SERGIO ARIEL GONZALES FUENTES

MELHORAMENTO DE UMA REDE NEURAL CONVOLUCIONAL PARA DETECÇÃO DE CÂNCER DE MAMA USANDO TPU

São Paulo
2022

SERGIO ARIEL GONZALES FUENTES

MELHORAMENTO DE UMA REDE NEURAL CONVOLUCIONAL PARA DETECÇÃO DE CÂNCER DE MAMA USANDO TPU

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro da Computação.

São Paulo
2022

SERGIO ARIEL GONZALES FUENTES

MELHORAMENTO DE UMA REDE NEURAL CONVOLUCIONAL PARA DETECÇÃO DE CÂNCER DE MAMA USANDO TPU

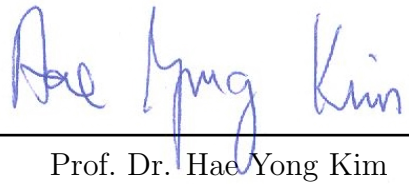
Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro da Computação.

Área de Concentração:
Engenharia de Computação

Orientador:
Prof. Dr. Hae Yong Kim

Co-orientador:
Prof. Daniel Gustavo Pellacani Petrini

São Paulo
2022



Prof. Dr. Hae Yong Kim

Prof. Daniel Gustavo Pellacani Petrini

Catalogação-na-publicação

Fuentes, Sergio
MELHORAMENTO DE UMA REDE NEURAL CONVOLUCIONAL PARA DETECÇÃO DE CÂNCER DE MAMA USANDO TPU / S. Fuentes, H. Kim, D. Petrini -- São Paulo, 2022.
60 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Redes Convolucionais 2.Câncer de Mama 3.Tensor Processing Units (TPUs) I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t. III.Kim, Hae IV.Petrini, Daniel

AGRADECIMENTOS

Gostaria de agradecer primeiramente ao meu orientador, o Professor Doutor Hae Yong Kim e ao meu co-orientador, Professor Daniel Petrini, pelos conselhos fornecidos durante o desenvolvimento do projeto.

Agradeço também a minha família e meus amigos que me deram apoio ao longo do ano durante a realização deste trabalho de formatura.

E, finalmente, gostaria de agradecer a todos os professores da Universidade de São Paulo que foram compartilhando seu conhecimento e permitindo que eu me forme um engenheiro de computação.

RESUMO

O câncer mais comum em mulheres no mundo é o câncer de mama. O sucesso no tratamento desse mal depende muito de um diagnóstico precoce. Para isso, a mamografia é o principal exame utilizado para reconhecer a patologia em seus estágios iniciais. Atualmente essas imagens radiológicas são interpretadas por radiologistas e, visando auxiliá-los nesse trabalho, sistemas CAD (*Computer-Aided Detection and Diagnosis*) vem sendo desenvolvidos. Dentre eles, existem alguns que utilizam técnicas de aprendizagem profunda e redes neurais convolucionais que vem obtendo um bom desempenho inclusive comparando com especialistas humanos.

Para o processo de treinamento em geral são utilizadas GPUs (*Graphical Processing Units*), que possuem um grande poder de processamento quando é possível paralelizar os cálculos que serão realizados, como é caso de multiplicação de matrizes, operação muito utilizada no treinamento de uma rede neural. Entretanto, existem dispositivos que foram projetados especificamente para esse tipo de tarefa, contando com mais recursos que as GPUs, eles são chamados TPUs (*Tensor Processing Units*) e são ASICs (*application-specific integrated circuits*) desenvolvidos pelo Google visando acelerar os processamento de aprendizagem de máquina. Este trabalho utilizará como base um modelo de rede que foi projetada inicialmente para rodar em GPUs, visando adaptá-la de modo a aproveitar os recursos extras fornecidos pelas TPUs, para em seguida analisar os resultados obtidos e verificar se realmente existe uma melhora na acurácia do sistema e, em caso positivo, quão melhor é o processo comparado ao caso utilizando uma GPU.

