0.531 ; 0.468	0.759 ; 0.672	0.608 ; 0.580
Valor Kappa	0.243 ; 0.183	0.112 ; 0.107	0.063 ; -0.06	0.524 ; 0.350	0.248 ; 0.206

Tabela 17 – Resultados da Arquitetura RNCPU-HCC Light com Fusão Intermediária.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.655 ; 0.668	0.057 ; 0.115	0.539 ; 0.503	0.774 ; 0.824	0.665 ; 0.716
Precisão	0.671 ; 0.698	0.075 ; 0.125	0.539 ; 0.506	0.826 ; 0.870	0.677 ; 0.725
Revocação	0.655 ; 0.668	0.057 ; 0.115	0.539 ; 0.503	0.774 ; 0.824	0.665 ; 0.716
F-Score	0.649 ; 0.647	0.055 ; 0.139	0.539 ; 0.353	0.774 ; 0.819	0.660 ; 0.706
Valor Kappa	0.310 ; 0.337	0.115 ; 0.230	0.079 ; 0.007	0.548 ; 0.649	0.331 ; 0.432

Tabela 18 – Resultados da Arquitetura Xception com Fusão Intermediária.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.854 ; 0.868	0.031 ; 0.002	0.798 ; 0.865	0.869 ; 0.869	0.869 ; 0.869
Precisão	0.885 ; 0.895	0.020 ; 0.003	0.849 ; 0.889	0.896 ; 0.896	0.896 ; 0.896
Revocação	0.854 ; 0.868	0.031 ; 0.002	0.798 ; 0.865	0.869 ; 0.869	0.869 ; 0.869
F-Score	0.851 ; 0.866	0.034 ; 0.002	0.790 ; 0.863	0.867 ; 0.867	0.867 ; 0.867
Valor Kappa	0.708 ; 0.737	0.063 ; 0.004	0.596 ; 0.730	0.739 ; 0.739	0.739 ; 0.739

Tabela 19 – Resultados da Arquitetura VGG19 com Fusão Intermediária.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.842 ; 0.868	0.061 ; 0.003	0.732 ; 0.863	0.871 ; 0.869	0.869 ; 0.869
Precisão	0.864 ; 0.895	0.073 ; 0.002	0.732 ; 0.890	0.897 ; 0.896	0.896 ; 0.896
Revocação	0.842 ; 0.868	0.061 ; 0.003	0.732 ; 0.863	0.871 ; 0.869	0.869 ; 0.869
F-Score	0.840 ; 0.866	0.060 ; 0.003	0.732 ; 0.860	0.868 ; 0.867	0.867 ; 0.867
Valor Kappa	0.685 ; 0.736	0.122 ; 0.006	0.465 ; 0.726	0.742 ; 0.739	0.739 ; 0.739

Tabela 20 – Resultados da Arquitetura InceptionV3 com Fusão Intermediária.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.851 ; 0.820	0.028 ; 0.043	0.800 ; 0.765	0.869 ; 0.864	0.864 ; 0.813
Precisão	0.881 ; 0.864	0.015 ; 0.022	0.857 ; 0.840	0.896 ; 0.888	0.888 ; 0.857
Revocação	0.851 ; 0.820	0.028 ; 0.043	0.800 ; 0.765	0.869 ; 0.864	0.864 ; 0.813
F-Score	0.847 ; 0.814	0.031 ; 0.048	0.792 ; 0.751	0.867 ; 0.862	0.862 ; 0.807
Valor Kappa	0.702 ; 0.641	0.057 ; 0.087	0.601 ; 0.530	0.739 ; 0.729	0.729 ; 0.626

5.1.2.2 Fusão Tardia

As Tabelas 21, 22, 23, 24 e 25 descrevem as métricas coletadas durante os experimentos preliminares realizados com as arquiteturas multimodais utilizando fusão de dados tardia. Nas células das tabelas, os valores listados a esquerda do ponto-e-vírgula referem-se aos experimentos com imagens de tomografia computadorizada que não passaram por filtragem de ruído e realce de nitidez e os valores listados a direita do ponto-e-vírgula referem-se aos experimentos com imagens que passaram pela filtragem de ruído e realce de nitidez.

Tabela 21 – Resultados da Arquitetura RNCPU-HCC com Fusão Tardia.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.722 ; 0.673	0.124 ; 0.158	0.460 ; 0.390	0.869 ; 0.869	0.751 ; 0.724
Precisão	0.763 ; 0.708	0.140 ; 0.195	0.458 ; 0.373	0.896 ; 0.896	0.829 ; 0.803
Revocação	0.722 ; 0.673	0.124 ; 0.158	0.460 ; 0.390	0.869 ; 0.869	0.751 ; 0.724
F-Score	0.713 ; 0.653	0.125 ; 0.173	0.454 ; 0.362	0.867 ; 0.867	0.735 ; 0.708
Valor Kappa	0.444 ; 0.346	0.249 ; 0.317	-0.07 ; -0.21	0.739 ; 0.739	0.503 ; 0.449

Tabela 22 – Resultados da Arquitetura RNCPU-HCC Light com Fusão Tardia.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.666 ; 0.702	0.124 ; 0.122	0.365 ; 0.398	0.869 ; 0.869	0.672 ; 0.735
Precisão	0.718 ; 0.746	0.144 ; 0.150	0.329 ; 0.354	0.896 ; 0.896	0.734 ; 0.803
Revocação	0.666 ; 0.702	0.124 ; 0.122	0.365 ; 0.398	0.869 ; 0.869	0.672 ; 0.735
F-Score	0.644 ; 0.680	0.141 ; 0.143	0.330 ; 0.349	0.867 ; 0.867	0.668 ; 0.723
Valor Kappa	0.332 ; 0.404	0.248 ; 0.244	-0.26 ; -0.20	0.739 ; 0.739	0.345 ; 0.470

Tabela 23 – Resultados da Arquitetura Xception com Fusão Tardia.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.835 ; 0.809	0.049 ; 0.088	0.765 ; 0.673	0.869 ; 0.869	0.869 ; 0.869
Precisão	0.877 ; 0.866	0.026 ; 0.043	0.840 ; 0.802	0.896 ; 0.896	0.896 ; 0.896
Revocação	0.835 ; 0.809	0.049 ; 0.088	0.765 ; 0.673	0.869 ; 0.869	0.869 ; 0.869
F-Score	0.829 ; 0.797	0.054 ; 0.104	0.751 ; 0.634	0.867 ; 0.867	0.867 ; 0.867
Valor Kappa	0.670 ; 0.619	0.098 ; 0.177	0.530 ; 0.346	0.739 ; 0.739	0.739 ; 0.739

Tabela 24 – Resultados da Arquitetura VGG19 com Fusão Tardia.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.868 ; 0.831	0.002 ; 0.085	0.864 ; 0.678	0.869 ; 0.869	0.869 ; 0.869
Precisão	0.895 ; 0.853	0.003 ; 0.096	0.888 ; 0.681	0.896 ; 0.896	0.896 ; 0.896
Revocação	0.868 ; 0.831	0.002 ; 0.085	0.864 ; 0.678	0.869 ; 0.869	0.869 ; 0.869
F-Score	0.866 ; 0.829	0.002 ; 0.085	0.862 ; 0.676	0.867 ; 0.867	0.867 ; 0.867
Valor Kappa	0.737 ; 0.663	0.004 ; 0.171	0.729 ; 0.356	0.739 ; 0.739	0.739 ; 0.739

Tabela 25 – Resultados da Arquitetura InceptionV3 com Fusão Tardia.

Métrica	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**	**Mediana**
Acurácia	0.856 ; 0.827	0.030 ; 0.057	0.800 ; 0.738	0.869 ; 0.869	0.869 ; 0.859
Precisão	0.888 ; 0.872	0.017 ; 0.029	0.857 ; 0.828	0.896 ; 0.896	0.896 ; 0.880
Revocação	0.856 ; 0.827	0.030 ; 0.057	0.800 ; 0.738	0.869 ; 0.869	0.869 ; 0.859
F-Score	0.852 ; 0.820	0.033 ; 0.064	0.792 ; 0.719	0.867 ; 0.867	0.867 ; 0.856
Valor Kappa	0.712 ; 0.655	0.061 ; 0.114	0.601 ; 0.477	0.739 ; 0.739	0.739 ; 0.718

Os resultados obtidos na fase de experimentos preliminares indicam a superioridade de desempenho das arquiteturas multimodais e que a aplicação da filtragem de ruído e realce de nitidez das imagens pode incrementar o desempenho dependendo da rede convolucional e da estratégia de fusão de dados utilizada. Apesar da aparente confirmação da hipótese de pesquisa, ainda não é possível afirmar que o desempenho obtido refere-se à capacidade máxima dos modelos experimentados devido a quantidade de épocas de treinamento e a configuração utilizada no *Early Stopping*. Os experimentos de verificação devem fornecer as informações faltantes para responder a questão de pesquisa e confirmar a superioridade das abordagens multimodais propostas. O Capítulo 6 discute em detalhes os resultados obtidos nas duas fases de experimentos.

5.2 Experimentos de Verificação

Os experimentos realizados para testar os limites das arquiteturas e comparar as redes convolucionais profundas multimodais com maior desempenho com suas respectivas versões unimodais estão descritos na seção 4.3.4.2. Em resumo, nesta etapa as redes convolucionais profundas unimodais e multimodais passam por uma quantidade maior de épocas de treinamento para verificar se o desempenho obtido nos experimentos preliminares pode ser melhorado e para quantificar o ganho na aplicação de abordagens multimodais.

A métrica definida *a priori* nesta dissertação para comparação entre as arquiteturas experimentadas é a Precisão. Neste quesito a rede convolucional Xception com fusão de dados intermediária e utilizando imagens que passaram pela filtragem de ruído e realce da nitidez obteve o maior desempenho nos experimentos preliminares. Portanto, esta foi a primeira arquitetura escolhida para ser testada nesta etapa.

A diferença de precisão obtida com as redes convolucionais multimodais InceptionV3 e VGG19 não foi maior do que 0.05 (5%) em relação a rede Xception. Portanto, é possível considerar que a rede convolucional VGG19 também obteve resultados promissores. Sendo assim, optou-se por também experimentar nesta fase a rede convolucional VGG19 com fusão de dados intermediária utilizando imagens que passaram pela filtragem de ruído e realce da nitidez.

O desempenho menos expressivo dentre as arquiteturas multimodais estado da arte experimentadas foi obtido pela rede convolucional InceptionV3. Para confirmar os resultados obtidos nos experimentos preliminares, a rede InceptionV3 utilizando fusão de dados tardia com imagens que não passaram pela filtragem de ruído e realce de nitidez, (o melhor resultado desta rede convolucional) também foi incluída nesta fase dos experimentos.

Os resultados dos experimentos de verificação estão descritos nas Tabelas 26 e 27.

Tabela 26 – Resultados dos Experimentos de Verificação Unimodais.

Arquitetura	**Acurácia**	**Precisão**	**Revocação**	***F-Score***
Xception	0.606	0.608	0.606	0.604
InceptionV3	0.615	0.710	0.615	0.567
VGG19	0.459	0.410	0.459	0.373

As matrizes de confusão dos experimentos de verificação para as redes convolucionais Xception, InceptionV3 e VGG19 unimodais são, respectivamente:

$$\begin{bmatrix} 1119 & 963 \\ 674 & 1408 \end{bmatrix}, \begin{bmatrix} 1980 & 1498 \\ 102 & 584 \end{bmatrix} e \begin{bmatrix} 186 & 354 \\ 1896 & 1728 \end{bmatrix}$$

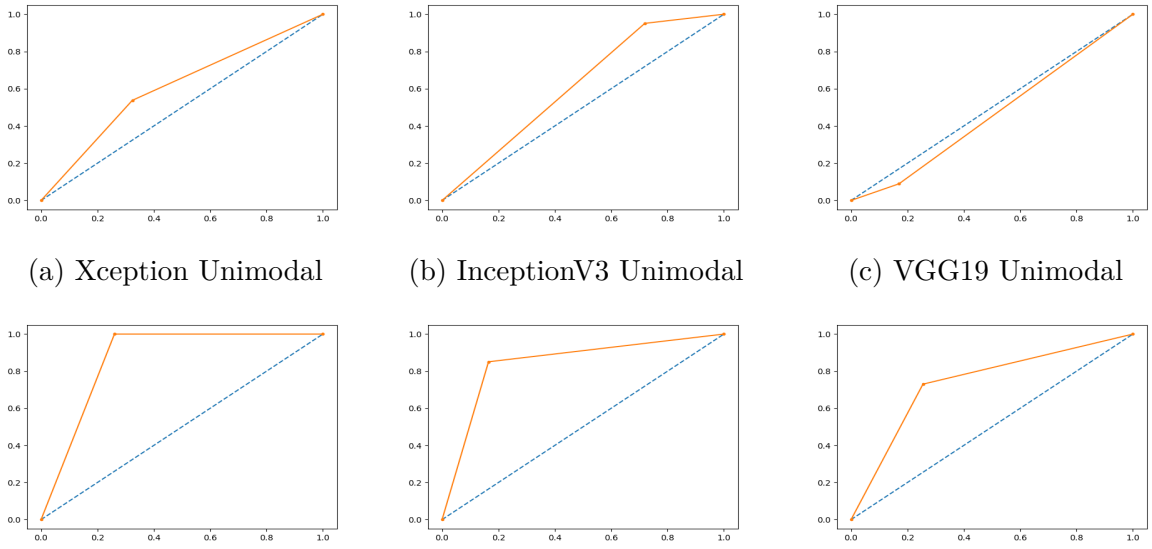
Tabela 27 – Resultados dos Experimentos de Verificação Multimodais.

Arquitetura	**Fusão**	**Acurácia**	**Precisão**	**Revocação**	***F-Score***
Xception	Intermediária	0.869	0.896	0.869	0.867
InceptionV3	Tardia	0.842	0.843	0.842	0.842
VGG19	Intermediária	0.737	0.737	0.737	0.737

As matrizes de confusão dos experimentos de verificação para as redes convolucionais Xception, InceptionV3 e VGG19 multimodais são, respectivamente:

$$\begin{bmatrix} 2082 & 0 \\ 542 & 1540 \end{bmatrix}, \begin{bmatrix} 1771 & 343 \\ 311 & 1739 \end{bmatrix} e \begin{bmatrix} 1520 & 531 \\ 562 & 1551 \end{bmatrix}$$

As Figuras 30a, 30b, 30c, 30d, 30e e 30f demonstram as curvas ROC obtidas nos experimentos de verificação.



(a) Xception Unimodal (b) InceptionV3 Unimodal (c) VGG19 Unimodal

(d) Xception (Fusão Interm.) (e) InceptionV3 (Fusão Tardia) (f) VGG19 (Fusão Interm.)

Figura 30 – Curvas ROC dos Experimentos de Verificação

5.3 Algoritmo de Auxílio-Diagnóstico de Hepatocarcinoma

A partir dos resultados obtidos com os experimentos de verificação é possível determinar os passos necessários para construção de um algoritmo de auxílio-diagnóstico para hepatocarcinoma com desempenho próximo ao estado da arte. A utilização de uma rede convolucional profunda multimodal estado da arte adaptada implica em um algoritmo com duas fases distintas. A primeira fase é responsável pela adaptação e treinamento da rede convolucional multimodal utilizada para o contexto deste projeto. O modelo criado e persistido nesta fase contém a arquitetura da rede convolucional Xception adaptada para fusão de dados intermediária, rede que apresentou os melhores resultados de precisão conforme visto anteriormente na Seção 5.2 com os pesos de cada neurônio e os valores dos hiperparâmetros. A sumarização dos passos necessários para este fim está descrita de

forma abstrata em pseudocódigo no Algoritmo 3.

Algoritmo 3: Treinamento para Auxílio-Diagnóstico de Hepatocarcinoma

Entradas : Imagem de Tomografia Computadorizada; Resultados de Exames Laboratoriais; Atributos Antropométricos e Sociodemográficos

Resultado : Rede Convolucional Multimodal Treinada para Auxílio-Diagnóstico de Hepatocarcinoma

1 Inicializar Parâmetros;
2 Normalizar Imagem DICOM;
3 Aplicar Equalização do Histograma Adaptativa;
4 Aplicar Filtro para Remoção de Ruído da Imagem;
5 Preencher Atributos Tabulares Faltantes;
6 Carregar Modelo Xception Pré-Treinado;
7 Adaptar Camada Densa para Fusão Intermediária;
8 Configurar Componente para *Data Augmentation*;
9 Configurar *Early Stopping* com 10 Épocas;
10 Configurar *Model Checkpoint*;
11 Configurar Otimizador SGD com *Learning Rate* = 0.0001 e *Momentum* = 0.9;
12 Treinar a Rede;
13 Salvar Modelo com Pesos que Atingiram Melhor Acurácia;

O modelo salvo na primeira fase pode ser carregado posteriormente através do *framework* Keras e reutilizado sem a necessidade de repetir os passos de treinamento descritos no Algoritmo 3. As possibilidades de reutilização do modelo salvo são (MENEGOTTO; BECKER; CAZELLA, 2019):

- De forma local através da execução manual do programa Python que realiza o auxílio-diagnóstico de hepatocarcinoma;
- De forma local e/ou remota através de scripts que interagem através da linha de comando do sistema operacional com o programa que realiza o auxílio-diagnóstico de hepatocarcinoma;
- De forma remota através de *endpoints Representational State Transfer* (REST) embutidos no programa que realiza o auxílio-diagnóstico de hepatocarcinoma;
- De forma remota através de uma página web que encapsula o programa que realiza o auxílio-diagnóstico de hepatocarcinoma;

O esqueleto de um programa que realiza o auxílio-diagnóstico de hepatocarcinoma com o resultado do treinamento realizado previamente está descrito de forma abstrata em

pseudocódigo no Algoritmo 4.

Algoritmo 4: Auxílio-Diagnóstico Computadorizado de Hepatocarcinoma

Entradas : Imagem de Tomografia Computadorizada; Resultados de Exames Laboratoriais; Atributos Antropométricos e Sociodemográficos

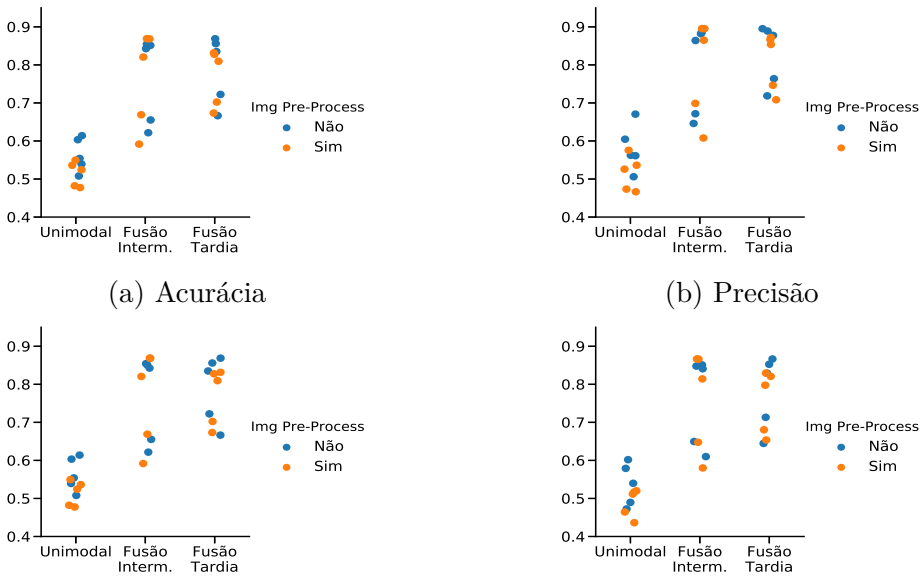
Resultado : Auxílio-Diagnóstico Computadorizado de Hepatocarcinoma

1 Inicializar Ambiente;
2 Normalizar Imagem DICOM;
3 Aplicar Equalização do Histograma Adaptativa;
4 Aplicar Filtro para Remoção de Ruído da Imagem;
5 Preencher Atributos Tabulares Faltantes;
6 Carregar Modelo Previamente Treinado;
7 Realizar Predição;

6 Análise dos Resultados

O algoritmo desenvolvido nesta dissertação utiliza imagens de tomografia computadorizada, resultados de exames laboratoriais, atributos antropométricos e sociodemográficos para auxílio-diagnóstico de hepatocarcinoma. O desenvolvimento do algoritmo foi baseado em decisões tomadas a partir de experimentos realizados para validar os passos de pré-processamento das imagens de tomografia computadorizada e para identificar a rede convolucional profunda e estratégia de fusão que obtém melhor precisão neste tipo de tarefa.

O pré-processamento realizado nas imagens serviu para redução de ruído e equalização do histograma visando melhora da nitidez. A olho nu a melhora da imagem é visível como já demonstrado na Figura 10. Em relação ao desempenho obtido nos experimentos, é possível ver nos resultados descritos no capítulo anterior que dependendo da estratégia de fusão de dados utilizada o pré-processamento das imagens beneficia o desempenho obtido. As Figuras 31a, 31b, 31c e 31d demonstram o panorama geral das médias da acurácia, precisão, revocação e *F-Score* obtidas nos experimentos preliminares comparando a aplicação do pré-processamento de imagens em relação a estratégia de fusão de dados. Percebe-se nestes gráficos que para a fusão intermediária obteve-se melhor desempenho utilizando as imagens com pré-processamento e na fusão tardia obteve-se melhor desempenho utilizando as imagens sem filtragem de ruído e realce de nitidez.



(a) Acurácia

(b) Precisão

(c) Revocação

(d) F-Score

Figura 31 – Estratégia de Fusão de Dados vs Pré-Processamento das Imagens

As redes convolucionais profundas RNCPU-HCC e RNCPU-HCC Light foram

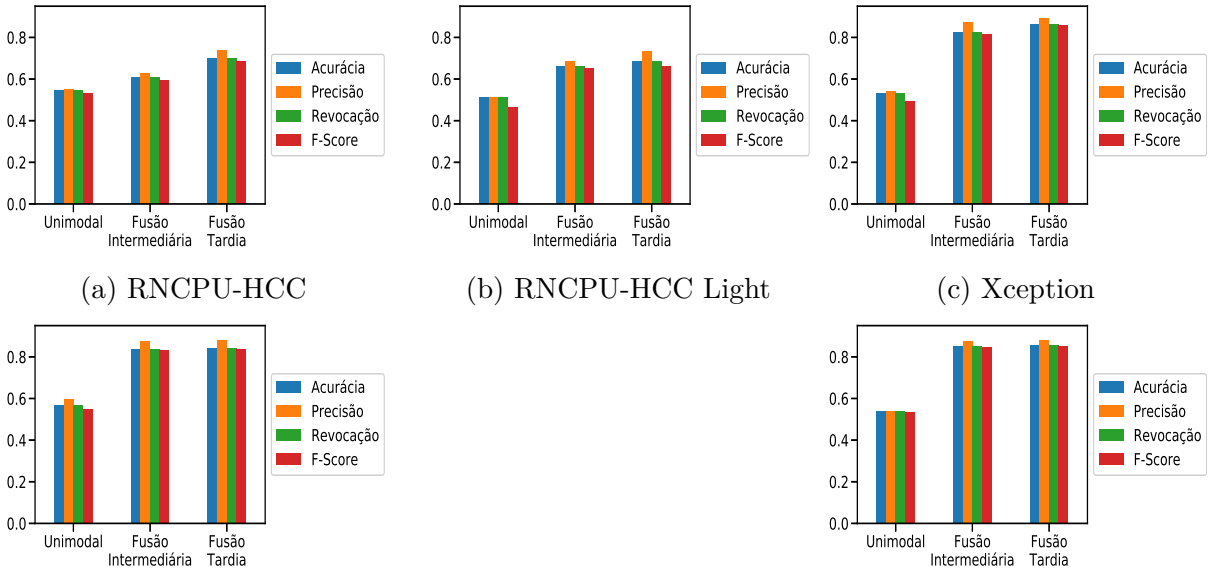
concebidas como alternativa para as redes estado da arte tendo por objetivo diminuir os requisitos computacionais necessários para treinamento e utilização da arquitetura proposta nesta dissertação. O processo de treinamento dessas redes necessitou uma quantidade maior de épocas visto que elas não possuíam nenhum conhecimento prévio. A duração aproximada de uma época de treinamento nos experimentos preliminares pode ser visto na Tabela 28. O objetivo de economizar recursos computacionais foi atingido, em detrimento a um maior poder de generalização devido a arquitetura reduzida.

Tabela 28 – Duração Aproximada de uma Época de Treinamento.

Arquitetura	**Abordagem**	**Duração**
RNCPU-HCC	Unimodal	45 segundos
RNCPU-HCC	Multimodal	49 segundos
RNCPU-HCC Light	Unimodal	45 segundos
RNCPU-HCC Light	Multimodal	49 segundos
Xception	Unimodal	181 segundos
Xception	Multimodal	222 segundos
InceptionV3	Unimodal	221 segundos
InceptionV3	Multimodal	243 segundos
VGG19	Unimodal	115 segundos
VGG19	Multimodal	140 segundos

A precisão média obtida com as redes convolucionais RNCPU-HCC e RNCPU-HCC Light na abordagem unimodal foi inferior as redes convolucionais estado da arte, porém a diferença de 0.108 (10.8%) não é proporcional ao número de camadas e/ou quantidade de neurônios que compõe estas redes. Observa-se entretanto uma diferença maior nas métricas obtidas nos experimentos com as abordagens multimodais, fato que pode ser explicado pela aplicação da transferência de conhecimento e pelo tamanho das redes convolucionais estado da arte que permitem o reconhecimento de um conjunto maior de padrões. Os pontos de máxima de cada métrica das redes RNCPU-HCC e RNCPU-HCC Light podem parecer promissores, entretanto ao confrontar esses pontos com as medidas de tendência central obtidas (média e mediana) nos experimentos preliminares, percebe-se que os pontos de máxima não representam a capacidade real dessas redes.

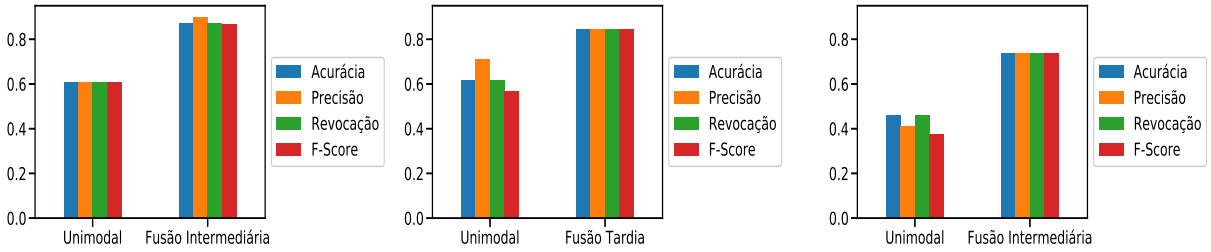
As métricas obtidas com as redes convolucionais profundas multimodais foram superiores em todas as fases de experimentos realizadas. As Figuras 32a, 32b, 32c, 32d e 32e contém um gráfico comparativo das métricas obtidas com as abordagens unimodais e multimodais nos experimentos preliminares. As Figuras 33a, 33b e 33c contém a mesma comparação para as arquiteturas testadas nos experimentos de verificação. A sumarização dos resultados nestes gráficos foi realizada através da média dos valores obtidos em cada métrica nos respectivos experimentos independente da utilização de imagens pré-processadas.



(a) RNCPU-HCC (b) RNCPU-HCC Light (c) Xception

(d) InceptionV3 (e) VGG19

Figura 32 – Sumarização Média dos Experimentos Preliminares



(a) Xception (b) InceptionV3 (c) VGG19

Figura 33 – Sumarização Média dos Experimentos de Verificação

Considerando a precisão obtida pelas arquiteturas testadas nos experimentos preliminares, percebe-se um ganho mínimo de 0.1573 (15.73%) com a rede convolucional RNCPU-HCC Light e um ganho máximo de 0.4220 (42.20%) com a rede convolucional VGG19 ao utilizar abordagens multimodais na aplicação destas técnicas para auxílio-diagnóstico de hepatocarcinoma. As redes convolucionais estado da arte obtiveram um ganho maior de desempenho ao utilizar abordagens multimodais nos experimentos preliminares e nos experimentos de verificação. A Tabela 29 ilustra a diferença de precisão e acurácia obtidas entre as abordagens unimodais e multimodais em cada uma das redes convolucionais testadas nos experimentos de verificação.

Tabela 29 – Ganho de Precisão e Acurácia nos Experimentos de Verificação.

Arquitetura	**Fusão de Dados**	**Ganho de Precisão**	**Ganho de Acurácia**
Xception	Intermediária	0.288	0.262
InceptionV3	Tardia	0.133	0.227
VGG19	Intermediária	0.327	0.277

As redes convolucionais RNCPU-HCC e RNCPU-HCC Light obtiveram um desempenho com as abordagens multimodais que se aproxima ao desempenho obtido pelas redes convolucionais estado da arte utilizando a abordagem unimodal. Este resultado corrobora com os achados de trabalhos relacionados na literatura que sugerem que a transferência de conhecimento e a adaptação de redes convolucionais estado da arte para um novo domínio de imagens fornece melhores resultados que a criação e o treinamento de uma rede convolucional utilizando bases de dados com poucos exemplos (ZHANG et al., 2017) (ESTEVA et al., 2017) (RATTANI; DERAKHSHANI, 2017).

É possível observar também nos gráficos diferenças entre o desempenho obtido nos experimentos preliminares em relação ao desempenho das redes convolucionais nos experimentos de verificação. As diferenças ficam mais evidentes na rede convolucional VGG19 e acredita-se ser um efeito colateral da maior quantidade de épocas de treinamento aplicada nos experimentos de verificação, fato observado também por outros trabalhos encontrados na literatura (HERMESSI; MOURALI; ZAGROUBA, 2019) (SWATI et al., 2019). A relação inversa entre a quantidade de épocas de treinamento e as métricas de classificação não foi observada nas redes convolucionais Xception e InceptionV3.

O desempenho para auxílio-diagnóstico de hepatocarcinoma obtido nos experimentos de verificação pela rede convolucional Xception com o conjunto de testes previamente descrito utilizando fusão de dados intermediária com imagens que passaram por filtragem de ruído e realce de nitidez foi: acurácia = 0.869 (86.9%), precisão = 0.896 (89.6%), revocação = 0.869 (86.9%) e *F-Score* = 0.867 (86.7%). Como somente foram realizados testes com um único conjunto de pacientes neste estudo, é precoce afirmar que o desempenho do algoritmo desenvolvido pode realmente atingir estas métricas quando aplicado em instituições assistenciais. O que é possível afirmar a partir dos resultados obtidos é que a utilização de exames de imagem combinados com outras modalidades de dados como resultados de exames laboratoriais, atributos antropométricos e sociodemográficos incrementa a precisão de sistemas de auxílio-diagnóstico de hepatocarcinoma desenvolvidos com aprendizado de máquina profundo multimodal.

A busca na literatura por trabalhos que comparem especificamente as métricas diagnósticas de especialistas (radiologistas, hepatologistas, etc.) utilizando imagens de tomografia computadorizada, resultados de exames laboratoriais, dados antropométricos e atributos sociodemográficos não retornou resultados. O que se encontrou na literatura foram comparativos do desempenho de insumos e/ou metodologias utilizadas para o diagnóstico de hepatocarcinoma.

Estudos de revisão realizados comparando o desempenho relatado na literatura para diagnóstico de hepatocarcinoma utilizando ultrassonografia não-contrastada e contrastada apontam para uma sensibilidade (revocação) aproximada de 59.3% e 84.4%, respectivamente. As imagens de tomografia computadorizada com contraste possuem sensibilidade

aproximada na faixa de [72.0% - 73.6%]. Diagnósticos realizados com ressonância magnética com contraste possuem sensibilidade aproximada na faixa de [72.0% - 73.6%] e [77.5% - 85.6%] respectivamente, dependendo da substância contrastante utilizada (gadolínio, ácido gadoxético) e do tamanho das lesões analisadas (HANNA et al., 2016) (LEE et al., 2015) (CARVALHO; PEREIRA, 2015).

A utilização de atributos microRNA para diagnóstico de hepatocarcinoma obtém sensibilidade aproximada de 81.9% porém a combinação destes atributos com o nível de Alfafetoproteína obtém sensibilidade diagnóstica de 87.4% (HE et al., 2016b). A utilização de biomarcadores AFP-L3 e *Protein Induced by Vitamin K deficiency or antagonist-II* (PIVKA-II) de forma isolada para diagnóstico de hepatocarcinoma obtém sensibilidade de 76.3% e 51.02% respectivamente. Porém, a combinação destes biomarcadores com microRNA ou a combinação de Alfafetoproteína e Golgi-73 obtém uma sensibilidade diagnóstica superior a 90% (WANG; WANG, 2018) (LUO et al., 2019).

É possível encontrar na literatura estudos que indicam que até os sistemas de classificação utilizados podem resultar em diferenças de desempenho diagnóstico entre especialistas. Para diagnósticos não-invasivos, o protocolo definido pela AASLD obtém uma sensibilidade entre [71.4% - 72.9%] dependendo do exame de imagem utilizado enquanto que ao utilizar a classificação LI-RADS obtém-se sensibilidade diagnóstica entre [67.5% - 72.5%] (RONOT et al., 2018).

Observa-se ao analisar a literatura que insumos diagnósticos utilizados de forma isolada obtém um desempenho inferior a utilização combinada destes insumos para diagnóstico de hepatocarcinoma por especialistas. O diferencial da abordagem descrita nesta dissertação é a replicação deste padrão aplicado na implementação de sistemas de auxílio-diagnóstico de hepatocarcinoma, ao utilizar imagens de tomografia computadorizada em conjunto com exames laboratoriais, atributos antropométricos e sociodemográficos. Portanto, apesar de múltiplas técnicas de inteligência artificial terem sido citadas no Capítulo 3, os trabalhos selecionados para base comparativa de desempenho utilizam exclusivamente técnicas de aprendizado de máquina profundo multimodal no auxílio-diagnóstico de carcinoma hepatocelular. A Tabela 30 lista o resultado obtido com a arquitetura desenvolvida nesta dissertação e o desempenho de abordagens alternativas encontradas na literatura.

Tabela 30 – Comparativo com Trabalhos Relacionados

Referência	**Técnica**	**Entradas**	**Resultado**
Arquitetura Proposta	Rede Convolucional Multimodal	Imagens de Tomografia Computadorizada, exames laboratoriais, atributos antropométricos e sociodemográficos	Acurácia = 0.86; Precisão = 0.89; Revocação = 0.86

Continuação da Tabela 30

Referência	**Técnica**	**Entradas**	**Resultado**
(MIOTTO; LI; DUDLEY, 2016)	*Stacked Denoising AutoEncoders* e Florestas Aleatórias	Laudos radiológicos, Atributos Clínicos e Sociodemográficos	AUROC = 0.92; *F-Score* = 0.22
(WANG et al., 2017)	Rede Convolucional Multimodal	Múltiplos Planos de Imagens de Ressonância Magnética	Acurácia = 0.99 ± 0.02; Revocação = 0.98 ± 0.03; Especificidade = 1.00 ± 0.00
(DOU; ZHOU, 2018)	Rede Convolucional Multimodal	*Patches* 2D e 3D de Imagens de Ressonância Magnética	Acurácia = 0.92 ± 0.05; Revocação = 0.90 ± 0.08; Especificidade = 0.93 ± 0.08
(DOU et al., 2018)	Rede Convolucional Multimodal	Atributos de Imagens de Ressonância Magnética	Acurácia = 0.93 ± 0.04; Revocação = 0.94 ± 0.08; Especificidade = 0.91 ± 0.11
(DOU; ZHANG; ZHOU, 2018)	Rede Convolucional Multimodal 3D	Fases de um estudo abdominal de ressonância magnética contrastada (pré-contraste, arterial e portal)	Acurácia = 0.89 ± 0.06; Revocação = 0.90 ± 0.13; Especificidade = 0.89 ± 0.10
(ZHOU et al., 2019)	Rede Convolucional Multimodal	Atributos de Imagens de Ressonância Magnéticas Difusas (DWI-MRI)	Acurácia = 0.80 ± 0.00; Revocação = 0.805 ± 0.00; Especificidade = 0.71 ± 0.02
(LIN et al., 2019)	Rede Convolucional Multimodal	Atributos e Imagens Histopatológicas	Acurácia = 0.94

O desempenho dos estudos que utilizaram arquiteturas profundas e imagens de ressonância magnética para realização do auxílio-diagnóstico é superior ao desempenho obtido com imagens de tomografia computadorizadas na arquitetura desenvolvida nesta dissertação, repetindo os resultados relatados em estudos previamente citados nesta seção sobre a acurácia diagnóstica de especialistas que utilizaram este tipo de exame de imagem (HANNA et al., 2016) (LEE et al., 2015) (CARVALHO; PEREIRA, 2015). Percebe-se também uma superioridade no desempenho de artigos que utilizaram abordagens 3D no processamentos dos estudos de imagem, fato que pode estar ligado a características que não são capturadas pelos algoritmos de aprendizado de máquina profundo ao processar as imagens de forma isolada (YAN et al., 2016).

Os trabalhos relacionados que utilizaram imagens de ressonância magnética são aqueles que possuem maior diferença de desempenho em relação a arquitetura proposta nesta dissertação. A diferença máxima de acurácia e revocação é respectivamente 0.123 (12.3%) e 0.1199 (11.99%) (WANG et al., 2017). A diferença média de acurácia e revocação obtida com os artigos que utilizaram ressonância magnética, descartando os trabalhos com melhor e pior desempenho é de 0.0466 (4.66%) e 0.0481 (4.81%), respectivamente (ZHOU et al., 2019) (DOU; ZHANG; ZHOU, 2018) (DOU; ZHOU, 2018) (DOU et al., 2018) (WANG et al., 2017). A acurácia da arquitetura descrita em (LIN et al., 2019) e que utiliza uma combinação de imagens e atributos histopatológicos difere do desempenho obtido nesta dissertação em 0.071 (7.1%). O resultado das métricas de desempenho descrito em (MIOTTO; LI; DUDLEY, 2016) é conflitante, considerando que ambas métricas descritas são derivadas da quantidade de verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos. Para fins de completude da análise, a diferença de desempenho desse artigo com o arquitetura proposta nesta dissertação é de 0.051 (5.1%) na AUROC e -0.65 (-65%) no *F-Score*.

Apesar dos resultados promissores obtidos pelos trabalhos (WANG et al., 2017), (DOU; ZHANG; ZHOU, 2018), (DOU; ZHOU, 2018) e (DOU et al., 2018), deve-se observar que todos utilizam a mesma base de dados, composta de 46 pacientes com hepatocarcinoma confirmado via exame histopatológico e cujas imagens de ressonância magnética contrastada foram adquiridos entre setembro de 2011 e outubro de 2015 utilizando equipamento da *GE Healthcare*.

Os resultados descritos na Tabela 30 demonstram que existe espaço para melhorias na arquitetura proposta nesta dissertação. A adaptação do algoritmo proposto neste trabalho para utilizar outros tipos de exames de imagem como ressonância magnética e imagens histopatológicas envolveria basicamente o pré-processamento das imagens e atributos tabulares seguido de um novo treinamento da rede convolucional Xception para adaptação ao novo domínio de imagem. Seguindo a lógica do desempenho encontrada nos artigos relacionados, a utilização de imagens de alta resolução como ressonância magnética

e imagens histopatológicas poderia aumentar o desempenho da arquitetura proposta.

Apesar da simplicidade na descrição da melhoria, a realização desta mudança envolve encontrar uma base de dados rotulada com relacionamento entre as imagens e os resultados de exames laboratoriais, atributos antropométricos e sociodemográficos que seja passível de uso respeitando os padrões éticos que devem ser seguidos com dados sensíveis em projetos de pesquisa. Além disso, o processo de aquisição desse tipo de imagem possui desvantagens em relação a tomografia computadorizada: 1) a utilização de ressonância magnética eleva o custo do diagnóstico; 2) a utilização de imagens histopatológicas requer a coleta invasiva dos tecidos para análise e costuma ser recomendada somente após exames prévios que indiquem uma lesão maligna (RASTOGI, 2018). Portanto, apesar de parecer uma mudança promissora para melhoria do desempenho da arquitetura proposta, existem implicações que dificultam a aplicabilidade desta abordagem.

O processamento dos estudos de tomografia computadorizada de forma 3D também poderia melhorar o desempenho da arquitetura proposta nesta dissertação considerando que bastaria modificar a arquitetura da rede convolucional multimodal para aceitar entradas volumétricas. Esta mudança pode ser realizada nas redes convolucionais RNCPU-HCC e RNCPU-HCC Light porém a utilização de imagens volumétricas inviabiliza a utilização da transferência de conhecimento e adaptação de redes estado da arte. Outra possibilidade seria acrescentar camadas recorrentes para trabalhar a questão da temporalidade no processamento das imagens dos estudos de tomografia computadorizada, na tentativa de capturar os padrões e características que seriam naturalmente reconhecidas por redes convolucionais 3D e conseguir utilizar transferência de conhecimento com as redes estado da arte.

Outra possível oportunidade de melhora está relacionada a estratégia de divisão de dados utilizada. A validação cruzada *k-fold* aplicada durante o processo de treinamento e validação do modelo pode aumentar o desempenho obtido (YADAV; SHUKLA, 2016) porém optou-se pela estratégia *Holdout* devido ao tamanho da base de imagens e ao tempo necessário para repetição dos ciclos da estratégia de validação *k-fold*. O trabalho para realização desta melhoria envolve a reunificação dos subconjuntos de treinamento, validação e testes em uma única base que seria então dividida em 10 partes iguais, considerando que seja utilizada a estratégia de validação cruzada *10-fold*. Após, é necessário modificar os experimentos para iterar manualmente entre os subconjuntos criados e reexecutar os experimentos previamente descritos.

A escolha dos atributos tabulares foi baseada nos protocolos de diagnóstico de hepatocarcinoma recomendados por instituições de referência e na existência desses dados nos arquivos obtidos do TCGA. Os atributos recomendados para utilização durante o diagnóstico de hepatocarcinoma são características consolidadas e com evidências científicas de assertividade na detecção de hepatocarcinoma. Atributos que ainda estão sendo pesqui-

sados pela comunidade científica e que apresentam bom desempenho no diagnóstico de hepatocarcinoma podem ser acoplados a arquitetura proposta neste dissertação bastando para isso congelar as camadas convolucionais e realizar um novo treinamento do ponto de fusão de dados e das camadas densas. Os biomarcadores Golgi 73, AFP-L3 e PIVKA-II são exemplos de atributos que podem ser acrescentados caso estejam disponíveis no prontuário eletrônico de pacientes para melhoria de desempenho da arquitetura proposta (LOU et al., 2017) (PARK et al., 2017).

A implementação da arquitetura proposta somente pode ser utilizada no dia-a-dia por profissionais assistenciais para auxílio-diagnóstico de hepatocarcinoma após a realização de novas sessões de treinamento e teste, reforçando o conhecimento já adquirido e utilizando conjuntos de dados reais de pacientes para aferição da continuidade do desempenho obtido com as bases de dados descritas nesta dissertação. A utilização de dados tabulares sintéticos em atributos faltantes pode introduzir ruído nas correlações e influenciar negativamente na continuidade do desempenho em outras bases de dados e esta é a primeira limitação da arquitetura descrita nesta dissertação.

A segunda limitação da arquitetura proposta é relacionada a quantidade de atributos tabulares utilizados. A popularização de sistemas de prontuário eletrônico por instituições hospitalares e assistenciais aumenta a probabilidade da existência de todos estes atributos, somados a existência de um estudo de tomografia computadorizada para que o algoritmo proposto possa classificar o estado de saúde do paciente em relação ao hepatocarcinoma. Apesar disso, no mundo real nem sempre temos todos os atributos preenchidos corretamente e a inexistência de atributos pode influenciar negativamente na aplicabilidade da arquitetura proposta no dia-a-dia.

A terceira limitação do trabalho é fruto do viés intrínseco que pode ser atribuído aos dados utilizados e as decisões tomadas durante a construção de um modelo classificatório com técnicas de aprendizado de máquina profundo.

Os resultados obtidos com a utilização de abordagens multimodais em sistemas de auxílio-diagnóstico de hepatocarcinoma são promissores conforme já descrito no Capítulo 3. A comparação do desempenho obtido pela arquitetura desenvolvida nesta dissertação com trabalhos relacionados fica comprometida, pois o único trabalho relacionado que utilizou imagens de tomografia computadorizada não foi implementado com aprendizado de máquina profundo.

O algoritmo descrito nesta dissertação, apesar de possuir desempenho inferior a uma parcela de trabalhos relacionados, possui pontos que podem ser explorados visando melhora no desempenho conforme descrito nesta seção. A realização destas melhorias pode resultar na diminuição do custo do processo diagnóstico de hepatocarcinoma ao utilizar um exame de menor custo e obter o mesmo desempenho na detecção da doença.

O desempenho obtido atualmente pela arquitetura descrita é superior aos resultados descritos na literatura para diagnósticos de hepatocarcinoma realizados por especialistas utilizando imagens de tomografia computadorizada e este fato embasa a continuidade deste projeto visando superar o estado da arte do auxílio-diagnóstico de hepatocarcinoma utilizando aprendizado de máquina profundo multimodal.

7 Conclusão

O desenvolvimento de um algoritmo de auxílio-diagnóstico de hepatocarcinoma objetiva fornecer uma ferramenta de apoio a decisão médica que agiliza o processo de diagnóstico e aumenta a probabilidade de sobrevida do paciente. O diferencial deste algoritmo em relação aos trabalhos encontrados na literatura é a utilização de imagens de tomografia computadorizada combinadas com resultados de exames laboratoriais, atributos antropométricos e atributos sociodemográficos.

O processo de desenvolvimento foi composto de uma fase de busca, pré-processamento e montagem das bases de dados seguido do projeto das arquiteturas de redes neurais convolucionais experimentadas. Experimentos preliminares foram realizados para atestar a robustez das arquiteturas utilizadas e escolher aquela com melhor desempenho no auxílio-diagnóstico de hepatocarcinoma. Após o término dos experimentos preliminares, as três arquiteturas com maior desempenho foram testadas utilizando um ciclo maior de treinamento para verificar se o desempenho previamente atingido nos experimentos preliminares representava efetivamente o desempenho máximo dessas redes e para medir o impacto da utilização de múltiplas modalidades de dados em sistemas de auxílio-diagnóstico de hepatocarcinoma desenvolvidos com aprendizado de máquina profundo.

A arquitetura com maior precisão no auxílio-diagnóstico de hepatocarcinoma utilizou imagens de tomografia computadorizada pré-processadas e a rede convolucional Xception adaptada para este fim, combinando exames de imagem e dados tabulares do paciente através de fusão intermediária. O desempenho obtido pela arquitetura proposta aproxima-se do estado da arte do auxílio-diagnóstico de hepatocarcinoma utilizando aprendizado de máquina profundo multimodal.

Os experimentos realizados sugerem que a utilização de múltiplas modalidades de dados como insumo de entrada de algoritmos de aprendizado de máquina profundo para auxílio-diagnóstico de hepatocarcinoma aumenta o desempenho desse tipo de ferramenta. Os resultados obtidos sugerem também que a utilização de imagens de tomografia computadorizadas pré-processadas fornecem desempenho superior caso combinadas com outras modalidades de dados, através de fusão intermediária de dados, e que as imagens sem pré-processamento fornecem maior desempenho quando combinadas com outras modalidades de dados através de fusão tardia.

A resposta para questão de pesquisa que norteou o trabalho conduzido é positiva, ou seja, a utilização de resultados de exames laboratoriais como Tempo de Protrombina, Albumina, Bilirrubinas Totais e Creatinina em conjunto com biomarcadores tumorais como Alfafetoproteína, dados antropométricos e dados sociodemográficos associados a exames

de imagem *aumenta a precisão* de um algoritmo de auxílio-diagnóstico computadorizado de hepatocarcinoma. Durante os experimentos preliminares, o ganho mínimo de precisão obtido com abordagens multimodais foi de 0.1573 (15.73%) com a rede convolucional RNCPU-HCC Light e o ganho máximo de precisão foi obtido pela rede convolucional VGG19 com 0.4220 (42.20%). O ganho de precisão e acurácia com a arquitetura proposta nesta dissertação durante os experimentos de verificação foi de 0.288 (28.80)% e 0.262 (26.20%) respectivamente.