Palavras-Chave – *Tensor Processing Units* (TPUs), Aprendizagem Profunda, Redes Convolucionais, Mamografias, Câncer de Mama, Classificação.

ABSTRACT

The most common cancer in women worldwide is breast cancer. The success in the treatment of this disease depends on an early diagnosis. For this, mammography is the main exam used to identify this pathology in its early stages. Currently, these radiological images are interpreted by radiologists and, in order to assist them in this task, CAD (Computer-Aided Detection and Diagnosis) systems have been developed. Among them, there are some that use deep learning techniques and convolutional neural networks that have been performing well, even compared to human experts.

For the training process, GPUs (Graphical Processing Units) are usually used. However, there are devices that were designed specifically for this type of task, with more resources than GPUs, they are called TPUs (Tensor Processing Units) and are ASICs (application-specific integrated circuits) developed by Google to accelerate Machine processing. Learning. This work will use a network model that was initially designed to run on GPUs as a basis, aiming to adapt it in order to take advantage of the extra resources provided by TPUs, to later analyze the results obtained and verify if there really is an improvement in the accuracy of the system and, if so, how much better they are compared to the case of using a GPU.

Keywords – Tensor Processing Units (TPUs), Deep Learning, Convolutional Networks, Mamograms, Breast Cancer, Classification.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O câncer de mama é o câncer que mais afeta mulheres no mundo. Apenas em 2020 foram pouco mais de 2,2 milhões de novos casos, além de 684 mil mortes decorrentes desse mal. Uma redução no índice de mortalidade dessa doença é observada em países com maior IDH, o que pode ser parcialmente explicado pelos programas de prevenção criados, permitindo um diagnóstico precoce da doença [1].

Por consequência do alto número de casos, essa enfermidade recebe cada vez mais atenção por parte das entidades públicas e privadas de maneira geral, as quais incentivam a realização de mamografias periodicamente para permitir a detecção desse mal no estágio mais precoce possível. Com isso, a quantidade desses exames vem aumentando cada dia mais [2].

1.1 Motivação

Para auxiliar os médicos com o aumento de demanda de exames, sistemas CAD (*Computer-Aided Detection and Diagnosis*) passaram a ser desenvolvidos. Destacam-se os baseados em redes neurais convolucionais profundas devido ao desempenho obtido equivalente ao de especialistas humanos [3].

Entre esses CADs que utilizam redes CNN (*Convolutional Neural Network*), existe uma técnica interessante de treino "ponta a ponta" que consiste em duas aprendizagens de transferência. A primeira inicializa o modelo com os pesos obtidos do treinamento com imagens naturais para criar um "classificador de *patch*" que reconhece lesões em pequenas regiões da mamografia. Já a segunda usa os pesos do classificador de *patch* para criar o "classificador de imagem inteira de vista única" [4].

A partir disso, em outro trabalho [5], foi proposto fazer uma terceira aprendizagem de transferência para obter um "classificador de imagem inteira de duas vistas" para usar as duas incidências da mamografia: bilateral craniocaudal (CC) e mediolateral oblíqua

(MLO), além de substituir os modelos utilizados por versões mais modernas.

Importante notar que os trabalhos mencionados [4] [5] apresentam limitações devido ao *hardware* utilizado, em especial considerando a falta de memória para utilizar as mamografias em seu tamanho original. Com isso, não foi explorado todo o potencial dessas arquiteturas.

1.2 Objetivo

Visando identificar de maneira mais precisa a capacidade desse tipo de arquitetura que utiliza múltiplas vistas para auxiliar no processo de classificação, o objetivo do projeto é avaliar se existe uma melhora no diagnóstico de câncer de mama ao utilizar as mamografias com uma maior resolução.

Para isso, pretende-se utilizar a rede criada por Petrini *et al* e descrita no artigo *Breast Cancer Diagnosis in Two-View Mammography Using End-to-End Trained EfficientNet-Based Convolutional Network* [5]. Ela será considerada a rede base deste trabalho e será adaptada para funcionar em uma TPU.