O desempenho obtido pela arquitetura proposta na base de dados utilizada para os experimentos é superior ao desempenho médio relatado na literatura de especialistas que utilizam exclusivamente imagens de tomografia computadorizada no diagnóstico. Apesar disso, a aplicação de técnicas de aprendizado de máquina profundo multimodal demanda ações e decisões que carregam o viés do pesquisador. Os experimentos descritos nesta dissertação utilizaram uma única base de dados e antes da aplicação deste algoritmo em ambientes assistenciais reais é necessário realizar novos ciclos de treinamento e testes com novas bases de dados. Outra abordagem para implantação desta arquitetura seria deixá-la executando em fase probatória durante um grande período de tempo (ex: um ano) em um hospital de grande porte utilizando erros e acertos das predições para reforçar os padrões previamente aprendidos e aumentar a robustez do modelo.

Trabalhos futuros com objetivo de melhorar o desempenho da arquitetura proposta podem aplicar diretamente as abordagens multimodais e experimentar alternativas para a rede neural convolucional Xception, para os dados tabulares (ex: novos biomarcadores tumorais), para o *framework* Keras e para os algoritmos de pré-processamento de imagens utilizados. Outra possível linha para investigação está relacionada ao impacto da aplicação de métodos estado-da-arte para preenchimento de dados tabulares faltantes no desempenho obtido no auxílio-diagnóstico de hepatocarcinoma. Pode-se também aplicar a arquitetura proposta para auxílio-diagnóstico de outras doenças que podem ser diagnosticadas combinando exames de imagens e dados tabulares como resultados de exames laboratoriais com atributos clínicos, antropométricos e sociodemográficos, bastando para isso obter as bases de dados do domínio trabalhado e realizar ciclos de treinamento e validação para adaptar a rede convolucional ao novo atributo classe.

Referências

ABADI, M. et al. Tensorflow: A system for large-scale machine learning. In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. [S.l.: s.n.], 2016. p. 265–283. Citado na página 23.

ABAJIAN, A. et al. Predicting treatment response to intra-arterial therapies for hepatocellular carcinoma with the use of supervised machine learning—an artificial intelligence concept. *Journal of Vascular and Interventional Radiology*, Elsevier, v. 29, n. 6, p. 850–857, 2018. Citado 2 vezes nas páginas 15 e 34.

AHARON, M.; ELAD, M.; BRUCKSTEIN, A. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, IEEE, v. 54, n. 11, p. 4311–4322, 2006. Citado na página 35.

AL-AMEEN, Z. et al. An innovative technique for contrast enhancement of computed tomography images using normalized gamma-corrected contrast-limited adaptive histogram equalization. *EURASIP Journal on Advances in Signal Processing*, SpringerOpen, v. 2015, n. 1, p. 32, 2015. Citado na página 42.

Alejandro Forner and María Reig and Jordi Bruix. Hepatocellular carcinoma. *The Lancet*, v. 391, n. 10127, p. 1301 – 1314, 2018. ISSN 0140-6736. Citado na página 31.

ALI, L. et al. Machine learning based computer-aided diagnosis of liver tumours. In: *2017 IEEE 16th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC)*. IEEE, 2017. p. 139–145. ISBN 978-1-5386-0771-8. Disponível em: <http://ieeexplore.ieee.org/document/8109742/>. Citado 2 vezes nas páginas 15 e 34.

ALZUBAIDI, A. K. et al. Computer aided diagnosis in digital pathology application: Review and perspective approach in lung cancer classification. In: IEEE. *2017 annual conference on new trends in information & Communications technology applications (NTICT)*. [S.l.], 2017. p. 219–224. Citado na página 24.

ANTONIO, V. A. A. et al. Classification of lung adenocarcinoma transcriptome subtypes from pathological images using deep convolutional networks. *International Journal of Computer Assisted Radiology and Surgery*, Springer Verlag, v. 13, n. 12, p. 1905–1913, 12 2018. ISSN 18616429. Citado na página 26.

ARAGON, G.; YOUNOSSI, Z. M. When and how to evaluate mildly elevated liver enzymes in apparently healthy patients. *Cleve Clin J Med*, v. 77, n. 3, p. 195–204, 2010. Citado na página 48.

ATTWA, M. H.; EL-ETREBY, S. A. *Guide for diagnosis and treatment of hepatocellular carcinoma*. 2015. Citado na página 14.

BALDUINI, C. L.; NORIS, P. Platelet count and aging. *Haematologica*, The Ferrata Storti Fundation, v. 99, n. 6, p. 953–955, 2014. Citado na página 49.

BALL, D.; ROSE, E.; ALPERT, E. Alpha-fetoprotein levels in normal adults. *The American journal of the medical sciences*, Elsevier, v. 303, n. 3, p. 157–159, 1992. Citado na página 49.

BEN-COHEN, A. et al. Sparsity-based liver metastases detection using learned dictionaries. In: IEEE. *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI)*. [S.l.], 2016. p. 1195–1198. Citado na página 35.

BENGIO, Y. et al. Greedy layer-wise training of deep networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2007. p. 153–160. Citado na página 24.

BIBAULT, J. E. et al. Deep Learning and Radiomics predict complete response after neo-adjuvant chemoradiation for locally advanced rectal cancer. *Scientific Reports*, Nature Publishing Group, v. 8, n. 1, 12 2018. ISSN 20452322. Citado na página 26.

BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 2nd. ed. Berlin, Heidelberg: Springer-Verlag, 2006. 1–33 p. ISBN 0387310738. Citado 2 vezes nas páginas 18 e 19.

BOMFORD, C. K. et al. *Walter and Miller's textbook of radiotherapy: radiation physics, therapy, and oncology*. 7th. ed. [S.l.]: Elsevier, 2012. 276–277 p. Citado na página 27.

BOMFORD, C. K. et al. *Walter and Miller's textbook of radiotherapy: radiation physics, therapy, and oncology*. 7th. ed. [S.l.]: Elsevier, 2012. 85–95 p. Citado na página 40.

BOSCH, F. X. et al. Epidemiology of hepatocellular carcinoma. *Clinics in liver disease*, Elsevier, v. 9, n. 2, p. 191–211, 2005. Citado 2 vezes nas páginas 45 e 48.

BUDA, M.; MAKI, A.; MAZUROWSKI, M. A. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, Elsevier, v. 106, p. 249–259, 2018. Citado na página 52.

BURTIS, C. A.; BRUNS, D. E. *Tietz fundamentals of clinical chemistry*. 6th. ed. [S.l.]: Elsevier Health Sciences, 2014. 366 p. Citado 2 vezes nas páginas 50 e 51.

BUZUG, T. M. Computed tomography. In: *Springer Handbook of Medical Technology*. [S.l.]: Springer, 2011. p. 329–335. Citado na página 41.

Cancer Research UK. *Worldwide cancer incidence statistics | Cancer Research UK*. 2018. Acesso Em: 19 set. 2019. Disponível em: <http://www.cancerresearchuk.org/health-professional/cancer-statistics/worldwide-cancer/incidence#heading-Five>. Citado na página 14.

CARVALHO, P. B.; PEREIRA, E. Imagiological diagnosis of gastrointestinal diseases–diagnostic criteria of hepatocellular carcinoma. *GE Portuguese journal of gastroenterology*, Elsevier, v. 22, n. 4, p. 153–160, 2015. Citado 2 vezes nas páginas 87 e 89.

CHAMBOLLE, A. An algorithm for total variation minimization and applications. *Journal of Mathematical imaging and vision*, Springer, v. 20, n. 1-2, p. 89–97, 2004. Citado na página 42.

CHEN, T. et al. Disc: Deep image saliency computing via progressive representation learning. *IEEE transactions on neural networks and learning systems*, IEEE, v. 27, n. 6, p. 1135–1149, 2016. Citado na página 23.

CHEN, Y. et al. Three-way decision support for diagnosis on focal liver lesions. *Knowledge-Based Systems*, Elsevier, v. 127, p. 85–99, 2017. ISSN 09507051. Citado 2 vezes nas páginas 15 e 34.

CHERNYAK, V. et al. Liver imaging reporting and data system (li-rads) version 2018: imaging of hepatocellular carcinoma in at-risk patients. *Radiology*, Radiological Society of North America, v. 289, n. 3, p. 816–830, 2018. Citado na página 28.

CHOLLET, F. *Keras*. 2015. Acesso Em: 19 set. 2019. Disponível em: <https://keras.io>. Citado na página 23.

CHOLLET, F. *Deep Learning with Python*. 1st. ed. [S.l.]: Manning Publications Co., 2017. 93–96 p. ISBN 1617294438, 9781617294433. Citado na página 18.

CHOLLET, F. *Deep Learning with Python*. 1st. ed. [S.l.]: Manning Publications Co., 2017. 1–20 p. ISBN 1617294438, 9781617294433. Citado na página 20.

CHOLLET, F. *Deep Learning with Python*. 1st. ed. [S.l.]: Manning Publications Co., 2017. 97–100 p. ISBN 1617294438, 9781617294433. Citado na página 51.

CHOLLET, F. *Deep Learning with Python*. 1st. ed. [S.l.]: Manning Publications Co., 2017. 135-142 p. ISBN 1617294438, 9781617294433. Citado 2 vezes nas páginas 55 e 71.

CHOLLET, F. *Deep Learning with Python*. 1st. ed. [S.l.]: Manning Publications Co., 2017. 107-108 p. ISBN 1617294438, 9781617294433. Citado na página 56.

CHOLLET, F. Xception: Deep learning with depthwise separable convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 1251–1258. Citado 2 vezes nas páginas 56 e 60.

CLARK, K. et al. The cancer imaging archive (tcia): maintaining and operating a public information repository. *Journal of digital imaging*, Springer, v. 26, n. 6, p. 1045–1057, 2013. Citado na página 38.

DOU, T. et al. Local and non-local deep feature fusion for malignancy characterization of hepatocellular carcinoma. In: SPRINGER. *International Conference on Medical Image Computing and Computer-Assisted Intervention*. [S.l.], 2018. p. 472–479. Citado 3 vezes nas páginas 36, 88 e 89.

DOU, T.; ZHANG, L.; ZHOU, W. 3d deep feature fusion in contrast-enhanced mr for malignancy characterization of hepatocellular carcinoma. In: IEEE. *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*. [S.l.], 2018. p. 29–33. Citado 3 vezes nas páginas 36, 88 e 89.

DOU, T.; ZHOU, W. 2D and 3D Convolutional Neural Network Fusion for Predicting the Histological Grade of Hepatocellular Carcinoma. In: *24th International Conference on Pattern Recognition (ICPR)*. IEEE, 2018. p. 3832–3837. ISBN 978-1-5386-3788-3. Disponível em: <https://ieeexplore.ieee.org/document/8545806/>. Citado 3 vezes nas páginas 36, 88 e 89.

DUA, D.; GRAFF, C. *UCI Machine Learning Repository*. 2017. Acesso Em: 19 set. 2019. Disponível em: <http://archive.ics.uci.edu/ml>. Citado na página 48.

DUFOUR, D. R. et al. Diagnosis and monitoring of hepatic injury. i. performance characteristics of laboratory tests. *Clinical chemistry*, Clinical Chemistry, v. 46, n. 12, p. 2027–2049, 2000. Citado na página 48.

EATON-ROSEN, Z. et al. Towards safe deep learning: accurately quantifying biomarker uncertainty in neural network predictions. In: SPRINGER. *International Conference on Medical Image Computing and Computer-Assisted Intervention*. [S.l.], 2018. p. 691–699. Citado na página 52.

EL-SERAG, H. B. et al. A new laboratory-based algorithm to predict development of hepatocellular carcinoma in patients with hepatitis c and cirrhosis. *Gastroenterology*, Elsevier, v. 146, n. 5, p. 1249–1255, 2014. Citado 2 vezes nas páginas 46 e 48.

ELLISON, E. C. et al. The impact of the aging population and incidence of cancer on future projections of general surgical workforce needs. *Surgery (United States)*, Mosby Inc., v. 163, n. 3, p. 553–559, 3 2018. ISSN 15327361. Citado na página 14.

ERICKSON, B. J. et al. *TCGA-LIHC - The Cancer Genome Atlas Liver Hepatocellular Carcinoma [TCGA-LIHC] collection*. 2017. Acesso Em: 19 set. 2019. Disponível em: <https://wiki.cancerimagingarchive.net/display/Public/TCGA-LIHC#49e04d416a274e2c9a1218c4350512e9>. Citado na página 39.

ESTEVA, A. et al. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, Nature Publishing Group, v. 542, n. 7639, p. 115, 2017. Citado 4 vezes nas páginas 14, 24, 59 e 86.

FRID-ADAR, M. et al. Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, Elsevier, v. 321, p. 321–331, 2018. Citado na página 55.

FRYAR, C. D. et al. Anthropometric reference data for children and adults; United States, 2011-2014. *Vital and health statistics*, CDC Stacks, v. 3, n. 39, 2016. Disponível em: <https://stacks.cdc.gov/view/cdc/40572>. Citado na página 48.

GALLE, P. R. et al. Easl clinical practice guidelines: management of hepatocellular carcinoma. *Journal of hepatology*, Elsevier, v. 69, n. 1, p. 182–236, 2018. Citado 6 vezes nas páginas 15, 29, 30, 31, 44 e 46.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. 1st. ed. [S.l.]: MIT Press, 2016. 100–107 p. ISBN 9780262035613. Citado na página 18.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. 1st. ed. [S.l.]: MIT Press, 2016. 168–171 p. ISBN 9780262035613. Citado na página 21.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. 1st. ed. [S.l.]: MIT Press, 2016. 326–366 p. ISBN 9780262035613. Citado na página 21.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. 1st. ed. [S.l.]: MIT press, 2016. 502–525 p. Citado na página 25.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. 1st. ed. [S.l.]: MIT Press, 2016. 228–252 p. ISBN 9780262035613. Citado 2 vezes nas páginas 55 e 56.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. 1st. ed. [S.l.]: MIT Press, 2016. 274–293 p. ISBN 9780262035613. Citado 2 vezes nas páginas 58 e 60.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. 1st. ed. [S.l.]: MIT Press, 2016. 258–268 p. ISBN 9780262035613. Citado na página 58.

Google Inc. *Google Trends*. 2019. Acesso Em: 19 set. 2019. Disponível em: <http://trends.google.com/trends>. Citado na página 32.

GRECH, V.; SAVONA-VENTURA, C.; VASSALLO-AGIUS, P. Unexplained differences in sex ratios at birth in europe and north america. *Bmj*, British Medical Journal Publishing Group, v. 324, n. 7344, p. 1010–1011, 2002. Citado na página 47.

GUO, L.-H. et al. A two-stage multi-view learning framework based computer-aided diagnosis of liver tumors with contrast enhanced ultrasound images. *Clinical hemorheology and microcirculation*, IOS Press, n. Preprint, p. 1–12, 2018. Citado na página 35.

HANNA, R. F. et al. Comparative 13-year meta-analysis of the sensitivity and positive predictive value of ultrasound, ct, and mri for detecting hepatocellular carcinoma. *Abdominal Radiology*, Springer, v. 41, n. 1, p. 71–90, 2016. Citado 2 vezes nas páginas 87 e 89.

HAYKIN, S. S.; ELEKTROINGENIEUR, K. *Neural networks and learning machines*. 3rd. ed. New Jersey: Pearson education Upper Saddle River, 2009. v. 3. 1–46 p. Citado na página 21.

HE, K. et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2015. p. 1026–1034. Citado na página 57.

HE, K. et al. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 770–778. Citado na página 23.

HE, S. et al. Accuracy of microrna s for the diagnosis of hepatocellular carcinoma: A systematic review and meta-analysis. *Clinics and research in hepatology and gastroenterology*, Elsevier, v. 40, n. 4, p. 405–417, 2016. Citado na página 87.

HEIMBACH, J. K. et al. AASLD guidelines for the treatment of hepatocellular carcinoma. *Hepatology*, Wiley Online Library, v. 67, n. 1, p. 358–380, 2018. Citado 3 vezes nas páginas 28, 29 e 46.

HERMESSI, H.; MOURALI, O.; ZAGROUBA, E. Transfer learning with multiple convolutional neural networks for soft tissue sarcoma mri classification. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Eleventh International Conference on Machine Vision (ICMV 2018)*. [S.l.], 2019. v. 11041, p. 110412L. Citado na página 86.

HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation*, v. 18, n. 7, p. 1527–54, 7 2006. ISSN 0899-7667. Disponível em: <http://www.ncbi.nlm.nih.gov/pubmed/16764513>. Citado 2 vezes nas páginas 20 e 24.

HU, J.; SHEN, L.; SUN, G. Squeeze-and-excitation networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2018. p. 7132–7141. Citado na página 23.

HUGENHOLTZ, G. G.; PORTE, R. J.; LISMAN, T. The platelet and platelet function testing in liver disease. *Clinics in liver disease*, Elsevier, v. 13, n. 1, p. 11–20, 2009. Citado na página 46.

HUMES, K. R.; JONES, N. A.; RAMIREZ, R. R. *Overview of race and Hispanic origin: 2010*. 2011. Acesso Em: 19 set. 2019. Disponível em: <https://www.census.gov/prod/cen2010/briefs/c2010br-02.pdf>. Citado na página 48.

HUTTER, C.; ZENKLUSEN, J. C. The cancer genome atlas: creating lasting value beyond its data. *Cell*, Elsevier, v. 173, n. 2, p. 283–285, 2018. Citado na página 38.

IMBERT-BISMUT, F. et al. Biochemical markers of liver fibrosis in patients with hepatitis c virus infection: a prospective study. *The Lancet*, Elsevier, v. 357, n. 9262, p. 1069–1075, 2001. Citado na página 48.

Instituto Nacional de Cancer Jose Alencar Gomes da Silva. *ABC do cancer: abordagens basicas para o controle do cancer*. 4. ed. ed. Rio de Janeiro: [s.n.], 2018. 13–16 p. ISBN 978-85-7318-316-0. Citado na página 27.

International Agency for Research on Cancer. *Cancer Tomorrow*. 2018. Acesso Em: 19 set. 2019. Disponível em: <http://gco.iarc.fr/tomorrow/home>. Citado na página 14.

International Agency for Research on Cancer. *New Global Cancer Data: GLOBOCAN 2018 | UICC*. 2018. Acesso Em: 19 set. 2019. Disponível em: <https://www.uicc.org/new-global-cancer-data-globocan-2018>. Citado na página 14.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: *International Conference on Machine Learning*. [S.l.: s.n.], 2015. p. 448–456. Citado na página 58.

JIA, Y. et al. Caffe: Convolutional architecture for fast feature embedding. In: ACM. *Proceedings of the 22nd ACM international conference on Multimedia*. [S.l.], 2014. p. 675–678. Citado na página 23.

JONES, E. et al. *SciPy: Open source scientific tools for Python*. 2019. Acesso Em: 19 set. 2019. Disponível em: <https://www.scipy.org/>. Citado na página 43.

JUN, Y. et al. Deep-learned 3D black-blood imaging using automatic labelling technique and 3D convolutional neural networks for detecting metastatic brain tumors. *Scientific Reports*, Nature Publishing Group, v. 8, n. 1, 12 2018. ISSN 20452322. Citado na página 32.

JUNSOMBOON, N.; PHIENTHRAKUL, T. Combining over-sampling and under-sampling techniques for imbalance dataset. In: ACM. *Proceedings of the 9th International Conference on Machine Learning and Computing*. [S.l.], 2017. p. 243–247. Citado na página 52.

KALLENBERG, M. et al. Unsupervised Deep Learning Applied to Breast Density Segmentation and Mammographic Risk Scoring. *IEEE Transactions on Medical Imaging*, v. 35, n. 5, p. 1322–1331, 5 2016. ISSN 0278-0062. Disponível em: <http://ieeexplore.ieee.org/document/7412749/>. Citado na página 14.

KAMNITSAS, K. et al. Efficient multi-scale 3d cnn with fully connected crf for accurate brain lesion segmentation. *Medical image analysis*, Elsevier, v. 36, p. 61–78, 2017. Citado na página 24.

KANG, H. The prevention and handling of the missing data. *Korean journal of anesthesiology*, Korean Society of Anesthesiologists, v. 64, n. 5, p. 402, 2013. Citado na página 44.

KAWAHARA, J. et al. Seven-Point Checklist and Skin Lesion Classification Using Multitask Multimodal Neural Nets. *IEEE Journal of Biomedical and Health Informatics*, Institute of Electrical and Electronics Engineers Inc., v. 23, n. 2, p. 538–546, 3 2019. ISSN 21682194. Citado na página 25.

KETKAR, N. *Deep Learning with Python: A Hands-on Introduction*. 1st. ed. Berkeley, CA: Apress, 2017. 195–208 p. ISBN 978-1-4842-2766-4. Citado na página 23.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. [s.n.], 2015. Disponível em: <http://arxiv.org/abs/1412.6980>. Citado na página 58.

KINGMA, D. P.; WELLING, M. Auto-Encoding Variational Bayes. 2013. Acesso Em: 19 set. 2019. Disponível em: <http://arxiv.org/abs/1312.6114>. Citado na página 25.

KORAM, K. et al. Population based reference intervals for common blood haematological and biochemical parameters in the akuapem north district. *Ghana medical journal*, Ghana Medical Association, v. 41, n. 4, 2007. Citado na página 48.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. USA: Curran Associates Inc., 2012. (NIPS'12), p. 1097–1105. Disponível em: <http://dl.acm.org/citation.cfm?id=2999134.2999257>. Citado 4 vezes nas páginas 22, 23, 56 e 58.

KSIĄŻEK, W. et al. A novel machine learning approach for early detection of hepatocellular carcinoma patients. *Cognitive Systems Research*, Elsevier, v. 54, p. 116–127, 2019. Citado na página 37.

KUHN, M.; JOHNSON, K. *Applied predictive modeling*. 2nd. ed. [S.l.]: Springer, 2013. 95–98 p. Citado na página 19.

KUHN, M.; JOHNSON, K. *Applied predictive modeling*. 2nd. ed. [S.l.]: Springer, 2013. 247–266 p. Citado na página 20.

LAHAT, D.; ADALI, T.; JUTTEN, C. Multimodal data fusion: an overview of methods, challenges, and prospects. *Proceedings of the IEEE*, IEEE, v. 103, n. 9, p. 1449–1477, 2015. Citado 2 vezes nas páginas 25 e 26.

LAUKAMP, K. R. et al. Fully automated detection and segmentation of meningiomas using deep learning on routine multiparametric MRI. *European Radiology*, Springer Verlag, v. 29, n. 1, p. 124–132, 1 2019. ISSN 14321084. Citado na página 32.

LEACH, P.; MEALLING, M.; SALZ, R. *RFC 4122: A universally unique identifier (uuid)*. 2005. Acesso Em: 19 set. 2019. Disponível em: <http://www.ietf.org/rfc/rfc4122.txt>. Citado na página 43.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 5 2015. ISSN 0028-0836. Disponível em: <http://www.nature.com/articles/nature14539>. Citado na página 14.

LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Taipei, Taiwan, v. 86, n. 11, p. 2278–2324, 1998. Citado na página 22.

LEE, Y. J. et al. Hepatocellular carcinoma: diagnostic performance of multidetector ct and mr imaging—a systematic review and meta-analysis. *Radiology*, Radiological Society of North America, v. 275, n. 1, p. 97–109, 2015. Citado 2 vezes nas páginas 87 e 89.

LI, Z.; HOIEM, D. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 40, n. 12, p. 2935–2947, 2017. Citado na página 23.

LI, Z. et al. Deep Learning based Radiomics (DLR) and its usage in noninvasive IDH1 prediction for low grade glioma. *Scientific Reports*, Nature Publishing Group, v. 7, n. 1, 12 2017. ISSN 20452322. Citado na página 25.