O objetivo é tirar proveito dos recursos adicionais disponíveis nesse dispositivo, sendo o principal deles a maior quantidade de memória, o que permite inserir imagens de maior resolução como entrada do sistema.

1.3 Justificativa

A ideia é consequência do grande tamanho ocupado pelos exames de mamografia, que em geral têm dimensões da ordem de 2400 x 3070 pixels ou até maiores. Em razão disso, um processo de redimensionamento é necessário para que as imagens consigam ser utilizadas para o treinamento.

Quando é preciso reduzir muito a imagem, temos uma perda na qualidade dessa, o que afeta a capacidade de diagnóstico. Essa piora nos dados da entrada causa a perda de detalhes, como a de texturas por exemplo. O câncer de mama do tipo microcalcificação é caracterizado justamente como um textura no exame, logo, uma piora na definição torna a detecção desse tipo de câncer mais difícil para o sistema.

Um aumento na quantidade de memória disponível no sistema, pode diminuir a escala do redimensionamento necessário e, consequentemente, a perda de informação. Essa

mudança pode ser feita trocando o uso de GPUs por TPUs, pois aquelas têm tipicamente 16GB de memória disponível, enquanto estas possuem pelo menos 64GB, permitindo no mínimo uma entrada com o dobro das dimensões lineares e, consequentemente, espera-se uma melhora nos resultados de detecção.

Um sistema capaz de detectar automaticamente a probabilidade de um paciente ter câncer em mamografias é de grande interesse público, considerando o grande número de pessoas afetadas por essa doença, principalmente em países como o Brasil, onde a mortalidade por conta desse câncer ainda cresce e torna-se cada vez mais necessário uma maior realização de mamografias para detectar a doença o mais cedo possível, melhorando as probabilidades de um tratamento efetivo [2].

1.4 Organização do Trabalho

O trabalho visa identificar se existe uma melhora na classificação de uma mamografia através de uma CNN que foi treinada com maior resolução em uma TPU, de modo a documentar todo o processo de desenvolvimento e os resultados obtidos. Com isso, pretende-se apresentar um trabalho reprodutível e capaz de ser continuado a partir do obtido ao final da execução deste.

Com base nisso, este documento inicialmente apresenta os conceitos para o entendimento do projeto, em seguida ocorre uma descrição de como se pretende desenvolvê-lo. Passa-se então a apresentar os requisitos e também as tecnologias utilizadas. Finalmente, retrata-se como foi o processo de desenvolvimento, a avaliação dos resultados e as conclusões obtidas.

Assim, o capítulo 2 apresenta os aspectos conceituais, e está dividido em três partes. Os aspectos biológicos, relacionados ao câncer de mama e o exame chamado de mamografia. Os aspectos computacionais, os quais incluem definições relacionadas ao âmbito de redes neurais convolucionais, além de relacionados a TPUs. E finalmente uma seção para apresentar brevemente os aspectos mais específicos da rede base para o trabalho.

O capítulo 3 descreve a metodologia para o desenvolvimento do trabalho. Ele especifica mais a fundo cada uma das partes da rede a ser criada, incluindo o processo de replicação até a montagem do sistema final. Além de especificar a escolha do *dataset* a ser utilizado e o processamento que será feito nele. Enquanto o capítulo 4 resume os requisitos do trabalho.

O capítulo 5 apresenta as tecnologias utilizadas, além de descrever o processo de de-

senvolvimento detalhadamente, permitindo a reprodutibilidade dos experimentos, finalmente, é apresentada a avaliação dos resultados, a qual inclui os resultados intermediários até o sistema final.

O capítulo 6, o último, expõe as conclusões do trabalho como um todo. Incluindo um resumo dos resultados obtidos, além de explicitar as contribuições do trabalho e as possíveis perspectivas de continuidade deste.