LIM, Y.-S. et al. Serum sodium, renal function, and survival of patients with end-stage liver disease. *Journal of hepatology*, Elsevier, v. 52, n. 4, p. 523–528, 2010. Citado na página 46.

LIN, H. et al. Automated classification of hepatocellular carcinoma differentiation using multiphoton microscopy and deep learning. *Journal of biophotonics*, Wiley Online Library, p. e201800435, 2019. Citado 3 vezes nas páginas 36, 88 e 89.

LINEHAN M., G. R. K.-S. L. Y. R. C. B. E. . R. *Radiology Data from The Cancer Genome Atlas Cervical Kidney renal papillary cell carcinoma [KIRP] collection*. 2016. Disponível em: <https://wiki.cancerimagingarchive.net/display/Public/TCGA-KIRP#a34f742158a14169822d8a6efc79a063>. Citado na página 39.

LITJENS, G. et al. A survey on deep learning in medical image analysis. *Medical image analysis*, Elsevier, v. 42, p. 60–88, 2017. Citado 3 vezes nas páginas 16, 20 e 39.

LIU, P.-H. et al. Prognosis of hepatocellular carcinoma: assessment of eleven staging systems. *Journal of hepatology*, Elsevier, v. 64, n. 3, p. 601–608, 2016. Citado na página 46.

LOU, J. et al. Biomarkers for hepatocellular carcinoma. *Biomarkers in cancer*, SAGE Publications Sage UK: London, England, v. 9, p. 1179299X16684640, 2017. Citado na página 91.

LUCCHESI F. R., . A.-N. D. *Radiology Data from The Cancer Genome Atlas Stomach Adenocarcinoma [TCGA-STAD] collection*. 2016. Disponível em: <https://wiki.cancerimagingarchive.net/display/Public/TCGA-STAD#83f2a7e42a374981b8085c22815065d6>. Citado na página 39.

LUO, P. et al. Current status and perspective biomarkers in afp negative hcc: Towards screening for and diagnosing hepatocellular carcinoma at an earlier stage. *Pathology & Oncology Research*, Springer, p. 1–5, 2019. Citado na página 87.

MAKHZANI, A.; FREY, B. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013. Acesso Em: 19 set. 2019. Citado na página 25.

MALIK, J. S. et al. Generating high tail accuracy gaussian random numbers in hardware using central limit theorem. In: IEEE. *2011 IEEE/IFIP 19th International Conference on VLSI and System-on-Chip*. [S.l.], 2011. p. 60–65. Citado na página 49.

MANGAI, U. G. et al. A survey of decision fusion and feature fusion strategies for pattern classification. *IETE Technical review*, Taylor & Francis, v. 27, n. 4, p. 293–307, 2010. Citado na página 26.

MARRERO, J. A. et al. Diagnosis, staging, and management of hepatocellular carcinoma: 2018 practice guidance by the american association for the study of liver diseases. *Hepatology*, Wiley Online Library, v. 68, n. 2, p. 723–750, 2018. Citado 3 vezes nas páginas 15, 28 e 46.

MEN, K.; DAI, J.; LI, Y. Automatic segmentation of the clinical target volume and organs at risk in the planning ct for rectal cancer using deep dilated convolutional neural networks. *Medical physics*, Wiley Online Library, v. 44, n. 12, p. 6377–6389, 2017. Citado na página 42.

MENEGOTTO, A. B.; BECKER, C. D. L.; CAZELLA, S. C. Computer-aided hepatocarcinoma diagnosis using multimodal deep learning. In: SPRINGER. *International Symposium on Ambient Intelligence*. [S.l.], 2019. p. 3–10. Citado na página 81.

MENG, F. et al. B-mode ultrasound based diagnosis of liver cancer with ceus images as privileged information. In: IEEE. *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. [S.l.], 2018. p. 3124–3127. Citado na página 35.

MicroDicom Inc. *MicroDicom DICOM viewer*. 2019. Acesso Em: 19 set. 2019. Disponível em: <http://www.microdicom.com/dicom-viewer.html>. Citado na página 41.

MIOTTO, R.; LI, L.; DUDLEY, J. T. Deep learning to predict patient future diseases from the electronic health records. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. [S.l.: s.n.], 2016. ISBN 9783319306704. ISSN 16113349. Citado 4 vezes nas páginas 15, 33, 88 e 89.

MITCHELL, T. M. *Machine Learning*. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 1997. 1–5 p. ISBN 0070428077, 9780070428072. Citado na página 18.

MLYNARSKI, P. et al. 3d convolutional neural networks for tumor segmentation using long-range 2d context. *Computerized Medical Imaging and Graphics*, Elsevier, v. 73, p. 60–72, 2019. Citado na página 32.

MOVAHEDI, F.; COYLE, J. L.; SEJDIĆ, E. Deep belief networks for electroencephalography: A review of recent contributions and future outlooks. *IEEE journal of biomedical and health informatics*, IEEE, v. 22, n. 3, p. 642–652, 2017. Citado na página 24.

MOVAHEDI, F.; COYLE, J. L.; SEJDIC, E. Deep belief networks for electroencephalography: A review of recent contributions and future outlooks. *IEEE Journal of Biomedical and Health Informatics*, Institute of Electrical and Electronics Engineers Inc., v. 22, n. 3, p. 642–652, 5 2018. ISSN 21682194. Citado na página 24.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. [S.l.: s.n.], 2010. p. 807–814. Citado na página 58.

National Cancer Institute Clinical Proteomic Tumor Analysis Consortium. *Radiology Data from the Clinical Proteomic Tumor Analysis Consortium Pancreatic Ductal Adenocarcinoma [CPTAC-PDA] Collection*. 2018. Acesso Em: 19 set. 2019. Disponível em: <https://wiki.cancerimagingarchive.net/display/Public/CPTAC-PDA#81570df507e4478e83710196b5b1f6c1>. Citado na página 39.

NGUYEN, D. T. et al. Hybrid multivariate pattern analysis combined with extreme learning machine for alzheimer's dementia diagnosis using multi-measure rs-fmri spatial patterns. *PloS one*, Public Library of Science, v. 14, n. 2, p. e0212582, 2019. Citado na página 32.

O'BRIEN, A.; WILLIAMS, R. Nutrition in end-stage liver disease: principles and practice. *Gastroenterology*, Elsevier, v. 134, n. 6, p. 1729–1740, 2008. Citado na página 45.

PARK, S. J. et al. Usefulness of afp, afp-l3, and pivka-ii, and their combinations in diagnosing hepatocellular carcinoma. *Medicine*, Wolters Kluwer Health, v. 96, n. 11, 2017. Citado na página 91.

PEACOCK, A. et al. Global statistics on alcohol, tobacco and illicit drug use: 2017 status report. *Addiction*, Wiley Online Library, v. 113, n. 10, p. 1905–1926, 2018. Citado na página 48.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011. Citado na página 68.

PIZER, S. M. et al. Adaptive histogram equalization and its variations. *Computer vision, graphics, and image processing*, Elsevier, v. 39, n. 3, p. 355–368, 1987. Citado na página 42.

PRAVEEN, G. et al. Combination of hand-crafted and unsupervised learned features for ischemic stroke lesion detection from magnetic resonance images. *Biocybernetics and Biomedical Engineering*, Elsevier, 2019. Citado na página 32.

PRICE, S. et al. Operations research models and methods in the screening, detection, and treatment of prostate cancer: A categorized, annotated review. *Operations Research for Health Care*, 2016. ISSN 22116923. Citado na página 14.

PROVAN, D. et al. *Oxford handbook of clinical haematology*. 2nd. ed. [S.l.]: Oxford university press, 2009. 688 p. Citado na página 51.

QIAN, Y. et al. Multimodal ultrasound imaging based diagnosis of liver cancers with a two-stage multi-view learning framework. In: IEEE. *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. [S.l.], 2017. p. 3232–3235. Citado na página 35.

RAMACHANDRAM, D.; TAYLOR, G. W. Deep multimodal learning: A survey on recent advances and trends. *IEEE Signal Processing Magazine*, IEEE, v. 34, n. 6, p. 96–108, 2017. Citado na página 26.

RAMKUMAR, N. et al. Prediction of liver cancer using conditional probability bayes theorem. In: IEEE. *2017 International Conference on Computer Communication and Informatics (ICCCI)*. [S.l.], 2017. p. 1–5. Citado 2 vezes nas páginas 15 e 34.

RASTOGI, A. Changing role of histopathology in the diagnosis and management of hepatocellular carcinoma. *World journal of gastroenterology*, Baishideng Publishing Group Inc, v. 24, n. 35, p. 4000, 2018. Citado na página 90.

RATTANI, A.; DERAKHSHANI, R. On fine-tuning convolutional neural networks for smartphone based ocular recognition. In: IEEE. *2017 IEEE International Joint Conference on Biometrics (IJCB)*. [S.l.], 2017. p. 762–767. Citado na página 86.

RONOT, M. et al. Comparison of the accuracy of aasld and li-rads criteria for the non-invasive diagnosis of hcc smaller than 3 cm. *Journal of hepatology*, Elsevier, v. 68, n. 4, p. 715–723, 2018. Citado na página 87.

ROSENTHAL, P.; PINCUS, M.; FINK, D. Sex-and age-related differences in bilirubin concentrations in serum. *Clinical chemistry*, Clinical Chemistry, v. 30, n. 8, p. 1380–1382, 1984. Citado na página 50.

RUSSAKOVSKY, O. et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, v. 115, n. 3, p. 211–252, 2015. Citado na página 22.

RUSSELL, S. J.; NORVIG, P. *Artificial intelligence: a modern approach*. 3rd. ed. [S.l.]: Pearson Education Limited,, 2016. 727–744 p. Citado na página 21.

RUSSELL, S. J. S. J.; NORVIG, P.; DAVIS, E. *Artificial intelligence : a modern approach*. 3rd. ed. [S.l.]: Prentice Hall, 2010. 693–748 p. ISBN 0136042597. Citado na página 18.

SANTOS, M. S. et al. A new cluster-based oversampling method for improving survival prediction of hepatocellular carcinoma patients. *Journal of biomedical informatics*, Elsevier, v. 58, p. 49–59, 2015. Citado na página 48.

SHERLOCK, S.; DOOLEY, J. et al. *Diseases of the liver and biliary system*. 12th. ed. [S.l.]: Wiley Online Library, 2012. 671–679 p. Citado na página 27.

SHERLOCK, S.; DOOLEY, J. et al. *Diseases of the liver and biliary system*. 12th. ed. [S.l.]: Wiley Online Library, 2012. 681–698 p. Citado 2 vezes nas páginas 27 e 46.

SHERLOCK, S.; DOOLEY, J. et al. *Diseases of the liver and biliary system*. 12th. ed. [S.l.]: Wiley Online Library, 2012. 660–669 p. Citado na página 40.

SIBI, P.; JONES, S. A.; SIDDARTH, P. Analysis of different activation functions using back propagation neural networks. *Journal of Theoretical and Applied Information Technology*, v. 47, n. 3, p. 1264–1268, 2013. Citado na página 58.

SILVA-RAMÍREZ, E.-L. et al. Missing value imputation on missing completely at random data using multilayer perceptrons. *Neural Networks*, Elsevier, v. 24, n. 1, p. 121–129, 2011. Citado na página 48.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. [s.n.], 2015. Disponível em: <http://arxiv.org/abs/1409.1556>. Citado 2 vezes nas páginas 23 e 61.

SINGANAMALLI, A.; WANG, H.; MADABHUSHI, A. Cascaded multi-view canonical correlation (camcco) for early diagnosis of alzheimer's disease via fusion of clinical, imaging and omic features. *Scientific reports*, Nature Publishing Group, v. 7, n. 1, p. 8137, 2017. Citado na página 32.

SMOLENSKY, P. *Information Processing in Dynamical Systems: Foundations of Harmony Theory*. Fort Belvoir, 1986. Acesso Em: 19 set. 2019. Disponível em: <https://apps.dtic.mil/dtic/tr/fulltext/u2/a620727.pdf>. Citado na página 24.

SOKOLOVA, M.; JAPKOWICZ, N.; SZPAKOWICZ, S. Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation. In: SATTAR, A.; KANG, B.-h. (Ed.). *AI 2006: Advances in Artificial Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 1015–1021. ISBN 978-3-540-49788-2. Citado na página 20.

SONG, B. et al. Automatic classification of dual-modalilty, smartphone-based oral dysplasia and malignancy images using deep learning. *Biomedical Optics Express*, The Optical Society, v. 9, n. 11, p. 5318, 10 2018. ISSN 2156-7085. Citado na página 26.

STICOVA, E.; JIRSA, M. New insights in bilirubin metabolism and their clinical implications. *World Journal of Gastroenterology: WJG*, Baishideng Publishing Group Inc, v. 19, n. 38, p. 6398, 2013. Citado na página 46.

SUN, W.; ZHENG, B.; QIAN, W. Computer aided lung cancer diagnosis with deep learning algorithms. In: TOURASSI, G. D.; ARMATO, S. G. (Ed.). International Society for Optics and Photonics, 2016. v. 9785, p. 97850Z. Disponível em: <http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2216307>. Citado na página 14.

SUTSKEVER, I. et al. On the importance of initialization and momentum in deep learning. In: *International conference on machine learning*. [S.l.: s.n.], 2013. p. 1139–1147. Citado na página 60.

SWATI, Z. N. K. et al. Brain tumor classification for mr images using transfer learning and fine-tuning. *Computerized Medical Imaging and Graphics*, Elsevier, v. 75, p. 34–46, 2019. Citado na página 86.

SZEGEDY, C. et al. Going deeper with convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 1–9. Citado 2 vezes nas páginas 23 e 59.

SZEGEDY, C. et al. Rethinking the inception architecture for computer vision. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 2818–2826. Citado na página 56.

THARWAT, A. Classification assessment methods. *Applied Computing and Informatics*, Elsevier, 2018. ISSN 2210-8327. In Press, Corrected Proof. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2210832718301546>. Citado 2 vezes nas páginas 20 e 67.

TIAN, J. et al. A diagnostic report generator from ct volumes on liver tumor with semi-supervised attention mechanism. In: SPRINGER. *International Conference on Medical Image Computing and Computer-Assisted Intervention*. [S.l.], 2018. p. 702–710. Citado na página 36.

TRIPODI, A.; MANNUCCI, P. M. The coagulopathy of chronic liver disease. *New England Journal of Medicine*, Mass Medical Soc, v. 365, n. 2, p. 147–156, 2011. Citado na página 46.

TSAI, W.-C. et al. Influence of the time interval from diagnosis to treatment on survival for early-stage liver cancer. *PloS one*, Public Library of Science, v. 13, n. 6, p. e0199532, 2018. Citado 2 vezes nas páginas 14 e 17.

US National Library of Medicine. *Pubmed*. 2019. Acesso Em: 19 set. 2019. Disponível em: <https://www.ncbi.nlm.nih.gov/pubmed/>. Citado na página 32.

VALERY, P. C. et al. Projections of primary liver cancer to 2030 in 30 countries worldwide. *Hepatology*, Wiley Online Library, v. 67, n. 2, p. 600–611, 2018. Citado na página 14.

VANG, Y. S.; CHEN, Z.; XIE, X. Deep learning framework for multi-class breast cancer histology image classification. In: SPRINGER. *International Conference Image Analysis and Recognition*. [S.l.], 2018. p. 914–922. Citado na página 52.

VINCENT, P. et al. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, v. 11, n. Dec, p. 3371–3408, 2010. Citado na página 25.

WAGHRAY, A.; MURALI, A. R.; MENON, K. N. Hepatocellular carcinoma: From diagnosis to treatment. *World journal of hepatology*, Baishideng Publishing Group Inc, v. 7, n. 8, p. 1020, 2015. Citado na página 48.

WALJEE, A. K. et al. Comparison of imputation methods for missing laboratory data in medicine. *BMJ open*, British Medical Journal Publishing Group, v. 3, n. 8, p. e002847, 2013. Citado na página 44.

WALT, S. Van der et al. scikit-image: image processing in python. *PeerJ*, PeerJ Inc., v. 2, p. 453, 2014. Citado na página 42.

WANG, J. et al. Development and Evaluation of Novel Statistical Methods in Urine Biomarker-Based Hepatocellular Carcinoma Screening. *Scientific Reports*, Nature Publishing Group, v. 8, n. 1, p. 3799, 12 2018. ISSN 2045-2322. Disponível em: <http://www.nature.com/articles/s41598-018-21922-9>. Citado 2 vezes nas páginas 15 e 34.

WANG, Q. et al. Malignancy characterization of hepatocellular carcinoma using hybrid texture and deep features. In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017. p. 4162–4166. ISBN 978-1-5090-2175-8. Disponível em: <http://ieeexplore.ieee.org/document/8297066/>. Citado 3 vezes nas páginas 36, 88 e 89.

WANG, X.; WANG, Q. Alpha-fetoprotein and hepatocellular carcinoma immunity. *Canadian Journal of Gastroenterology and Hepatology*, Hindawi, v. 2018, 2018. Citado na página 87.

WANG, Z. et al. Development of diagnostic model of lung cancer based on multiple tumor markers and data mining. *Oncotarget*, Impact Journals, LLC, v. 8, n. 55, p. 94793, 2017. Citado na página 14.

WEAVING, G.; BATSTONE, G. F.; JONES, R. G. Age and sex variation in serum albumin concentration: an observational study. *Annals of clinical biochemistry*, SAGE Publications Sage UK: London, England, v. 53, n. 1, p. 106–111, 2016. Citado 3 vezes nas páginas 46, 49 e 50.

Working Subgroup for Clinical Practice Guidelines for Primary Biliary Cirrhosis. Guidelines for the management of primary biliary cirrhosis: The intractable hepatobiliary disease study group supported by the ministry of health, labour and welfare of japan. *Hepatology Research*, Wiley Online Library, v. 44, p. 71–90, 2014. Citado na página 28.

XIE, S. et al. Aggregated residual transformations for deep neural networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 1492–1500. Citado na página 23.

XUE, Z. et al. Localizing tuberculosis in chest radiographs with deep learning. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Medical imaging 2018: imaging informatics for healthcare, research, and applications*. [S.l.], 2018. v. 10579, p. 105790U. Citado na página 52.

YADAV, S.; SHUKLA, S. Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification. In: IEEE. *2016 IEEE 6th International Conference on Advanced Computing (IACC)*. [S.l.], 2016. p. 78–83. Citado 2 vezes nas páginas 51 e 90.

YAN, X. et al. Classification of lung nodule malignancy risk on computed tomography images using convolutional neural network: A comparison between 2d and 3d strategies. In: SPRINGER. *Asian Conference on Computer Vision*. [S.l.], 2016. p. 91–101. Citado na página 89.

YUAN, K.-H.; BENTLER, P. M. Consistency of normal-distribution-based pseudo maximum likelihood estimates when data are missing at random. *The American Statistician*, Taylor & Francis, v. 64, n. 3, p. 263–267, 2010. Citado na página 51.

ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. In: FLEET, D. et al. (Ed.). *Computer Vision – ECCV 2014*. Cham: Springer International Publishing, 2014. p. 818–833. ISBN 978-3-319-10590-1. Citado na página 23.

ZHANG, J. et al. Classification of medical images in the biomedical literature by jointly using deep and handcrafted visual features. *IEEE journal of biomedical and health informatics*, IEEE, v. 22, n. 5, p. 1521–1530, 2017. Citado na página 86.

ZHAO, J. et al. Supervised brain tumor segmentation based on gradient and context-sensitive features. *Frontiers in Neuroscience*, Frontiers Media SA, v. 13, 2019. Citado na página 32.

ZHENG, R. et al. Liver cancer incidence and mortality in china: Temporal trends and projections to 2030. *Chinese Journal of Cancer Research*, Beijing Institute for Cancer Research, v. 30, n. 6, p. 571, 2018. Citado na página 14.

ZHOU, L. et al. Edmondson-steiner grade: A crucial predictor of recurrence and survival in hepatocellular carcinoma without microvascular invasio. *Pathology-Research and Practice*, Elsevier, v. 213, n. 7, p. 824–830, 2017. Citado na página 36.

ZHOU, W. et al. Grading of hepatocellular carcinoma based on diffusion weighted images with multiple b-values using convolutional neural networks. *Medical physics*, Wiley Online Library, 2019. Citado 3 vezes nas páginas 36, 88 e 89.

Apêndices

APÊNDICE A – Artefatos de *Software* Desenvolvidos

A.1 ImageCount.py

```
import os
import pydicom

SRC_DIR="/home/amenegotto/Desktop/tcga-lihc"
DEST_DIR="/tmp/tcga-lihc"

if not os.path.exists(DEST_DIR):
    os.makedirs(DEST_DIR)
qtlist=[]
pathlist=[]
for dirpath, dirs, files in os.walk(SRC_DIR):
        path = dirpath.split('/')
        qt_total=0
        qt_ct=0
        qt_mr=0
        for f in files:
                qt_total = qt_total + 1
                if os.path.splitext(f)[1] == ".dcm":
                        ds = pydicom.dcmread(dirpath + "/" + f)
                        if ds.Modality=="CT":
                                qt_ct = qt_ct + 1
                        elif ds.Modality=="MR":
                                qt_mr = qt_mr + 1
        if qt_ct > 0:
                qtlist.append(qt_ct)
                pathlist.append(dirpath)
qttot=0
qtmin=99999999
qtmax=0
qtavg=0
for it in range(len(qtlist)):
        print(pathlist[it], ' => ', qtlist[it])
        qttot += qtlist[it]
        if qtlist[it] < qtmin:
                qtmin = qtlist[it]
        if qtlist[it] > qtmax:
                qtmax = qtlist[it]
print("Total: ", qttot)
print("Min: ", qtmin)
print("Max: ", qtmax)
print("Avg: ", qttot / len(qtlist))
qt = qtmin
for path in pathlist:
        s = path.split("/")
        study=s[5]
        basedir=DEST_DIR + '/' + study
        nextid=1
        if not os.path.exists(basedir):
```

```
            os.makedirs(basedir)
        while (1>0):
            currid=basedir + '/' + str(nextid)
            if not os.path.exists(currid):
                os.makedirs(currid)
                break
            nextid = nextid + 1
        currcount=0
        for f in os.listdir(path):
            os.system("cp " + path.replace(" ", "\ ") + "/" + f + " " + currid + "/" + f)
            currcount = currcount + 1
            if currcount > qt:
                break;
```

A.2 ImageCleanup.py

```
import os
import pydicom

SRC_DIR="C:/Users/hp/Downloads/tcga-lihc/TCGA-LIHC_CT_DCM/"
for dirpath, dirs, files in os.walk(SRC_DIR):
        path = dirpath.split('/')

        for f in files:
            if os.path.splitext(f)[1] == ".dcm":
                ds = pydicom.dcmread(dirpath + "/" + f)

                if any("axial" in s.lower() for s in ds.ImageType):
                    print(ds.ImageType)
                else:
                    print(dirpath + "/" + f)
                    print("Delete!")
                    os.remove(dirpath + "/" + f)

        if len(os.listdir(dirpath + "/")) == 0:
            os.rmdir(dirpath + "/")
```

A.3 SeriesPreProcess.py

```
import os
import pydicom
import numpy as np
import uuid
from scipy.misc import imsave
from skimage.restoration import denoise_tv_chambolle
from skimage import exposure

SRC_DIR = 'C:/Users/hp/Downloads/tcga-kirp/TCGA-KIRP_CT'
OUT_DIR = 'C:/Users/hp/Downloads/tcga-kirp/tcga-kirp-png'
CSV_FILENAME = 'images-id-with-filters.csv'
CREATE_CSV_HEADER = False
def create_image_id(file_path, file_name):
    slice_id = uuid.uuid4().hex
    path = file_path.split(os.sep)
    if 'lihc' in path[0].lower():
        hcc_class = 'POS'
    else:
```

```
        hcc_class = 'NEG'
    fname = path[1] + '_' + slice_id + '.png'
    with open(CSV_FILENAME, "a") as f:
        f.write(OUT_DIR + ',' + path[1] + ',' + path[2] + ',' + path[
            3] + ',' + file_name + ',' + slice_id + ',' + fname + ',' + hcc_class + ', , \n')
    return fname

def save_png(imgs, file_path, file_name, keep_dir=True):
    if (keep_dir):
        out_path = file_path.replace(SRC_DIR, OUT_DIR)
    else:
        out_path = OUT_DIR + '/'
    os.makedirs(out_path, exist_ok=True)
    dst_name = out_path + create_image_id(file_path, file_name)
    print(file_path, file_name, ' => ', dst_name)
    imsave(dst_name, imgs)

def read_dcm(inputdir, file):
    ds = pydicom.dcmread(inputdir + '/' + file)
    image = np.stack(ds.pixel_array)
    image = image.astype(np.int16)
    image[image == -2000] = 0
    intercept = ds.RescaleIntercept
    slope = ds.RescaleSlope
    if slope != 1:
        image = slope * image.astype(np.float64)
        image = image.astype(np.int16)
    image += np.int16(intercept)
    return image

def do_adaptative_histogram(img):
    return exposure.equalize_adapthist(img, clip_limit=0.15)

def do_tv_denoise(img):
    return denoise_tv_chambolle(img, weight=0.1, multichannel=False)

def dcm_dir_convert(inputdir):
    for f in os.listdir(inputdir):
        if not os.path.isdir(inputdir + f):
            img_original = read_dcm(inputdir, f)
            img_clahe = do_adaptative_histogram(img_original)
            img_denoise = do_tv_denoise(img_clahe)
            save_png(img_denoise, inputdir, f, False)

if CREATE_CSV_HEADER:
    with open(CSV_FILENAME, "a") as x:
        x.write('base_path, patient, study, series,
        dcm_fname, slice_uid, png_fname, hcc_class, dataset, dclass \n')
# search recursively for dcm directories
pathlist=[]
for dirpath, dirs, files in os.walk(SRC_DIR):
        path = dirpath.split('/')

        for f in files:
                if os.path.splitext(f)[1] == ".dcm":
                    pathlist.append(dirpath)
                    break
for path in pathlist:
    dcm_dir_convert(path+'/')
```

A.4 ImputationWeightedRandom_SexRatio.py

```
import numpy as np

CSV_FILENAME='csv/weighted-random-sexratio.csv'
gender = np.random.choice([1, 0], 32, p=[0.56, 0.44])
with open(CSV_FILENAME, "a") as f:
    for i in range(32):
        f.write(str(gender[i]) + '\n')
```

A.5 RandomUniformImputation_Anthroprometric.py

```
import numpy as np
import pandas as pd
import math

def pounds_to_kg(val):
    return val * 0.45359237

def inches_to_cm(val):
    return val * 2.54

def truncate(number, digits) -> float:
    stepper = pow(10.0, digits)
    return math.trunc(stepper * number) / stepper

def calculate_height(d, gender, age, race, etnicity):
    for ir, rr in d[(d.Gender == gender)].iterrows():
        if race == rr['Race'] and etnicity == rr['Etnicity'] and
            age >= rr['Min Age'] and age <= rr['Max Age']:
            val = 0
            while True:
                val = np.random.normal(rr['Height Mean'], rr['Height Std'], 10000)[5000]
                if val > 0 and
                rr['Height Mean'] - rr['Height Std'] <= val <= rr['Height Mean'] +
                    rr['Height Std']:
                    break
            return val

def calculate_weight(d, gender, age, race, etnicity):
    for ir, rr in d[(d.Gender == gender)].iterrows():
        if race == rr['Race'] and etnicity == rr['Etnicity'] and
            age >= rr['Min Age'] and age <= rr['Max Age']:
            val = 0
            while True:
                val = np.random.normal(rr['Weight Mean'], rr['Weight Std'], 10000)[5000]
                if val > 0 and
                rr['Weight Mean'] - rr['Weight Std'] <= val <= rr['Weight Mean'] +
                    rr['Weight Std']:
                    break
            return val

df_rr = pd.read_csv('csv/reference-range-anthroprometric.csv')
# first convert weight (pounds => kg) and height (inches => cm) for reference ranges
for i, r in df_rr.iterrows():
    df_rr.at[i, 'Height Mean'] = inches_to_cm(r['Height Mean'])
    df_rr.at[i, 'Height Std'] = inches_to_cm(r['Height Std'])
    df_rr.at[i, 'Weight Mean'] = pounds_to_kg(r['Weight Mean'])
    df_rr.at[i, 'Weight Std'] = pounds_to_kg(r['Weight Std'])
```

```
# next step is the imputation
df = pd.read_csv('csv/input-uniform-anthroprometric.csv')
for i, r in df.iterrows():
    df.at[i, 'Height'] =
        truncate(calculate_height(df_rr, r['Gender'], r['Age'], r['Race'], r['Etnicity']), 0)
    df.at[i, 'Weight'] =
        truncate(calculate_weight(df_rr, r['Gender'], r['Age'], r['Race'], r['Etnicity']), 0)
df.to_csv('csv/uniform-inputation-anthroprometric.csv')
```

A.6 ImputationWeightedRandom_RiskFactors.py

```
import numpy as np

CSV_FILENAME='csv/weighted-random.csv'
alcohol = np.random.choice([1, 0], 109, p=[0.182, 0.818])
hemocromatosis = np.random.choice([1, 0], 109, p=[0.025, 0.975])
hepatitis = np.random.choice([1, 0], 109, p=[0.018, 0.982])
nafld = np.random.choice([1, 0], 109, p=[0.03, 0.97])
other = np.random.choice([1, 0], 109, p=[0.01, 0.99])
with open(CSV_FILENAME, "a") as f:
    for i in range(109):
        f.write(str(alcohol[i]) + ',' + str(hemocromatosis[i]) + ',' +
        str(hepatitis[i]) + ',' + str(nafld[i]) + ',' + str(other[i]) + '\n')
```

A.7 NnImputation_ExamResults.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense

def create_model():
    model = Sequential()
    model.add(Dense(6, input_dim=6, kernel_initializer='normal', activation='relu'))
    model.add(Dense(4, activation='relu'))
    model.add(Dense(1, activation='linear'))
    model.summary()
    model.compile(loss='mse', optimizer='adam', metrics=['mse','mae'])
    return model

def train_model(column_name, model, plot_loss = False):
    df_train = pd.read_csv('csv/hcc-data-spline-best-features.csv')
    # here divides in labeles / target. In my case should do for each laboratory exam result
    X = df_train.iloc[:, df_train.columns != column_name].values
    y = df_train.iloc[:, df_train.columns.get_loc(column_name)].values
    # divides in test/validation with 90/10
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
    history = model.fit(X_train, y_train, epochs=1000,
        batch_size=5, verbose=1, validation_split=0.1)
    if plot_loss:
        print(history.history.keys())
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title('model loss')
        plt.ylabel('loss')
```

```
        plt.xlabel('epoch')
        plt.legend(['train', 'validation'], loc='upper left')
        plt.show()

def populate_column(column, df):
    model = create_model
    train_model(column, model, False)
    for i, r in df.iterrows():
        if pd.isnull(r[column]) or pd.isna(r[column]):
            df.at[i, column] = model.predict(r.values())[0]

df = pd.read_csv('csv/hcc-missing-column.csv')
populate_column('AFP', df)
populate_column('Platelets', df)
populate_column('Prothrombin Time', df)
populate_column('Albumin', df)
populate_column('Total Bilirubin', df)
populate_column('Creatinine', df)
df.to_csv('csv/hcc-missing-column-filled-nn.csv')
```

A.8 ImputationWeightedRandom_Race.py

```
import numpy as np
CSV_FILENAME='csv/weighted-random-race_etnicity.csv'
race = np.random.choice([2, 1, 0], 32, p=[0.048, 0.367, 0.585])
etnicity = np.random.choice([1, 0], 32, p=[0.163, 0.837])

with open(CSV_FILENAME, "a") as f:
    for i in range(32):
        f.write(str(race[i]) + ',' + str(etnicity[i]) + '\n')
```

A.9 RandomUniformImputation_ExamResults.py

```
import numpy as np
import pandas as pd
import math

def truncate(number, digits) -> float:
    stepper = pow(10.0, digits)
    return math.trunc(stepper * number) / stepper

def calculate_value(d, exam, gender, age):
    for ir, rr in d[(d.Gender == gender) & (d.Exam == exam)].iterrows():
        if age >= rr['Min Age'] and age <= rr['Max Age']:
            val = 0
            while True:
                val = np.random.normal(rr['Mean'], rr['Standard Deviation'], 10000)[5000]
                if val > 0 and
                rr['Mean'] - rr['Standard Deviation'] <= val <= rr['Mean'] +
                rr['Standard Deviation']:
                    break
            return val

df_rr = pd.read_csv('csv/reference-range-examresults.csv')
df = pd.read_csv('csv/input-uniform-examresults-cptac.csv')
for i, r in df.iterrows():
    df.at[i, 'AFP'] = truncate(calculate_value(df_rr, 'AFP', r['Gender'], r['Age']), 0)
```

```
    df.at[i, 'Platelets'] =
        truncate(calculate_value(df_rr, 'Platelets', r['Gender'], r['Age']), 0)
    df.at[i, 'Prothrombin Time'] =
        truncate(calculate_value(df_rr, 'Prothrombin Time', r['Gender'], r['Age']), 1)
    df.at[i, 'Albumin'] =
        truncate(calculate_value(df_rr, 'Albumin', r['Gender'], r['Age']), 1)
    df.at[i, 'Total Bilirubin'] =
        truncate(calculate_value(df_rr, 'Total Bilirubin', r['Gender'], r['Age']), 1)
    df.at[i, 'Creatinine'] =
        truncate(calculate_value(df_rr, 'Creatinine', r['Gender'], r['Age']), 1)
df.to_csv('csv/uniform-inputation-examresults-cptac.csv')
```

A.10 UnimodalCnn.py

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense, BatchNormalization
from keras import backend as K
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras import regularizers
from ExecutionAttributes import ExecutionAttribute
from TimeCallback import TimeCallback
from Summary import plot_train_stats, create_results_dir,
    get_base_name, write_summary_txt, save_model, copy_to_s3
from TrainingResume import save_execution_attributes
import os
from keras.utils import plot_model
from Datasets import create_image_generator
import multiprocessing

SUMMARY_PATH = "/mnt/data/results"
NETWORK_FORMAT = "Unimodal"
IMAGE_FORMAT = "2D"
SUMMARY_BASEPATH = create_results_dir(SUMMARY_PATH, NETWORK_FORMAT, IMAGE_FORMAT)
CYCLES = 20

attr = ExecutionAttribute()
attr.img_width, attr.img_height = 96, 96
attr.path = '/mnt/data/image/2d/com_pre_proc/'
attr.summ_basename = get_base_name(SUMMARY_BASEPATH)
attr.s3_path = NETWORK_FORMAT + '/' + IMAGE_FORMAT
attr.epochs = 100
attr.batch_size = 128
attr.set_dir_names()

if K.image_data_format() == 'channels_first':
    input_s = (3, attr.img_width, attr.img_height)
else:
    input_s = (attr.img_width, attr.img_height, 3)

for i in range(0, CYCLES):
    attr.model = Sequential()
    attr.model.add(Conv2D(128, (3, 3), input_shape=input_s, kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005)))
    attr.model.add(BatchNormalization())
    attr.model.add(Activation('relu'))
    attr.model.add(Dropout(0.25))
    attr.model.add(Conv2D(128, (3, 3), input_shape=input_s, kernel_initializer='he_normal',
```

```
        kernel_regularizer=regularizers.l2(0.0005)))
    attr.model.add(BatchNormalization())
    attr.model.add(Activation('relu'))
    attr.model.add(Dropout(0.25))
    attr.model.add(MaxPooling2D(pool_size=(3, 3)))
    attr.model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
    attr.model.add(Dense(256, kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005)))
    attr.model.add(Activation('relu'))
    attr.model.add(Dropout(0.40))
    attr.model.add(Dense(256, kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005)))
    attr.model.add(Activation('relu'))
    attr.model.add(Dropout(0.40))
    attr.model.add(Dense(1))
    attr.model.add(Activation('sigmoid'))
    attr.model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.00001),
                metrics=['accuracy'])
    plot_model(attr.model, to_file=attr.summ_basename + '-architecture.png')
    train_datagen = create_image_generator(True, True)
    test_datagen = create_image_generator(True, False)
    attr.train_generator = train_datagen.flow_from_directory(
        attr.train_data_dir, target_size=(attr.img_width, attr.img_height),
        batch_size=attr.batch_size, shuffle=True, class_mode='binary')
    attr.validation_generator = test_datagen.flow_from_directory(
        attr.validation_data_dir, target_size=(attr.img_width, attr.img_height),
        batch_size=attr.batch_size, shuffle=True, class_mode='binary')
    attr.test_generator = test_datagen.flow_from_directory(
        attr.test_data_dir, target_size=(attr.img_width, attr.img_height),
        batch_size=1, shuffle=False, class_mode='binary')
    attr.calculate_steps()
    attr.increment_seq()
    save_execution_attributes(attr, attr.summ_basename + '-execution-attributes.properties')
    time_callback = TimeCallback()
    callbacks = [time_callback, EarlyStopping(monitor='val_acc', patience=20, mode='max',
        restore_best_weights=True),
                ModelCheckpoint(attr.curr_basename + "-ckweights.h5", mode='max', verbose=1,
                monitor='val_acc', save_best_only=True)]
    history = attr.model.fit_generator(
        attr.train_generator, steps_per_epoch=attr.steps_train,
        epochs=attr.epochs, validation_data=attr.validation_generator,
        validation_steps=attr.steps_valid, use_multiprocessing=True,
        workers=multiprocessing.cpu_count() - 1, callbacks=callbacks)
    # plot loss and accuracy
    plot_train_stats(history, attr.curr_basename + '-training_loss.png', attr.curr_basename +
        '-training_accuracy.png')
    # make sure that the best weights are loaded (even if restore_best_weights is already true)
    attr.model.load_weights(filepath=attr.curr_basename + "-ckweights.h5")
    # save model with weights for later reuse
    save_model(attr)
    # delete ckweights to save space - model file already has the best weights
    os.remove(attr.curr_basename + "-ckweights.h5")
    # create confusion matrix and report with accuracy, precision, recall, f-score
    write_summary_txt(attr, NETWORK_FORMAT, IMAGE_FORMAT, ['negative', 'positive'],
    time_callback, callbacks[1].stopped_epoch)
    K.clear_session()
copy_to_s3(attr)
```

A.11 UnimodalCnnLight.py

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense, BatchNormalization
from keras import backend as K
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras import regularizers
from ExecutionAttributes import ExecutionAttribute
from TimeCallback import TimeCallback
from Summary import plot_train_stats, create_results_dir, get_base_name, write_summary_txt,
    save_model, copy_to_s3
import os
from TrainingResume import save_execution_attributes
from keras.utils import plot_model
from Datasets import create_image_generator
import multiprocessing

SUMMARY_PATH="/mnt/data/results"
NETWORK_FORMAT = "Unimodal"
IMAGE_FORMAT = "2D"
SUMMARY_BASEPATH = create_results_dir(SUMMARY_PATH, NETWORK_FORMAT, IMAGE_FORMAT)
CYCLES = 20

attr = ExecutionAttribute()
attr.img_width, attr.img_height = 96, 96
attr.path = '/mnt/data/image/2d/com_pre_proc'
attr.summ_basename = get_base_name(SUMMARY_BASEPATH)
attr.s3_path = NETWORK_FORMAT + '/' + IMAGE_FORMAT
attr.epochs = 100
attr.batch_size = 128
attr.set_dir_names()

if K.image_data_format() == 'channels_first':
    input_s = (3, attr.img_width, attr.img_height)
else:
    input_s = (attr.img_width, attr.img_height, 3)

for i in range(0, CYCLES):
    attr.model = Sequential()
    attr.model.add(Conv2D(32, (3, 3), input_shape=input_s, kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005)))
    attr.model.add(BatchNormalization())
    attr.model.add(Activation('relu'))
    attr.model.add(Dropout(0.25))
    attr.model.add(Conv2D(32, (3, 3), input_shape=input_s, kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005)))
    attr.model.add(BatchNormalization())
    attr.model.add(Activation('relu'))
    attr.model.add(Dropout(0.25))
    attr.model.add(MaxPooling2D(pool_size=(3, 3), input_shape=input_s))
    attr.model.add(Conv2D(32, (3, 3), input_shape=input_s, kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005)))
    attr.model.add(BatchNormalization())
    attr.model.add(Activation('relu'))
    attr.model.add(Dropout(0.25))
    attr.model.add(Conv2D(32, (3, 3), input_shape=input_s, kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005)))
    attr.model.add(BatchNormalization())
```

```
    attr.model.add(Activation('relu'))
    attr.model.add(Dropout(0.25))
    attr.model.add(MaxPooling2D(pool_size=(3, 3), input_shape=input_s))
    attr.model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
    attr.model.add(Dense(512, kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005)))
    attr.model.add(Activation('relu'))
    attr.model.add(Dropout(0.40))
    attr.model.add(Dense(1024, kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005)))
    attr.model.add(Activation('relu'))
    attr.model.add(Dropout(0.40))
    attr.model.add(Dense(1))
    attr.model.add(Activation('sigmoid'))
    plot_model(attr.model, to_file=attr.summ_basename + '-architecture.png')
    attr.model.compile(loss='binary_crossentropy', optimizer=Adam(lr=0.00001),
                metrics=['accuracy'])
    plot_model(attr.model, to_file=attr.summ_basename + '-architecture.png')
    train_datagen = create_image_generator(True, True)
    test_datagen = create_image_generator(True, False)
    attr.train_generator = train_datagen.flow_from_directory(
        attr.train_data_dir, target_size=(attr.img_width, attr.img_height),
        batch_size=attr.batch_size, shuffle=True, class_mode='binary')
    attr.validation_generator = test_datagen.flow_from_directory(
        attr.validation_data_dir, target_size=(attr.img_width, attr.img_height),
        batch_size=attr.batch_size, shuffle=True, class_mode='binary')
    attr.test_generator = test_datagen.flow_from_directory(
        attr.test_data_dir, target_size=(attr.img_width, attr.img_height),
        batch_size=1, shuffle=False, class_mode='binary')
    # calculate steps based on number of images and batch size
    attr.calculate_steps()
    attr.increment_seq()
    time_callback = TimeCallback()
    callbacks = [time_callback, EarlyStopping(monitor='val_acc', patience=20, mode='max',
        restore_best_weights=True),
                 ModelCheckpoint(attr.curr_basename + "-ckweights.h5", mode='max', verbose=1,
                    monitor='val_acc', save_best_only=True)]
    save_execution_attributes(attr, attr.summ_basename + '-execution-attributes.properties')
    history = attr.model.fit_generator(
        attr.train_generator, steps_per_epoch=attr.steps_train,
        epochs=attr.epochs, validation_data=attr.validation_generator,
        validation_steps=attr.steps_valid, use_multiprocessing=True,
        workers=multiprocessing.cpu_count() - 1, callbacks=callbacks)
    # plot loss and accuracy
    plot_train_stats(history, attr.curr_basename + '-training_loss.png', attr.curr_basename +
        '-training_accuracy.png')
    # make sure that the best weights are loaded (even if restore_best_weights is already true)
    attr.model.load_weights(filepath=attr.curr_basename + "-ckweights.h5")
    # save model with weights for later reuse
    save_model(attr)
    # delete ckweights to save space - model file already has the best weights
    os.remove(attr.curr_basename + "-ckweights.h5")
    # create confusion matrix and report with accuracy, precision, recall, f-score
    write_summary_txt(attr, NETWORK_FORMAT, IMAGE_FORMAT, ['negative', 'positive'],
        time_callback, callbacks[1].stopped_epoch)
    K.clear_session()
copy_to_s3(attr)
```

A.12 InceptionFineTuning.py

```
from keras.applications.inception_v3 import InceptionV3
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D, Dropout
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.optimizers import SGD
import numpy as np
from Summary import create_results_dir, get_base_name, plot_train_stats, write_summary_txt,
    copy_to_s3
from ExecutionAttributes import ExecutionAttribute
from TimeCallback import TimeCallback
from TrainingResume import save_execution_attributes
from keras.utils import plot_model
from Datasets import create_image_generator
import multiprocessing
from keras import backend as K

SUMMARY_PATH = "/mnt/data/results"
NETWORK_FORMAT = "Unimodal"
IMAGE_FORMAT = "2D"
SUMMARY_BASEPATH = create_results_dir(SUMMARY_PATH, NETWORK_FORMAT, IMAGE_FORMAT)

attr = ExecutionAttribute()
attr.architecture = 'InceptionV3'
results_path = create_results_dir(SUMMARY_BASEPATH, 'fine-tuning', attr.architecture)
attr.summ_basename = get_base_name(results_path)
attr.s3_path = NETWORK_FORMAT + '/' + IMAGE_FORMAT
attr.path = '/mnt/data/image/2d/sem_pre_proc'
attr.set_dir_names()
# 4, 8, 16, 32, 64, 128, 256 dependent on CPU/GPU memory capacity (powers of 2 values).
attr.batch_size = 128
attr.epochs = 500
CYCLES = 5

for i in range(0, CYCLES):
    base_model = InceptionV3(weights='imagenet', include_top=False)
    attr.img_width, attr.img_height = 299, 299
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    drop = Dropout(0.20)(x)
    predictions = Dense(2, activation='softmax')(drop)
    attr.model = Model(inputs=base_model.input, outputs=predictions)
    for layer in base_model.layers:
        layer.trainable = False
    attr.model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy',
        metrics=['accuracy'], )
    train_datagen = create_image_generator(True, True)
    test_datagen = create_image_generator(True, False)
    attr.train_generator = train_datagen.flow_from_directory(
        attr.train_data_dir, target_size=(attr.img_height, attr.img_width),
        batch_size=attr.batch_size, shuffle=True, class_mode='categorical')
    attr.validation_generator = test_datagen.flow_from_directory(
        attr.validation_data_dir, target_size=(attr.img_height, attr.img_width),
        batch_size=attr.batch_size, shuffle=True, class_mode='categorical')
    attr.test_generator = test_datagen.flow_from_directory(
        attr.test_data_dir, target_size=(attr.img_height, attr.img_width),
        batch_size=1, shuffle=False, class_mode='categorical')
    attr.calculate_steps()
```

```
attr.increment_seq()
callbacks_top = [
    ModelCheckpoint(attr.curr_basename + "-mid-ckweights.h5", monitor='val_acc', verbose=1,
        save_best_only=True),
    EarlyStopping(monitor='val_acc', patience=10, verbose=0)
]
# Persist execution attributes for session resume
save_execution_attributes(attr, attr.summ_basename + '-execution-attributes.properties')
attr.model.fit_generator(
    attr.train_generator, steps_per_epoch=attr.steps_train,
    epochs=attr.epochs, validation_data=attr.validation_generator,
    validation_steps=attr.steps_valid, use_multiprocessing=True,
    workers=multiprocessing.cpu_count() - 1, callbacks=callbacks_top)
for i, layer in enumerate(base_model.layers):
    print(i, layer.name)
attr.model.load_weights(attr.curr_basename + "-mid-ckweights.h5")
time_callback = TimeCallback()
#Save the model after every epoch.
callbacks_list = [time_callback,
    ModelCheckpoint(attr.curr_basename + "-ckweights.h5", monitor='val_acc', verbose=1,
        save_best_only=True),
    EarlyStopping(monitor='val_acc', patience=50, verbose=0)
]
# train the top 2 inception blocks, i.e. we will freeze
# the first 249 layers and unfreeze the rest:
for layer in attr.model.layers[:249]:
    layer.trainable = False
for layer in attr.model.layers[249:]:
    layer.trainable = True
attr.model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy',
    metrics=['accuracy'])
plot_model(attr.model, to_file=attr.summ_basename + '-architecture.png')
history = attr.model.fit_generator(
    attr.train_generator, steps_per_epoch=attr.steps_train,
    epochs=attr.epochs, validation_data=attr.validation_generator,
    validation_steps=attr.steps_valid, use_multiprocessing=True,
    workers=multiprocessing.cpu_count() - 1, callbacks=callbacks_list)
attr.model.save(attr.curr_basename + '-weights.h5')
# Plot train stats
plot_train_stats(history, attr.curr_basename + '-training_loss.png', attr.curr_basename +
    '-training_accuracy.png')
# Reset test generator before raw predictions
attr.test_generator.reset()
# Get the filenames from the generator
fnames = attr.test_generator.filenames
# Get the ground truth from generator
ground_truth = attr.test_generator.classes
# Get the label to class mapping from the generator
label2index = attr.test_generator.class_indices
# Getting the mapping from class index to class label
idx2label = dict((v, k) for k, v in label2index.items())
# Get the predictions from the model using the generator
predictions = attr.model.predict_generator(attr.test_generator, steps=attr.steps_test,
    verbose=1)
predicted_classes = np.argmax(predictions, axis=1)
errors = np.where(predicted_classes != ground_truth)[0]
res = "No of errors = {}/{}".format(len(errors), attr.test_generator.samples)
with open(attr.curr_basename + "-predicts.txt", "a") as f:
    f.write(res)
    print(res)
```

```
        f.close()
    # Reset test generator before summary predictions
    attr.test_generator.reset()
    write_summary_txt(attr, NETWORK_FORMAT, IMAGE_FORMAT, ['negative', 'positive'],
        time_callback, callbacks_list[2].stopped_epoch)
    K.clear_session()
copy_to_s3(attr)
```

A.13 XceptionFineTuning.py

```
from keras.layers import *
from keras.applications import *
from keras.models import Model
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.optimizers import SGD
from Summary import create_results_dir, get_base_name, plot_train_stats, write_summary_txt,
    copy_to_s3
from ExecutionAttributes import ExecutionAttribute
from TimeCallback import TimeCallback
from TrainingResume import save_execution_attributes
from keras.utils import plot_model
from Datasets import create_image_generator
import multiprocessing
from keras import backend as K

SUMMARY_PATH = "/mnt/data/results"
NETWORK_FORMAT = "Unimodal"
IMAGE_FORMAT = "2D"
SUMMARY_BASEPATH = create_results_dir(SUMMARY_PATH, NETWORK_FORMAT, IMAGE_FORMAT)

attr = ExecutionAttribute()
attr.architecture = 'Xception'

results_path = create_results_dir(SUMMARY_BASEPATH, 'fine-tuning', attr.architecture)
attr.summ_basename = get_base_name(results_path)
attr.s3_path = NETWORK_FORMAT + '/' + IMAGE_FORMAT
attr.path = '/mnt/data/image/2d/com_pre_proc'
attr.set_dir_names()
# 4, 8, 16, 32, 64, 128, 256 dependent on CPU/GPU memory capacity (powers of 2 values).
attr.batch_size = 128
attr.epochs = 500
nb_classes = 2  # number of classes
based_model_last_block_layer_number = 126
attr.img_width, attr.img_height = 299, 299
learn_rate = 1e-4  # sgd learning rate
momentum = .9  # sgd momentum to avoid local minimum
CYCLES = 1

for i in range(0, CYCLES):
    base_model = Xception(input_shape=(attr.img_width, attr.img_height, 3), weights='imagenet',
        include_top=False)
    # Top Model Block
    x = GlobalAveragePooling2D()(base_model.output)
    hidden1 = Dense(1024, activation='relu', kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005))(x)
    drop = Dropout(0.20)(hidden1)
    predictions = Dense(nb_classes, activation='softmax')(drop)
    # add your top layer block to your base model
    attr.model = Model(base_model.input, predictions)
```

```
print(attr.model.summary())
# first: train only the top layers (which were randomly initialized)
# i.e. freeze all layers of the based model that is already pre-trained.
for layer in base_model.layers:
    layer.trainable = False
train_datagen = create_image_generator(True, True)
test_datagen = create_image_generator(True, False)
attr.train_generator = train_datagen.flow_from_directory(attr.train_data_dir,
                                                         target_size=(attr.img_width,
                                                         attr.img_height),
                                                         batch_size=attr.batch_size,
                                                         shuffle=True,
                                                         class_mode='categorical')
attr.validation_generator = test_datagen.flow_from_directory(attr.validation_data_dir,
                                                             target_size=(attr.img_width,
                                                             attr.img_height),
                                                             batch_size=attr.batch_size,
                                                             shuffle=True,
                                                             class_mode='categorical')
attr.test_generator = test_datagen.flow_from_directory(
        attr.test_data_dir, target_size=(attr.img_height, attr.img_width),
        batch_size=1, class_mode='categorical', shuffle=False)
attr.model.compile(optimizer=SGD(lr=0.0001,momentum=0.9), loss='categorical_crossentropy',
    metrics=['accuracy'])
plot_model(attr.model, to_file=attr.summ_basename + '-architecture.png')
# calculate steps based on number of images and batch size
attr.calculate_steps()
attr.increment_seq()
callbacks = [
    ModelCheckpoint(attr.curr_basename + "-mid-ckweights.h5", monitor='val_acc', verbose=1,
        save_best_only=True),
    EarlyStopping(monitor='val_acc', patience=10, verbose=0)
]
# Train Simple CNN
attr.model.fit_generator(attr.train_generator,
                    steps_per_epoch=attr.steps_train, epochs=attr.epochs,
                    validation_data=attr.validation_generator,
                    validation_steps=attr.steps_valid,
                    use_multiprocessing=True, workers=multiprocessing.cpu_count() - 1,
                    callbacks=callbacks)
# verbose
print("\nStarting to Fine Tune Model\n")
attr.model.load_weights(attr.curr_basename + "-mid-ckweights.h5")
for layer in attr.model.layers[:based_model_last_block_layer_number]:
    layer.trainable = False
for layer in attr.model.layers[based_model_last_block_layer_number:]:
    layer.trainable = True
attr.model.compile(optimizer=SGD(lr=0.0001, momentum=0.9),
              loss='categorical_crossentropy',
               metrics=['accuracy'])
time_callback = TimeCallback()
# save weights of best training epoch: monitor either val_loss or val_acc
callbacks_list = [time_callback,
    ModelCheckpoint(attr.curr_basename + "-ckweights.h5", monitor='val_acc', verbose=1,
        save_best_only=True),
    EarlyStopping(monitor='val_acc', patience=50, verbose=0)
]
# Persist execution attributes for session resume
save_execution_attributes(attr, attr.summ_basename + '-execution-attributes.properties')
# fine-tune the model
```

```
    history = attr.model.fit_generator(attr.train_generator, steps_per_epoch=attr.steps_train,
                        epochs=attr.epochs, validation_data=attr.validation_generator,
                        validation_steps=attr.steps_valid, use_multiprocessing=True,
                        workers=multiprocessing.cpu_count() - 1, callbacks=callbacks_list)
    # Save the model
    attr.model.save(attr.curr_basename + '-weights.h5')
    # Plot train stats
    plot_train_stats(history, attr.curr_basename + '-training_loss.png', attr.curr_basename +
        '-training_accuracy.png')
    # Reset test generator before raw predictions
    attr.test_generator.reset()
    # Get the filenames from the generator
    fnames = attr.test_generator.filenames
    # Get the ground truth from generator
    ground_truth = attr.test_generator.classes
    # Get the label to class mapping from the generator
    label2index = attr.test_generator.class_indices
    # Getting the mapping from class index to class label
    idx2label = dict((v, k) for k, v in label2index.items())
    # Get the predictions from the model using the generator
    predictions = attr.model.predict_generator(attr.test_generator, steps=attr.steps_test,
        verbose=1)
    predicted_classes = np.argmax(predictions, axis=1)
    errors = np.where(predicted_classes != ground_truth)[0]
    res = "No of errors = {}/{}".format(len(errors), attr.test_generator.samples)
    with open(attr.curr_basename + "-predicts.txt", "a") as f:
        f.write(res)
        print(res)
        f.close()
    # Reset test generator before summary predictions
    attr.test_generator.reset()
    write_summary_txt(attr, NETWORK_FORMAT, IMAGE_FORMAT, ['negative', 'positive'],
        time_callback, callbacks_list[2].stopped_epoch)
    K.clear_session()
copy_to_s3(attr)
```

A.14 VggFineTuning.py

```
import numpy as np
from keras.applications import VGG19
from keras import models
from keras import layers
from keras import optimizers
from keras.callbacks import EarlyStopping, ModelCheckpoint
from Summary import create_results_dir, get_base_name, plot_train_stats, write_summary_txt,
    copy_to_s3
from ExecutionAttributes import ExecutionAttribute
from TimeCallback import TimeCallback
from TrainingResume import save_execution_attributes
from keras.utils import plot_model
from Datasets import create_image_generator
import multiprocessing
from keras import backend as K

# Summary Information
SUMMARY_PATH = "/mnt/data/results"
NETWORK_FORMAT = "Unimodal"
IMAGE_FORMAT = "2D"
SUMMARY_BASEPATH = create_results_dir(SUMMARY_PATH, NETWORK_FORMAT, IMAGE_FORMAT)
```

```
CYCLES = 5

# Execution Attributes
attr = ExecutionAttribute()
attr.architecture = 'vgg19'
results_path = create_results_dir(SUMMARY_BASEPATH, 'fine-tuning', attr.architecture)
attr.summ_basename = get_base_name(results_path)
attr.s3_path = NETWORK_FORMAT + '/' + IMAGE_FORMAT
attr.path = '/mnt/data/image/2d/com_pre_proc'
attr.set_dir_names()
attr.batch_size = 128
attr.epochs = 500
attr.img_width = 224
attr.img_height = 224

for i in range(0, CYCLES):
    #Load the VGG model
    vgg_conv = VGG19(weights='imagenet', include_top=False,
        input_shape=(attr.img_width, attr.img_height, 3))
    # Freeze the layers except the last 4 layers
    for layer in vgg_conv.layers[:-4]:
        layer.trainable = False
    # Check the trainable status of the individual layers
    for layer in vgg_conv.layers:
        print(layer, layer.trainable)
    # Create the model
    attr.model = models.Sequential()
    # Add the vgg convolutional base model
    attr.model.add(vgg_conv)
    # Add new layers
    attr.model.add(layers.Flatten())
    attr.model.add(layers.Dense(1024, activation='relu'))
    attr.model.add(layers.Dropout(0.3))
    attr.model.add(layers.Dense(2, activation='softmax'))
    # Show a summary of the model. Check the number of trainable parameters
    attr.model.summary()
    plot_model(attr.model, to_file=attr.summ_basename + '-architecture.png')
    # prepare data augmentation configuration
    train_datagen = create_image_generator(True, True)
    test_datagen = create_image_generator(True, False)
    attr.train_generator = train_datagen.flow_from_directory(
        attr.train_data_dir, target_size=(attr.img_height, attr.img_width),
        batch_size=attr.batch_size, class_mode='categorical', shuffle=True)
    attr.validation_generator = test_datagen.flow_from_directory(
            attr.validation_data_dir, target_size=(attr.img_height, attr.img_width),
            batch_size=attr.batch_size, class_mode='categorical', shuffle=True)
    attr.test_generator = test_datagen.flow_from_directory(
            attr.test_data_dir, target_size=(attr.img_height, attr.img_width),
            batch_size=1, class_mode='categorical', shuffle=False)
    # calculate steps based on number of images and batch size
    attr.calculate_steps()
    attr.increment_seq()
    time_callback = TimeCallback()
    callbacks = [time_callback, EarlyStopping(monitor='val_acc', patience=50, mode='max',
        restore_best_weights=True),
                 ModelCheckpoint(attr.curr_basename + "-ckweights.h5", mode='max', verbose=1,
                    monitor='val_acc', save_best_only=True)]
    # Compile the model
    attr.model.compile(loss='categorical_crossentropy', optimizer=optimizers.SGD(lr=0.002),
        metrics=['acc'])
```

```
    # Persist execution attributes for session resume
    save_execution_attributes(attr, attr.summ_basename + '-execution-attributes.properties')
    # Train the model
    history = attr.model.fit_generator(
          attr.train_generator, steps_per_epoch=attr.steps_train,
          epochs=attr.epochs, validation_data=attr.validation_generator,
          validation_steps=attr.steps_valid, callbacks=callbacks,
          use_multiprocessing=True, workers=multiprocessing.cpu_count() - 1, verbose=1)
    # Save the model
    attr.model.save(attr.curr_basename + '-weights.h5')
    # Plot train stats
    plot_train_stats(history, attr.curr_basename + '-training_loss.png', attr.curr_basename +
        '-training_accuracy.png')
    # Reset test generator before raw predictions
    attr.test_generator.reset()
    # Get the filenames from the generator
    fnames = attr.test_generator.filenames
    # Get the ground truth from generator
    ground_truth = attr.test_generator.classes
    # Get the label to class mapping from the generator
    label2index = attr.test_generator.class_indices
    # Getting the mapping from class index to class label
    idx2label = dict((v, k) for k, v in label2index.items())
    # Get the predictions from the model using the generator
    predictions = attr.model.predict_generator(attr.test_generator, steps=attr.steps_test,
        verbose=1)
    predicted_classes = np.argmax(predictions, axis=1)
    errors = np.where(predicted_classes != ground_truth)[0]
    res = "No of errors = {}/{}".format(len(errors), attr.test_generator.samples)
    with open(attr.curr_basename + "-predicts.txt", "a") as f:
        f.write(res)
        print(res)
        f.close()
    # Reset test generator before summary predictions
    attr.test_generator.reset()
    write_summary_txt(attr, NETWORK_FORMAT, IMAGE_FORMAT, ['negative', 'positive'],
        time_callback, callbacks[1].stopped_epoch)
    K.clear_session()
copy_to_s3(attr)
```

A.15 MultimodalCnnCustomGenerator.py

```
from keras.utils import plot_model
from keras.layers import Conv2D, MaxPooling2D, Input, concatenate
from keras.layers import Activation, Dropout, Flatten, Dense, BatchNormalization
from keras import backend as K
from keras.models import Model
from keras.optimizers import SGD
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras import regularizers
from ExecutionAttributes import ExecutionAttribute
from TimeCallback import TimeCallback
from Summary import plot_train_stats, create_results_dir, get_base_name, write_summary_txt,
    save_model, copy_to_s3
from TrainingResume import save_execution_attributes
import os
from MultimodalGenerator import MultimodalGenerator
import multiprocessing
import gc
```

```
# when running on p3.2xlarge
os.environ["HDF5_USE_FILE_LOCKING"]="FALSE"

# Summary Information
IMG_TYPE = "com_pre_proc/"
SUMMARY_PATH = "/mnt/data/results"
NETWORK_FORMAT = "Multimodal"
IMAGE_FORMAT = "2D"
SUMMARY_BASEPATH = create_results_dir(SUMMARY_PATH, NETWORK_FORMAT, IMAGE_FORMAT)
INTERMEDIATE_FUSION = False
LATE_FUSION = True
CYCLES = 20

# Execution Attributes
attr = ExecutionAttribute()
# dimensions of our images.
attr.img_width, attr.img_height = 96, 96
# network parameters
attr.csv_path = 'csv/clinical_data.csv'
attr.numpy_path = '/mnt/data/image/2d/numpy/' + IMG_TYPE
# attr.numpy_path = '/home/amenegotto/dataset/2d/numpy/' + IMG_TYPE
attr.path = '/mnt/data/image/2d/' + IMG_TYPE
attr.summ_basename = get_base_name(SUMMARY_BASEPATH)
attr.s3_path = NETWORK_FORMAT + '/' + IMAGE_FORMAT
attr.epochs = 100
attr.batch_size = 128
attr.set_dir_names()
if K.image_data_format() == 'channels_first':
    input_image_s = (3, attr.img_width, attr.img_height)
else:
    input_image_s = (attr.img_width, attr.img_height, 3)
input_attributes_s = (20,)
for i in range(0, CYCLES):
    # image input
    visible = Input(shape=input_image_s)
    conv1 = Conv2D(128, (3, 3), kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005))(visible)
    bn1 = BatchNormalization()(conv1)
    act1 = Activation('relu')(bn1)
    drop1 = Dropout(0.25)(act1)
    conv2 = Conv2D(128, (3, 3), kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005))(drop1)
    bn2 = BatchNormalization()(conv2)
    act2 = Activation('relu')(bn2)
    drop2 = Dropout(0.25)(act2)
    pool1 = MaxPooling2D(pool_size=(2, 2))(drop2)
    flat = Flatten()(pool1)

    if INTERMEDIATE_FUSION:
        attr.fusion = "Intermediate Fusion"
        attributes_input = Input(shape=input_attributes_s)
        concat = concatenate([flat, attributes_input])
        hidden1 = Dense(128, kernel_initializer='he_normal',
            activation='relu', kernel_regularizer=regularizers.l2(0.0005))(concat)
        drop3 = Dropout(0.20)(hidden1)
        hidden2 = Dense(64, activation='relu', kernel_initializer='he_normal',
                        kernel_regularizer=regularizers.l2(0.0005))(drop3)
        drop4 = Dropout(0.20)(hidden2)
        output = Dense(1, activation='sigmoid')(drop4)
```

```
    attr.model = Model(inputs=[visible, attributes_input], outputs=output)

if LATE_FUSION:
    attr.fusion = "Late Fusion"
    hidden1 = Dense(128, activation='relu', kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005))(flat)
    drop5 = Dropout(0.30)(hidden1)
    hidden2 = Dense(128, activation='relu', kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005))(drop5)
    drop6 = Dropout(0.30)(hidden2)
    output_img = Dense(1, activation='sigmoid')(drop6)
    model_img = Model(inputs=visible, outputs=output_img)
    attributes_input = Input(shape=input_attributes_s)
    hidden3 = Dense(128, activation='relu')(attributes_input)
    drop6 = Dropout(0.20)(hidden3)
    hidden4 = Dense(64, activation='relu')(drop6)
    drop7 = Dropout(0.20)(hidden4)
    output_attributes = Dense(1, activation='sigmoid')(drop7)
    model_attr = Model(inputs=attributes_input, outputs=output_attributes)
    concat = concatenate([model_img.output, model_attr.output])
    hidden5 = Dense(8, activation='relu')(concat)
    output = Dense(1, activation='sigmoid')(hidden5)
    attr.model = Model(inputs=[model_img.input, model_attr.input], outputs=output)

plot_model(attr.model, to_file=attr.summ_basename + '-architecture.png')
attr.model.compile(loss='binary_crossentropy',
              optimizer=SGD(lr=0.0001, momentum=0.9),
              metrics=['accuracy'])
attr.train_generator = MultimodalGenerator(
        npy_path = attr.numpy_path + '/train.npy',
        batch_size = attr.batch_size, height = attr.img_height,
        width = attr.img_width, channels = 3,
        classes = 2, should_shuffle = True,
        is_categorical = False, is_debug = False,
        width_shift = 0.2, height_shift = 0.2,
        rotation_angle = 15, shear_factor = 10, zoom_factor = 0.2)
attr.validation_generator = MultimodalGenerator(
        npy_path = attr.numpy_path + '/valid.npy',
        batch_size = attr.batch_size, height = attr.img_height,
        width = attr.img_width, channels = 3,
        classes = 2, should_shuffle = True,
        is_categorical = False, is_debug = False,
        width_shift = 0.2, height_shift = 0.2,
        rotation_angle = 15, shear_factor = 10, zoom_factor = 0.2)
attr.test_generator = MultimodalGenerator(
        npy_path = attr.numpy_path + '/test.npy',
        batch_size = 1, height = attr.img_height,
        width = attr.img_width, channels = 3,
        classes = 2, should_shuffle = False,
        is_categorical = False, is_debug = False)
print("[INFO] Calculating samples and steps...")
attr.calculate_samples_len()
attr.calculate_steps()
attr.increment_seq()
# Persist execution attributes for session resume
save_execution_attributes(attr, attr.summ_basename + '-execution-attributes.properties')
time_callback = TimeCallback()
callbacks = [time_callback, EarlyStopping(monitor='val_acc', patience=20, mode='max',
    restore_best_weights=True),
            ModelCheckpoint(attr.curr_basename + "-ckweights.h5", mode='max', verbose=1,
```

```
                    monitor='val_acc', save_best_only=True)]
    history = attr.model.fit_generator(
        attr.train_generator, steps_per_epoch=attr.steps_train,
        epochs=attr.epochs, validation_data=attr.validation_generator,
        validation_steps=attr.steps_valid, use_multiprocessing=True,
        workers=multiprocessing.cpu_count() - 1, callbacks=callbacks)

    # plot loss and accuracy
    plot_train_stats(history, attr.curr_basename + '-training_loss.png', attr.curr_basename +
        '-training_accuracy.png')
    # make sure that the best weights are loaded (even if restore_best_weights is already true)
    attr.model.load_weights(filepath=attr.curr_basename + "-ckweights.h5")
    # save model with weights for later reuse
    save_model(attr)
    # delete ckweights to save space - model file already has the best weights
    os.remove(attr.curr_basename + "-ckweights.h5")
    # create confusion matrix and report with accuracy, precision, recall, f-score
    write_summary_txt(attr, NETWORK_FORMAT, IMAGE_FORMAT, ['negative', 'positive'],
        time_callback, callbacks[1].stopped_epoch)
    K.clear_session()
    del attr.model
    del callbacks
    del history
    del attr.train_generator
    del attr.validation_generator
    del attr.test_generator
    gc.collect()

copy_to_s3(attr)
```

A.16 MultimodalCnnCustomGeneratorLight.py

```
from keras.utils import plot_model
from keras.layers import Conv2D, MaxPooling2D, Input, concatenate
from keras.layers import Activation, Dropout, Flatten, Dense, BatchNormalization
from keras import backend as K
from keras.models import Model
from keras.optimizers import SGD
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras import regularizers
from ExecutionAttributes import ExecutionAttribute
from TimeCallback import TimeCallback
from Summary import plot_train_stats, create_results_dir, get_base_name, write_summary_txt,
    save_model, copy_to_s3
from TrainingResume import save_execution_attributes
import os
from MultimodalGenerator import MultimodalGenerator
import multiprocessing

# when running on p3.2xlarge
os.environ["HDF5_USE_FILE_LOCKING"]="FALSE"

# Summary Information
IMG_TYPE = "com_pre_proc/"
SUMMARY_PATH = "/mnt/data/results"
# SUMMARY_PATH="c:/temp/results"
# SUMMARY_PATH="/tmp/results"
NETWORK_FORMAT = "Multimodal"
IMAGE_FORMAT = "2D"
```

```
SUMMARY_BASEPATH = create_results_dir(SUMMARY_PATH, NETWORK_FORMAT, IMAGE_FORMAT)
INTERMEDIATE_FUSION = False
LATE_FUSION = True
CYCLES = 20
# Execution Attributes
attr = ExecutionAttribute()
# dimensions of our images.
attr.img_width, attr.img_height = 96, 96
# network parameters
attr.csv_path = 'csv/clinical_data.csv'
attr.numpy_path = '/mnt/data/image/2d/numpy/' + IMG_TYPE
# attr.numpy_path = '/home/amenegotto/dataset/2d/numpy/' + IMG_TYPE
attr.path = '/mnt/data/image/2d/' + IMG_TYPE
attr.summ_basename = get_base_name(SUMMARY_BASEPATH)
attr.s3_path = NETWORK_FORMAT + '/' + IMAGE_FORMAT
attr.epochs = 100
attr.batch_size = 128
attr.set_dir_names()
if K.image_data_format() == 'channels_first':
    input_image_s = (3, attr.img_width, attr.img_height)
else:
    input_image_s = (attr.img_width, attr.img_height, 3)
input_attributes_s = (20,)
for i in range(0, CYCLES):
    # image input
    visible = Input(shape=input_image_s)
    conv1 = Conv2D(32, (3, 3), kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005))(visible)
    bn1 = BatchNormalization()(conv1)
    act1 = Activation('relu')(bn1)
    drop1 = Dropout(0.25)(act1)
    conv2 = Conv2D(32, (3, 3), kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005))(drop1)
    bn2 = BatchNormalization()(conv2)
    act2 = Activation('relu')(bn2)
    drop2 = Dropout(0.25)(act2)
    pool1 = MaxPooling2D(pool_size=(2, 2))(drop2)
    conv3 = Conv2D(32, (3, 3), kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005))(pool1)
    bn3 = BatchNormalization()(conv3)
    act3 = Activation('relu')(bn3)
    drop3 = Dropout(0.25)(act3)
    conv4 = Conv2D(32, (3, 3), kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005))(drop3)
    bn4 = BatchNormalization()(conv4)
    act5 = Activation('relu')(bn4)
    drop4 = Dropout(0.25)(act5)
    pool2 = MaxPooling2D(pool_size=(2, 2))(drop4)
    flat = Flatten()(pool2)

    if INTERMEDIATE_FUSION:
        attr.fusion = "Intermediate Fusion"
        attributes_input = Input(shape=input_attributes_s)
        concat = concatenate([flat, attributes_input])
        hidden1 = Dense(512, activation='relu', kernel_initializer='he_normal',
            kernel_regularizer=regularizers.l2(0.0005))(concat)
        drop5 = Dropout(0.40)(hidden1)
        hidden2 = Dense(1024, activation='relu', kernel_initializer='he_normal',
                        kernel_regularizer=regularizers.l2(0.0005))(drop5)
        drop6 = Dropout(0.40)(hidden2)
```

```
        output = Dense(1, activation='sigmoid')(drop6)
        attr.model = Model(inputs=[visible, attributes_input], outputs=output)

    if LATE_FUSION:
        attr.fusion = "Late Fusion"
        hidden1 = Dense(512, activation='relu', kernel_initializer='he_normal',
            kernel_regularizer=regularizers.l2(0.0005))(flat)
        drop5 = Dropout(0.40)(hidden1)
        hidden2 = Dense(1024, activation='relu', kernel_initializer='he_normal',
            kernel_regularizer=regularizers.l2(0.0005))(drop5)
        drop6 = Dropout(0.40)(hidden2)
        output_img = Dense(1, activation='sigmoid')(drop6)
        model_img = Model(inputs=visible, outputs=output_img)
        attributes_input = Input(shape=input_attributes_s)
        hidden3 = Dense(128, activation='relu')(attributes_input)
        drop6 = Dropout(0.20)(hidden3)
        hidden4 = Dense(256, activation='relu')(drop6)
        drop7 = Dropout(0.20)(hidden4)
        output_attributes = Dense(1, activation='sigmoid')(drop7)
        model_attr = Model(inputs=attributes_input, outputs=output_attributes)
        concat = concatenate([model_img.output, model_attr.output])
        hidden5 = Dense(8, activation='relu')(concat)
        output = Dense(1, activation='sigmoid')(hidden5)
        attr.model = Model(inputs=[model_img.input, model_attr.input], outputs=output)

    plot_model(attr.model, to_file=attr.summ_basename + '-architecture.png')
    attr.model.compile(loss='binary_crossentropy',
                  optimizer=SGD(lr=0.0001, momentum=0.9),
                  metrics=['accuracy'])
    attr.train_generator = MultimodalGenerator(
            npy_path = attr.numpy_path + '/train.npy',
            batch_size = attr.batch_size, height = attr.img_height,
            width = attr.img_width, channels = 3,
            classes = 2, should_shuffle = True,
            is_categorical = False, is_debug = False,
            width_shift = 0.2, height_shift = 0.2,
            rotation_angle = 15, shear_factor = 10, zoom_factor = 0.2)
    attr.validation_generator = MultimodalGenerator(
            npy_path = attr.numpy_path + '/valid.npy',
            batch_size = attr.batch_size, height = attr.img_height,
            width = attr.img_width, channels = 3,
            classes = 2, should_shuffle = True,
            is_categorical = False, is_debug = False,
            width_shift = 0.2, height_shift = 0.2,
            rotation_angle = 15, shear_factor = 10, zoom_factor = 0.2)
    attr.test_generator = MultimodalGenerator(
            npy_path = attr.numpy_path + '/test.npy',
            batch_size = 1, height = attr.img_height,
            width = attr.img_width, channels = 3,
            classes = 2, should_shuffle = False,
            is_categorical = False, is_debug = False)
    print("[INFO] Calculating samples and steps...")
    attr.calculate_samples_len()
    attr.calculate_steps()
    attr.increment_seq()
    # Persist execution attributes for session resume
    save_execution_attributes(attr, attr.summ_basename + '-execution-attributes.properties')
    time_callback = TimeCallback()
    callbacks = [time_callback, EarlyStopping(monitor='val_acc', patience=20, mode='max',
        restore_best_weights=True),
```

```
                ModelCheckpoint(attr.curr_basename + "-ckweights.h5", mode='max', verbose=1,
                    monitor='val_acc', save_best_only=True)]
    # training time
    history = attr.model.fit_generator(
        attr.train_generator, steps_per_epoch=attr.steps_train,
        epochs=attr.epochs, validation_data=attr.validation_generator,
        validation_steps=attr.steps_valid, use_multiprocessing=True,
        workers=multiprocessing.cpu_count() - 1, callbacks=callbacks)
    # plot loss and accuracy
    plot_train_stats(history, attr.curr_basename + '-training_loss.png', attr.curr_basename +
        '-training_accuracy.png')
    # make sure that the best weights are loaded (even if restore_best_weights is already true)
    attr.model.load_weights(filepath=attr.curr_basename + "-ckweights.h5")
    # save model with weights for later reuse
    save_model(attr)
    # delete ckweights to save space - model file already has the best weights
    os.remove(attr.curr_basename + "-ckweights.h5")
    # create confusion matrix and report with accuracy, precision, recall, f-score
    write_summary_txt(attr, NETWORK_FORMAT, IMAGE_FORMAT, ['negative', 'positive'],
        time_callback, callbacks[1].stopped_epoch)
    K.clear_session()
copy_to_s3(attr)
```

A.17 InceptionMultimodalFineTuning.py

```
from keras.applications.inception_v3 import InceptionV3
from keras.models import Model
from keras.layers import GlobalAveragePooling2D
from keras.layers import Input, concatenate
from keras.layers import Dropout, Dense
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.optimizers import SGD
from keras import regularizers
import numpy as np
from Summary import create_results_dir, get_base_name, plot_train_stats,
    write_summary_txt, copy_to_s3
from ExecutionAttributes import ExecutionAttribute
from TimeCallback import TimeCallback
from TrainingResume import save_execution_attributes
from keras.utils import plot_model
from MultimodalGenerator import MultimodalGenerator
import multiprocessing
from keras import backend as K

# Summary Information
IMG_TYPE = "sem_pre_proc/"
SUMMARY_PATH = "/mnt/data/results"
# SUMMARY_PATH="c:/temp/results"
# SUMMARY_PATH="/tmp/results"
NETWORK_FORMAT = "Multimodal"
IMAGE_FORMAT = "2D"
SUMMARY_BASEPATH = create_results_dir(SUMMARY_PATH, NETWORK_FORMAT, IMAGE_FORMAT)
INTERMEDIATE_FUSION = False
LATE_FUSION = True

# Execution Attributes
attr = ExecutionAttribute()
attr.architecture = 'InceptionV3'
attr.numpy_path = '/mnt/data/image/2d/numpy/' + IMG_TYPE
```

```
attr.path = '/mnt/data/image/2d/' + IMG_TYPE
results_path = create_results_dir(SUMMARY_BASEPATH, 'fine-tuning', attr.architecture)
attr.summ_basename = get_base_name(results_path)
attr.s3_path = NETWORK_FORMAT + '/' + IMAGE_FORMAT
attr.set_dir_names()
# 4, 8, 16, 32, 64, 128, 256 dependent on CPU/GPU memory capacity (powers of 2 values).
attr.batch_size = 128
attr.epochs = 500
# how many times to execute the training/validation/test cycle
CYCLES = 1
for i in range(0, CYCLES):
    # create the base pre-trained model
    base_model = InceptionV3(weights='imagenet', include_top=False)
    # dimensions of our images.
    # Inception input size
    attr.img_width, attr.img_height = 299, 299
    input_attributes_s = (20,)
    # Top Model Block
    glob1 = GlobalAveragePooling2D()(base_model.output)
    hidout = Dense(1024, activation='relu')(glob1)
    if INTERMEDIATE_FUSION:
        attr.fusion = "Intermediate Fusion"
        attributes_input = Input(shape=input_attributes_s)
        concat = concatenate([hidout, attributes_input])
        hidden1 = Dense(128, activation='relu')(concat)
        drop6 = Dropout(0.20)(hidden1)
        hidden2 = Dense(64, activation='relu')(drop6)
        drop6 = Dropout(0.20)(hidden2)
        output = Dense(2, activation='softmax')(drop6)
        attr.model = Model(inputs=[base_model.input, attributes_input], outputs=output)

    if LATE_FUSION:
        attr.fusion = "Late Fusion"
        output_img = Dense(2, activation='softmax')(hidout)
        model_img = Model(inputs=base_model.input, outputs=output_img)
        attributes_input = Input(shape=input_attributes_s)
        hidden3 = Dense(128, activation='relu')(attributes_input)
        drop6 = Dropout(0.20)(hidden3)
        hidden4 = Dense(64, activation='relu')(drop6)
        drop7 = Dropout(0.20)(hidden4)
        output_attributes = Dense(1, activation='sigmoid')(drop7)
        model_attr = Model(inputs=attributes_input, outputs=output_attributes)
        concat = concatenate([model_img.output, model_attr.output])
        hidden5 = Dense(8, activation='relu')(concat)
        output = Dense(2, activation='softmax')(hidden5)
        attr.model = Model(inputs=[model_img.input, model_attr.input], outputs=output)

    plot_model(attr.model, to_file=attr.summ_basename + '-architecture.png')
    # first: train only the top layers (which were randomly initialized)
    # i.e. freeze all convolutional InceptionV3 layers
    for layer in base_model.layers:
        layer.trainable = False
    # compile the model (should be done *after* setting layers to non-trainable)
    attr.model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy',
        metrics=['accuracy'], )
    attr.train_generator = MultimodalGenerator(
                npy_path = attr.numpy_path + '/train-categorical.npy',
                batch_size = attr.batch_size, height = attr.img_height,
                width = attr.img_width, channels = 3,
                classes = 2, should_shuffle = True,
```

```
            is_categorical = True, is_debug = False,
            width_shift = 0.2, height_shift = 0.2,
            rotation_angle = 15, shear_factor = 10, zoom_factor = 0.2)
attr.validation_generator = MultimodalGenerator(
            npy_path = attr.numpy_path + '/valid-categorical.npy',
            batch_size = attr.batch_size, height = attr.img_height,
            width = attr.img_width, channels = 3,
            classes = 2, should_shuffle = True,
            is_categorical = True, is_debug = False,
            width_shift = 0.2, height_shift = 0.2,
            rotation_angle = 15, shear_factor = 10, zoom_factor = 0.2)
attr.test_generator = MultimodalGenerator(
            npy_path = attr.numpy_path + '/test-categorical.npy',
            batch_size = 1, height = attr.img_height,
            width = attr.img_width, channels = 3,
            classes = 2, should_shuffle = False,
            is_categorical = True, is_debug = False)
print("[INFO] Calculating samples and steps...")
attr.calculate_samples_len()
attr.calculate_steps()
attr.increment_seq()
callbacks_top = [
    ModelCheckpoint(attr.curr_basename + "-mid-ckweights.h5", monitor='val_acc', verbose=1,
        save_best_only=True),
    EarlyStopping(monitor='val_acc', patience=10, verbose=0)
]
save_execution_attributes(attr, attr.summ_basename + '-execution-attributes.properties')
attr.model.fit_generator(
    attr.train_generator, steps_per_epoch=attr.steps_train,
    epochs=attr.epochs, validation_data=attr.validation_generator,
    validation_steps=attr.steps_valid, use_multiprocessing=True,
    workers=multiprocessing.cpu_count() - 1, callbacks=callbacks_top)
for i, layer in enumerate(base_model.layers):
    print(i, layer.name)
attr.model.load_weights(attr.curr_basename + "-mid-ckweights.h5")
time_callback = TimeCallback()
callbacks_list = [time_callback,
    ModelCheckpoint(attr.curr_basename + "-ckweights.h5", monitor='val_acc', verbose=1,
        save_best_only=True),
    EarlyStopping(monitor='val_acc', patience=50, verbose=0)
]
# train the top 2 inception blocks, i.e. we will freeze
# the first 249 layers and unfreeze the rest:
for layer in attr.model.layers[:249]:
    layer.trainable = False
for layer in attr.model.layers[249:]:
    layer.trainable = True
# we need to recompile the model for these modifications to take effect
# we use SGD with a low learning rate
attr.model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy',
    metrics=['accuracy'])
plot_model(attr.model, to_file=attr.summ_basename + '-architecture.png')
history = attr.model.fit_generator(
    attr.train_generator, steps_per_epoch=attr.steps_train,
    epochs=attr.epochs, validation_data=attr.validation_generator,
    validation_steps=attr.steps_valid, use_multiprocessing=True,
    workers=multiprocessing.cpu_count() - 1, callbacks=callbacks_list)
# Save the model
attr.model.save(attr.curr_basename + '-weights.h5')
# Plot train stats
```

```
    plot_train_stats(history, attr.curr_basename + '-training_loss.png', attr.curr_basename +
        '-training_accuracy.png')
    # Reset test generator before raw predictions
    attr.test_generator.reset()
    # Get the filenames from the generator
    fnames = attr.fnames_test
    # Get the ground truth from generator
    ground_truth = attr.test_generator.get_labels()
    # Get the predictions from the model using the generator
    predictions = attr.model.predict_generator(attr.test_generator, steps=attr.steps_test,
        verbose=1)
    predicted_classes = np.argmax(predictions, axis=1)
    errors = np.where(predicted_classes != ground_truth)[0]
    res = "No of errors = {}/{}".format(len(errors), len(attr.fnames_test))
    with open(attr.curr_basename + "-predicts.txt", "a") as f:
        f.write(res)
        print(res)
        f.close()
    # Reset test generator before summary predictions
    attr.test_generator.reset()
    write_summary_txt(attr, NETWORK_FORMAT, IMAGE_FORMAT, ['negative', 'positive'],
        time_callback, callbacks_list[2].stopped_epoch)
    K.clear_session()
copy_to_s3(attr)
```

A.18 XceptionMultimodalFineTuning.py

```
from keras.layers import *
from keras.applications import *
from keras.models import Model
from keras.layers import Input, concatenate
from keras.layers import Dropout, Dense
from keras.optimizers import SGD
from keras.callbacks import ModelCheckpoint, EarlyStopping
from Summary import create_results_dir, get_base_name, plot_train_stats, write_summary_txt,
    copy_to_s3
from ExecutionAttributes import ExecutionAttribute
from TimeCallback import TimeCallback
from TrainingResume import save_execution_attributes
from keras.utils import plot_model
from MultimodalGenerator import MultimodalGenerator
import multiprocessing
from keras import backend as K

IMG_TYPE = "com_pre_proc/"
SUMMARY_PATH = "/mnt/data/results"
# SUMMARY_PATH="c:/temp/results"
# SUMMARY_PATH="/tmp/results"
NETWORK_FORMAT = "Multimodal"
IMAGE_FORMAT = "2D"
SUMMARY_BASEPATH = create_results_dir(SUMMARY_PATH, NETWORK_FORMAT, IMAGE_FORMAT)
INTERMEDIATE_FUSION = True
LATE_FUSION = False

attr = ExecutionAttribute()
attr.architecture = 'Xception'
attr.csv_path = 'csv/clinical_data.csv'
attr.numpy_path = '/mnt/data/image/2d/numpy/' + IMG_TYPE
# attr.numpy_path = '/home/amenegotto/dataset/2d/numpy/' + IMG_TYPE
```

```
attr.path = '/mnt/data/image/2d/' + IMG_TYPE
results_path = create_results_dir(SUMMARY_BASEPATH, 'fine-tuning', attr.architecture)
attr.summ_basename = get_base_name(results_path)
attr.s3_path = NETWORK_FORMAT + '/' + IMAGE_FORMAT
attr.set_dir_names()
# 4, 8, 16, 32, 64, 128, 256 dependent on CPU/GPU memory capacity (powers of 2 values).
attr.batch_size = 128
attr.epochs = 500
# hyper parameters for model
nb_classes = 2  # number of classes
based_model_last_block_layer_number = 126  # value is based on based model selected.
attr.img_width, attr.img_height = 299, 299
input_attributes_s = (20,)
# how many times to execute the training/validation/test cycle
CYCLES = 1
for i in range(0, CYCLES):
    # Pre-Trained CNN Model using imagenet dataset for pre-trained weights
    base_model = Xception(input_shape=(attr.img_width, attr.img_height, 3), weights='imagenet',
        include_top=False)
    # Top Model Block
    glob1 = GlobalAveragePooling2D()(base_model.output)
    hidout = Dense(1024, activation='relu', kernel_initializer='he_normal',
        kernel_regularizer=regularizers.l2(0.0005))(glob1)

    if INTERMEDIATE_FUSION:
        attr.fusion = "Intermediate Fusion"
        attributes_input = Input(shape=input_attributes_s)
        concat = concatenate([hidout, attributes_input])
        hidden1 = Dense(128, activation='relu')(concat)
        drop6 = Dropout(0.20)(hidden1)
        hidden2 = Dense(64, activation='relu')(drop6)
        drop6 = Dropout(0.20)(hidden2)
        output = Dense(2, activation='softmax')(drop6)
        attr.model = Model(inputs=[base_model.input, attributes_input], outputs=output)

    if LATE_FUSION:
        attr.fusion = "Late Fusion"
        output_img = Dense(nb_classes, activation='softmax')(hidout)
        model_img = Model(inputs=base_model.input, outputs=output_img)
        attributes_input = Input(shape=input_attributes_s)
        hidden3 = Dense(128, activation='relu')(attributes_input)
        drop6 = Dropout(0.20)(hidden3)
        hidden4 = Dense(64, activation='relu')(drop6)
        drop7 = Dropout(0.20)(hidden4)
        output_attributes = Dense(1, activation='sigmoid')(drop7)
        model_attr = Model(inputs=attributes_input, outputs=output_attributes)
        concat = concatenate([model_img.output, model_attr.output])
        hidden5 = Dense(8, activation='relu')(concat)
        output = Dense(2, activation='softmax')(hidden5)
        attr.model = Model(inputs=[model_img.input, model_attr.input], outputs=output)

    plot_model(attr.model, to_file=attr.summ_basename + '-architecture.png')
    attr.train_generator = MultimodalGenerator(
                npy_path = attr.numpy_path + '/train-categorical.npy',
                batch_size = attr.batch_size, height = attr.img_height,
                width = attr.img_width, channels = 3,
                classes = 2, should_shuffle = True,
                is_categorical = True, is_debug = False,
                width_shift = 0.2, height_shift = 0.2,
                rotation_angle = 15, shear_factor = 10, zoom_factor = 0.2)
```

```
attr.validation_generator = MultimodalGenerator(
            npy_path = attr.numpy_path + '/valid-categorical.npy',
            batch_size = attr.batch_size, height = attr.img_height,
            width = attr.img_width, channels = 3,
            classes = 2, should_shuffle = True,
            is_categorical = True, is_debug = False,
            width_shift = 0.2, height_shift = 0.2,
            rotation_angle = 15, shear_factor = 10, zoom_factor = 0.2)
attr.test_generator = MultimodalGenerator(
            npy_path = attr.numpy_path + '/test-categorical.npy',
            batch_size = 1, height = attr.img_height,
            width = attr.img_width, channels = 3,
            classes = 2, should_shuffle = False,
            is_categorical = True, is_debug = False)
print("[INFO] Calculating samples and steps...")
attr.calculate_samples_len()
attr.calculate_steps()
attr.increment_seq()
# first: train only the top layers (which were randomly initialized)
# i.e. freeze all layers of the based model that is already pre-trained.
for layer in base_model.layers:
    layer.trainable = False
callbacks = [
    ModelCheckpoint(attr.curr_basename + "-mid-ckweights.h5", monitor='val_acc', verbose=1,
        save_best_only=True),
    EarlyStopping(monitor='val_acc', patience=10, verbose=0)
]
attr.model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy',
    metrics=['accuracy'])
attr.model.fit_generator(attr.train_generator,
                    steps_per_epoch=attr.steps_train,
                    epochs=attr.epochs,
                    validation_data=attr.validation_generator,
                    validation_steps=attr.steps_valid,
                    use_multiprocessing=True,
                    workers=multiprocessing.cpu_count() - 1,
                    callbacks=callbacks)
print("\nStarting to Fine Tune Model\n")
attr.model.load_weights(attr.curr_basename + "-mid-ckweights.h5")
for layer in attr.model.layers[:based_model_last_block_layer_number]:
    layer.trainable = False
for layer in attr.model.layers[based_model_last_block_layer_number:]:
    layer.trainable = True
# compile the model with a SGD/momentum optimizer
# and a very slow learning rate.
attr.model.compile(optimizer=SGD(lr=0.0001, momentum=0.9),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

time_callback = TimeCallback()
# save weights of best training epoch: monitor either val_loss or val_acc
callbacks_list = [time_callback,
    ModelCheckpoint(attr.curr_basename + "-ckweights.h5", monitor='val_acc', verbose=1,
        save_best_only=True),
    EarlyStopping(monitor='val_acc', patience=50, verbose=0)
]
# Persist execution attributes for session resume
save_execution_attributes(attr, attr.summ_basename + '-execution-attributes.properties')
# fine-tune the model
history = attr.model.fit_generator(attr.train_generator,
```

```
                        steps_per_epoch=attr.steps_train,
                        epochs=attr.epochs,
                        validation_data=attr.validation_generator,
                        validation_steps=attr.steps_valid,
                        use_multiprocessing=True,
                        workers=multiprocessing.cpu_count() - 1,
                        callbacks=callbacks_list)
    # Save the model
    attr.model.save(attr.curr_basename + '-weights.h5')
    # Reset test generator before raw predictions
    attr.test_generator.reset()
    # Plot train stats
    plot_train_stats(history, attr.curr_basename + '-training_loss.png', attr.curr_basename +
        '-training_accuracy.png')
    # Get the filenames from the generator
    fnames = attr.fnames_test
    # Get the ground truth from generator
    ground_truth = attr.test_generator.get_labels()
    # Get the predictions from the model using the generator
    predictions = attr.model.predict_generator(attr.test_generator, steps=attr.steps_test,
        verbose=1)
    predicted_classes = np.argmax(predictions, axis=1)
    errors = np.where(predicted_classes != ground_truth)[0]
    res = "No of errors = {}/{}".format(len(errors), len(attr.fnames_test))
    with open(attr.curr_basename + "-predicts.txt", "a") as f:
        f.write(res)
        print(res)
        f.close()
    # Reset test generator before summary predictions
    attr.test_generator.reset()
    write_summary_txt(attr, NETWORK_FORMAT, IMAGE_FORMAT, ['negative', 'positive'],
        time_callback, callbacks_list[2].stopped_epoch)
    K.clear_session()
copy_to_s3(attr)
```

A.19 VggMultimodalFineTuning.py

```
import numpy as np
from keras.applications import VGG19
from keras.layers import Input, concatenate
from keras.layers import Dropout, Flatten, Dense
from keras.models import Model
from keras.optimizers import SGD
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras import regularizers
from Summary import create_results_dir, get_base_name, plot_train_stats, write_summary_txt,
    copy_to_s3
from ExecutionAttributes import ExecutionAttribute
from TimeCallback import TimeCallback
from TrainingResume import save_execution_attributes
from keras.utils import plot_model
from MultimodalGenerator import MultimodalGenerator
import multiprocessing
import os
from keras import backend as K

# when running on p3.2xlarge
os.environ["HDF5_USE_FILE_LOCKING"]="FALSE"
```

```
# Summary Information
IMG_TYPE = "com_pre_proc/"
SUMMARY_PATH = "/mnt/data/results"
# SUMMARY_PATH="c:/temp/results"
# SUMMARY_PATH="/tmp/results"
NETWORK_FORMAT = "Multimodal"
IMAGE_FORMAT = "2D"
SUMMARY_BASEPATH = create_results_dir(SUMMARY_PATH, NETWORK_FORMAT, IMAGE_FORMAT)
INTERMEDIATE_FUSION = True
LATE_FUSION = False

# Execution Attributes
attr = ExecutionAttribute()
attr.architecture = 'vgg19'
attr.csv_path = 'csv/clinical_data.csv'
attr.s3_path = NETWORK_FORMAT + '/' + IMAGE_FORMAT
attr.numpy_path = '/mnt/data/image/2d/numpy/' + IMG_TYPE
# attr.numpy_path = '/home/amenegotto/dataset/2d/numpy/' + IMG_TYPE
attr.path = '/mnt/data/image/2d/' + IMG_TYPE
results_path = create_results_dir(SUMMARY_BASEPATH, 'fine-tuning', attr.architecture)
attr.summ_basename = get_base_name(results_path)
attr.set_dir_names()
attr.batch_size = 128
attr.epochs = 500
attr.img_width = 224
attr.img_height = 224
input_attributes_s = (20,)
# how many times to execute the training/validation/test cycle
CYCLES = 1
for i in range(0, CYCLES):
    #Load the VGG model
    vgg_conv = VGG19(weights='imagenet', include_top=False,
        input_shape=(attr.img_width, attr.img_height, 3))
    # Freeze the layers except the last 4 layers
    for layer in vgg_conv.layers[:-4]:
        layer.trainable = False
    # Check the trainable status of the individual layers
    for layer in vgg_conv.layers:
        print(layer, layer.trainable)
    flat = Flatten()(vgg_conv.output)

    if INTERMEDIATE_FUSION:
        attr.fusion = "Intermediate Fusion"
        attributes_input = Input(shape=input_attributes_s)
        concat = concatenate([flat, attributes_input])
        hidden1 = Dense(1024, activation='relu', kernel_initializer='he_normal',
            kernel_regularizer=regularizers.l2(0.0005))(concat)
        drop6 = Dropout(0.30)(hidden1)
        hidden2 = Dense(128, activation='relu')(drop6)
        drop7 = Dropout(0.20)(hidden2)
        hidden3 = Dense(64, activation='relu')(drop7)
        drop8 = Dropout(0.20)(hidden3)
        output = Dense(2, activation='softmax')(drop8)
        attr.model = Model(inputs=[vgg_conv.input, attributes_input], outputs=output)

    if LATE_FUSION:
        attr.fusion = "Late Fusion"
        hidden1 = Dense(1024, activation='relu')(flat)
        drop3 = Dropout(0.30)(hidden1)
        output_img = Dense(2, activation='softmax')(drop3)
```

```
    model_img = Model(inputs=vgg_conv.input, outputs=output_img)
    attributes_input = Input(shape=input_attributes_s)
    hidden3 = Dense(128, activation='relu')(attributes_input)
    drop6 = Dropout(0.20)(hidden3)
    hidden4 = Dense(64, activation='relu')(drop6)
    drop7 = Dropout(0.20)(hidden4)
    output_attributes = Dense(1, activation='sigmoid')(drop7)
    model_attr = Model(inputs=attributes_input, outputs=output_attributes)
    concat = concatenate([model_img.output, model_attr.output])
    hidden5 = Dense(8, activation='relu')(concat)
    output = Dense(2, activation='softmax')(hidden5)
    attr.model = Model(inputs=[model_img.input, model_attr.input], outputs=output)

plot_model(attr.model, to_file=attr.summ_basename + '-architecture.png')
attr.train_generator = MultimodalGenerator(
            npy_path = attr.numpy_path + '/train-categorical.npy',
            batch_size = attr.batch_size, height = attr.img_height,
            width = attr.img_width, channels = 3,
            classes = 2, should_shuffle = True,
            is_categorical = True, is_debug = False,
            width_shift = 0.2, height_shift = 0.2,
            rotation_angle = 15, shear_factor = 10, zoom_factor = 0.2)
attr.validation_generator = MultimodalGenerator(
            npy_path = attr.numpy_path + '/valid-categorical.npy',
            batch_size = attr.batch_size, height = attr.img_height,
            width = attr.img_width, channels = 3,
            classes = 2, should_shuffle = True,
            is_categorical = True, is_debug = False,
            width_shift = 0.2, height_shift = 0.2,
            rotation_angle = 15, shear_factor = 10, zoom_factor = 0.2)
attr.test_generator = MultimodalGenerator(
            npy_path = attr.numpy_path + '/test-categorical.npy',
            batch_size = 1, height = attr.img_height,
            width = attr.img_width, channels = 3,
            classes = 2, should_shuffle = False,
            is_categorical = True, is_debug = False)
print("[INFO] Calculating samples and steps...")
attr.calculate_samples_len()
attr.calculate_steps()
attr.increment_seq()
time_callback = TimeCallback()
callbacks = [time_callback, EarlyStopping(monitor='val_acc', patience=50, mode='max',
    restore_best_weights=True),
             ModelCheckpoint(attr.curr_basename + "-ckweights.h5", mode='max', verbose=1,
                monitor='val_acc', save_best_only=True)]
# Compile the model
attr.model.compile(loss='categorical_crossentropy', optimizer=SGD(lr=0.002), metrics=['acc'])
# Persist execution attributes for session resume
save_execution_attributes(attr, attr.summ_basename + '-execution-attributes.properties')
# Train the model
history = attr.model.fit_generator(
      attr.train_generator,
      steps_per_epoch=attr.steps_train,
      epochs=attr.epochs,
      validation_data=attr.validation_generator,
      validation_steps=attr.steps_valid,
      callbacks=callbacks,
      use_multiprocessing=True,
      workers=multiprocessing.cpu_count() - 1,
      verbose=1)
```

```
    # Save the model
    attr.model.save(attr.curr_basename + '-weights.h5')
    # Plot train stats
    plot_train_stats(history, attr.curr_basename + '-training_loss.png', attr.curr_basename +
        '-training_accuracy.png')
    # Get the filenames from the generator
    fnames = attr.fnames_test
    # Get the ground truth from generator
    ground_truth = attr.test_generator.get_labels()
    # Get the predictions from the model using the generator
    predictions = attr.model.predict_generator(attr.test_generator, steps=attr.steps_test,
        verbose=1)
    predicted_classes = np.argmax(predictions, axis=1)
    errors = np.where(predicted_classes != ground_truth)[0]
    res = "No of errors = {}/{}".format(len(errors), len(attr.fnames_test))
    with open(attr.curr_basename + "-predicts.txt", "a") as f:
        f.write(res)
        print(res)
        f.close()
    # Reset test generator before summary predictions
    attr.test_generator.reset()
    write_summary_txt(attr, NETWORK_FORMAT, IMAGE_FORMAT, ['negative', 'positive'],
        time_callback, callbacks[1].stopped_epoch)
    K.clear_session()
copy_to_s3(attr)
```

A.20 ExecutionAttribute.py

```
import datetime
from keras.preprocessing.image import ImageDataGenerator

class ExecutionAttribute:
    def __init__(self, summ_basename="", img_width=0, img_height=0, path="", epochs=0,
    batch_size=0, train_generator=None, validation_generator=None,
    test_generator=None, model=None, seq=0,
    architecture="",csv_path="", fusion="None",
    fnames_test=None, labels_test=None, npy_path=None, s3_path=None):
        self.img_width = img_width
        self.img_height = img_height
        self.path = path
        self.epochs = epochs
        self.model = model
        self.batch_size = batch_size
        self.train_generator = train_generator
        self.validation_generator = validation_generator
        self.test_generator = test_generator
        self.summ_basename = summ_basename
        self.train_data_dir = path + '/train'
        self.validation_data_dir = path + '/valid'
        self.test_data_dir = path + '/test'
        self.steps_train = 0
        self.steps_valid = 0
        self.steps_test = 0
        self.seq = seq
        self.architecture = architecture
        self.curr_basename = self.summ_basename + '-' + str(self.seq)
        self.init_timestamp = datetime.datetime.now()
        self.csv_path = csv_path
        self.fusion = fusion
```

```
        self.train_samples = 0
        self.valid_samples = 0
        self.test_samples = 0
        self.fnames_test = fnames_test
        self.labels_test = labels_test
        self.numpy_path = npy_path
        self.s3_path = s3_path

    def calculate_steps(self):
        if self.fusion != "None":
            self.steps_train = self.train_samples // self.batch_size
            self.steps_valid = self.valid_samples // self.batch_size
            self.steps_test = self.test_samples // 1
        else:
            self.steps_train = self.train_generator.n // self.train_generator.batch_size
            self.steps_valid = self.validation_generator.n // self.validation_generator.batch_size
            self.steps_test = self.test_generator.samples // self.test_generator.batch_size

    def set_dir_names(self):
        self.train_data_dir = self.path + '/train'
        self.validation_data_dir = self.path + '/valid'
        self.test_data_dir = self.path + '/test'

    def increment_seq(self):
        self.seq = self.seq + 1
        self.curr_basename = self.summ_basename + '-' + str(self.seq)

    def calculate_samples_len(self):
        gen = ImageDataGenerator()
        transient_train_generator = gen.flow_from_directory(self.train_data_dir, shuffle=False,
            batch_size=self.batch_size, seed=666)
        transient_validation_generator = gen.flow_from_directory(self.validation_data_dir,
        shuffle=False, batch_size=self.batch_size, seed=666)
        transient_test_generator = gen.flow_from_directory(self.test_data_dir, shuffle=False,
            batch_size=self.batch_size, seed=666)
        self.train_samples = len(transient_train_generator.filenames)
        self.valid_samples = len(transient_validation_generator.filenames)
        self.test_samples = len(transient_test_generator.filenames)
        self.labels_test = transient_test_generator.classes
        self.fnames_test = transient_test_generator.filenames
```

A.21 Summary.py

```
import matplotlib.pyplot as plt
from datetime import datetime
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix, cohen_kappa_score,
    roc_auc_score, roc_curve
from sklearn.metrics import precision_recall_fscore_support as score
from ExecutionAttributes import ExecutionAttribute
from TimeCallback import TimeCallback
from keras import backend as K
import os

def write_summary_txt(execattr : ExecutionAttribute, network_format, image_format, labels,
    time_callback: TimeCallback, stopped_epoch):
    with open(execattr.curr_basename + ".txt", "a") as f:
        f.write('EXECUTION SUMMARY\n')
        f.write('-----------------\n\n')
```

```
if execattr.architecture != "":
    f.write('Architecture: ' + execattr.architecture + '\n')
else:
    f.write('Architecture: From scratch\n')
f.write('Fusion: ' + execattr.fusion + '\n')
f.write('Execution Seq: ' + str(execattr.seq) + '\n')
f.write('Network Type: ' + network_format + '\n')
f.write('Image Format: ' + image_format + '\n')
f.write('Image Size: (' + str(execattr.img_width) + ',' + str(execattr.img_height) + ')\n')
f.write('Date: ' + datetime.now().strftime("%Y%m%d") + '\n')
f.write('Cycle Started at: ' + execattr.init_timestamp.strftime("%Y%m%d-%H%M%S") + '\n')
f.write('Cycle Finished at: ' + datetime.now().strftime("%Y%m%d-%H%M%S") + '\n')
if execattr.csv_path != "":
    f.write('CSV File: ' + execattr.csv_path + '\n')
if execattr.numpy_path is not None:
    f.write('Numpy Path: ' + execattr.numpy_path + '\n')
if execattr.fusion != "None":
    f.write('Train Samples: ' + str(execattr.train_samples) + '\n')
else:
    f.write('Train Data Path: ' + execattr.train_data_dir + '\n')
    f.write('Train Samples: ' + str(len(execattr.train_generator.filenames)) + '\n')
f.write('Train Steps: ' + str(execattr.steps_train) + '\n')
if execattr.fusion != "None":
    f.write('Validation Samples: ' + str(execattr.valid_samples) + '\n')
else:
    f.write('Validation Data Path: ' + execattr.validation_data_dir + '\n')
    f.write('Validation Samples: ' + str(len(execattr.validation_generator.filenames)) +
        '\n')
f.write('Validation Steps: ' + str(execattr.steps_valid) + '\n')
if execattr.fusion != "None":
    f.write('Test Samples: ' + str(execattr.test_samples) + '\n')
else:
    f.write('Test Data Path: ' + execattr.test_data_dir + '\n')
    f.write('Test Samples: ' + str(len(execattr.test_generator.filenames)) + '\n')
f.write('Test Steps: ' + str(execattr.steps_test) + '\n')
f.write('Epochs: ' + str(execattr.epochs) + '\n')
if stopped_epoch == 0:
    f.write('Stopped Epoch: ' + str(execattr.epochs) + '\n')
else:
    f.write('Stopped Epoch: ' + str(stopped_epoch) + '\n')
f.write('Batch Size: ' + str(execattr.batch_size) + '\n')
f.write('Learning Rate: ' + str(K.eval(execattr.model.optimizer.lr)) + '\n')
if execattr.fnames_test is None:
    filenames = execattr.test_generator.filenames
else:
    filenames = execattr.fnames_test
nb_samples = len(filenames)
print(filenames)
print(nb_samples)
f.write("Test Generator Filenames:\n")
print(filenames, file=f)
f.write("\nNumber of Test Samples:\n")
f.write(str(nb_samples) + "\n\n")
score_gen = execattr.model.evaluate_generator(generator=execattr.test_generator, workers=1,
    use_multiprocessing=False, steps=execattr.steps_test, verbose=1)
print(score)
print('Test Loss:', score_gen[0])
print('Test accuracy:', score_gen[1])
f.write('Test Loss:' + str(score_gen[0]) + '\n')
f.write('Test accuracy:' + str(score_gen[1]) + '\n\n')
```

```
print("Training Epochs Duration: ")
print(time_callback.times)
f.write("\n\nTraining Epochs Duration: " + "\n")
print(time_callback.times, file=f)
# Confusion Matrix and Classification Report
if execattr.test_generator is not None:
    execattr.test_generator.reset()
if execattr.architecture != "":
    Y_pred = execattr.model.predict_generator(execattr.test_generator, workers=1,
        use_multiprocessing=False, steps=execattr.steps_test, verbose=1)
    y_pred = np.argmax(Y_pred, axis=1)
    print(Y_pred)
    print(y_pred)
    if execattr.fusion != "None":
        classes = execattr.test_generator.get_labels()
    else:
        classes = execattr.test_generator.classes
    print(classes)
    f.write('Predicted Values: \n')
    print(Y_pred, file=f)
    f.write('\nRounded Values: \n')
    print(y_pred, file=f)
    f.write('\nClasses: \n')
    print(classes, file=f)
    mtx = confusion_matrix(classes, y_pred, labels=[1, 0])
    print('Confusion Matrix:')
    print(mtx)
    f.write('\n\nConfusion Matrix:\n')
    f.write('TP   FN\n')
    f.write('FP   TN\n')
    print(mtx, file=f)
    plt.imshow(mtx, cmap='binary', interpolation='None')
    plt.savefig(execattr.curr_basename + '-confusion_matrix.png')
    plt.clf()
    print('Classification Report')
    print('Classification Report:', file=f)
    if execattr.fusion != "None":
        target_names = labels
    else:
        target_names = list(execattr.test_generator.class_indices.keys())

    print(classification_report(classes, y_pred, target_names=target_names))
    print(classification_report(classes, y_pred, target_names=target_names), file=f)
    cohen_score = cohen_kappa_score(classes, y_pred)
    print("Kappa Score = " + str(cohen_score))
    f.write("Kappa Score = " + str(cohen_score))
    auc_score = roc_auc_score(classes, y_pred)
    print("ROC AUC Score = " + str(auc_score))
    f.write("\n\nROC AUC Score = " + str(auc_score))
    # plot the roc curve for the model
    fpr, tpr, thresholds = roc_curve(classes, y_pred)
    plt.plot([0, 1], [0, 1], linestyle='--')
    plt.plot(fpr, tpr, marker='.')
    plt.savefig(execattr.curr_basename + '-roc-curve.png')
    plt.clf()
else:
    Y_pred = execattr.model.predict_generator(execattr.test_generator, workers=1,
        use_multiprocessing=False, steps=execattr.steps_test, verbose=1)
    y_pred = np.rint(Y_pred)
    print(Y_pred)
```

```
            print(y_pred)
            if execattr.fusion != "None":
                classes = execattr.labels_test
            else:
                classes = execattr.test_generator.classes
            print(classes)
            f.write('Predicted Values: \n')
            print(Y_pred, file=f)
            f.write('\nRounded Values: \n')
            print(y_pred, file=f)
            f.write('\nClasses: \n')
            print(classes, file=f)
            mtx = confusion_matrix(classes, y_pred, labels=[1, 0])
            print('Confusion Matrix:')
            print(mtx)
            f.write('\n\nConfusion Matrix:\n')
            f.write('TP   FN\n')
            f.write('FP   TN\n')
            print(mtx, file=f)
            plt.imshow(mtx, cmap='binary', interpolation='None')
            plt.savefig(execattr.curr_basename + '-confusion_matrix.png')
            plt.clf()
            # print('Classification Report')
            target_names = labels
            print(classification_report(classes, y_pred, target_names=target_names))
            print(classification_report(classes, y_pred, target_names=target_names), file=f)
            cohen_score = cohen_kappa_score(classes, y_pred)
            print("Kappa Score = " + str(cohen_score))
            f.write("Kappa Score = " + str(cohen_score))
            auc_score = roc_auc_score(classes, y_pred)
            print("ROC AUC Score = " + str(auc_score))
            f.write("\n\nROC AUC Score = " + str(auc_score))
            # plot the roc curve for the model
            fpr, tpr, thresholds = roc_curve(classes, y_pred)
            plt.plot([0, 1], [0, 1], linestyle='--')
            plt.plot(fpr, tpr, marker='.')
            plt.savefig(execattr.curr_basename + '-roc-curve.png')
            plt.clf()
        f.close()
    write_csv_test_result(execattr, score_gen, y_pred, mtx, classes, stopped_epoch)

def plot_train_stats(history, filename_loss, filename_accuracy):
    # plot history
    loss_list = [s for s in history.history.keys() if 'loss' in s and 'val' not in s]
    val_loss_list = [s for s in history.history.keys() if 'loss' in s and 'val' in s]
    acc_list = [s for s in history.history.keys() if 'acc' in s and 'val' not in s]
    val_acc_list = [s for s in history.history.keys() if 'acc' in s and 'val' in s]
    if len(loss_list) == 0:
        print('Loss is missing in history')
    pepochs = range(1, len(history.history[loss_list[0]]) + 1)
    ## Loss
    plt.figure(1)
    for l in loss_list:
        plt.plot(pepochs, history.history[l], 'b',
                 label='Training loss (' + str(str(format(history.history[l][-1], '.5f')) + ')'))
    for l in val_loss_list:
        plt.plot(pepochs, history.history[l], 'g',
                 label='Validation loss (' + str(str(format(history.history[l][-1], '.5f')) + ')'))
    plt.title('Loss')
    plt.xlabel('Epochs')
```

```
    plt.ylabel('Loss')
    plt.legend()
    plt.savefig(filename_loss)
    plt.clf()
    ## Accuracy
    plt.figure(2)
    for l in acc_list:
        plt.plot(pepochs, history.history[l], 'b',
                 label='Training accuracy (' + str(format(history.history[l][-1], '.5f')) + ')')
    for l in val_acc_list:
        plt.plot(pepochs, history.history[l], 'g',
                 label='Validation accuracy (' + str(format(history.history[l][-1], '.5f')) + ')')
    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.savefig(filename_accuracy)
    plt.clf()

def write_csv_test_result(execattr: ExecutionAttribute, score_gen, y_pred, mtx, classes,
    stopped_epoch):
    with open(execattr.summ_basename + ".csv", "a") as csv:
        if execattr.seq == 1:
            csv.write('Test Loss, Test Accuracy, TP, FP, TN, FN, Precision, Recall, F-Score,
                Support, Stopped Epoch\n')
        precision, recall, fscore, support = score(classes, y_pred, average='macro')
        if stopped_epoch == 0:
            stopped_epoch = execattr.epochs
        csv.write(str(score_gen[0]) + ',' + str(score_gen[1]) + ',' + str(mtx[0, 0]) + ',' +
            str(mtx[1, 0]) + ',' + str(mtx[1, 1]) + ',' + str(mtx[0, 1]) + ',' +
            str(precision) + ',' + str(recall) + ',' + str(fscore) + ',' +
                str(support) + ',' + str(stopped_epoch) + '\n')
        csv.close()

def create_results_dir(basepath, network_format, image_format):
    if not os.path.exists(basepath):
        os.makedirs(basepath)
    if not os.path.exists(basepath + '/' + network_format):
        os.makedirs(basepath + '/' + network_format)
    if not os.path.exists(basepath + '/' + network_format + '/' + image_format):
        os.makedirs(basepath + '/' + network_format + '/' + image_format)
    return basepath + '/' + network_format + '/' + image_format

def get_base_name(basepath):
    return basepath + '/' + datetime.now().strftime("%Y%m%d-%H%M%S")

def save_model(execattr: ExecutionAttribute):
    execattr.model.save(execattr.curr_basename + "-model.h5")

def copy_to_s3(execattr: ExecutionAttribute):
    src_dir = execattr.summ_basename.split('/')
    os.system("aws s3 cp "+execattr.summ_basename.replace(src_dir[len(src_dir)-1],"") +
        " s3://pyliver-logs/"+ execattr.s3_path + " --recursive --exclude \"*\"
            --include \"*" + src_dir[len(src_dir)-1] + "*\"")
```

A.22 MultimodalGenerator.py

```
from keras.utils import Sequence
from Datasets import get_image
```

```
import numpy as np
from ImageAugmentation import rnd_shift, rnd_rotation, rnd_shear, rnd_zoom
import random

class MultimodalGenerator(Sequence):

    def __init__(self, npy_path, batch_size, height, width, channels, classes,
        should_shuffle=True, is_categorical = False, is_debug=False, width_shift=None,
        height_shift=None, rotation_angle=None, shear_factor=None, zoom_factor=None):
        self.debug = is_debug

        if self.debug:
            print("On generator init")

        self.dataset = np.load(npy_path)

        if self.debug:
            print("Loaded numpy array " + npy_path)
            print("dataset shape = " + str(self.dataset.shape))

        self.batch_size = batch_size
        self.height = height
        self.width = width
        self.n_channels = channels
        self.n_classes = classes
        self.shuffle = should_shuffle
        self.width_shift = width_shift
        self.height_shift = height_shift
        self.rotation_angle = rotation_angle
        self.shear_factor = shear_factor
        self.zoom_factor = zoom_factor
        self.is_categorical = is_categorical
        self.batch_index = 0
        self.on_epoch_end()

    def __len__(self):
        if self.debug:
            print("__len__")
        return int(np.floor(len(self.dataset) // self.batch_size))

    def on_epoch_end(self):
        if self.debug:
            print("__on_epoch_end__")
        self.indexes = np.arange(len(self.dataset))
        if self.shuffle:
            np.random.shuffle(self.indexes)

    def __getitem__(self, index):
        if self.debug:
            print("__get_item__")
        batch_array = self.dataset[range(index*self.batch_size, (index+1)*self.batch_size)]
        return self.__data_generation(batch_array)

    def __data_generation(self, batch_array):
        if self.debug:
            print("__data_generation__")
            print("len(batch_array) = " + str(len(batch_array)))

        imgs_path = batch_array[:, 0]
        batch_img = []
```

```
        batch_attributes = batch_array[:, range(1, 21)]
        if self.is_categorical:
            batch_labels = batch_array[:, range(21,23)]
        else:
            batch_labels = batch_array[:, 21]

        for img_path in imgs_path:
            img = get_image(img_path, self.width, self.height)

            tr = random.randint(0, 3)

            if tr == 0 and (self.width_shift is not None or self.height_shift is not None):
                img = rnd_shift(img, self.width_shift, self.height_shift)
            elif tr == 1 and self.rotation_angle is not None:
                img = rnd_rotation(img, self.rotation_angle)
            elif tr == 2 and self.shear_factor is not None:
                img = rnd_shear(img, self.shear_factor)
            elif tr == 3 and (self.zoom_factor is not None):
                img = rnd_zoom(img, self.zoom_factor)

            batch_img.append(img)

        X = [(np.array(batch_img, dtype="float") / 255.0), batch_attributes]
        return X, batch_labels

    def reset(self):
        self.batch_index = 0

    def get_labels(self):
        if self.is_categorical:
           return np.argmax(self.dataset[:, range(21,23)], axis=1)
        else:
           return self.dataset[:, 21]
```