2 ASPECTOS CONCEITUAIS

Nesta seção discorre-se sobre os aspectos conceituais necessários para compreender o trabalho. Em primeiro lugar, apresentam-se informações relacionados à enfermidade em que se pretende realizar os experimentos, além de informações quanto ao exame utilizado.

A seguir, são explicados os conceitos base para a utilização de redes neurais convolucionais e também os relacionados ao uso do dispositivo TPU. Finalmente, descreve-se a rede que servirá de base para o projeto, explicitando os aspectos característicos dela.

2.1 Aspectos Biológicos

Nesta parte do trabalho são apresentados os aspectos relacionados à doença a ser estudada, neste caso tem-se o câncer de mama, e em seguida um exame muito utilizado para tentar diagnosticá-la, a mamografia.

2.1.1 Câncer de Mama

Trata-se do câncer mais comum entre mulheres e existem vários tipos dele. Alguns são bastante agressivos e capazes de se espalhar para outros órgãos. Uma rápida detecção visa evitar esse espalhamento, além de permitir o início do tratamento, aumentando as chances de um melhor prognóstico.

O tecido cancerígeno pode estar presente tanto na forma de calcificações, pequenos depósitos de minérios no tecido mamário, quanto na forma de massas. Ambas maneiras podem ser detectadas por exames como a mamografia, entretanto, a presença de massas ou calcificações não são apenas geradas por tecido cancerígeno, podendo ter outras razões para sua formação [6] [7].

Para exemplificar os dois casos, tem-se imagens de partes de duas mamografias. A figura 1 mostra como se observa pelo exame a presença de massa e a figura 2 apresenta um caso de microcalcificação.

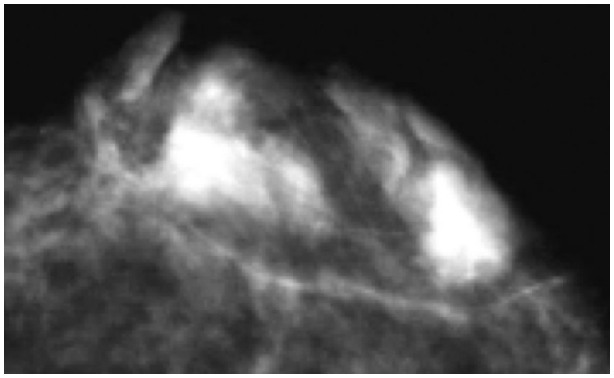


Figura 1: Massa em mamografia
Fonte: [7]

Figura 2: Microcalcificação em mamografia
Fonte: [7]

Vale notar que as imagens acima são apenas exemplos, e que os aspectos visuais de cada tipo variam bastante em razão de diversos fatores como densidade, forma ou ainda tamanho de cada uma dessas estruturas

2.1.2 Mamografia

Exame muito utilizado para checagem de possíveis estruturas que indiquem a presença de câncer na mama. Trata-se de uma exame que emite raios-X pelo tecido mamário e obtém uma imagem da mama. Geralmente retiram-se duas incidências da mama cada vez que o exame é realizado, a MLO (mediolateral-oblíqua) e a CC (craniocaudal), caso necessário, podem-se retirar vistas complementares [7] [8].

A utilização de pelo menos duas incidências é decorrente da necessidade de verificar se alguma alteração aparece em mais de uma visão, confirmando ou não os achados da outra vista, além de também permitir a visualização de mais quadrantes da mama,

possibilitando encontrar novas alterações não presentes na outra vista [8].

Importante notar que existem dois tipos do exame, a SFM (*screen-film mammography*), na qual a imagem é passada diretamente para um papel filme, sendo que para a utilização em computadores é necessário escanear o exame, o que pode reduzir a qualidade da imagem. O outro seria o FFDM (*full-field digital mammography*), que utiliza um detector que transforma o raio-X em uma sinal elétrico para que a imagem produzida seja digital [5].

2.2 Aspectos Computacionais

Agora são apresentados os conceitos computacionais necessários para a compreensão do sistema de uma rede neural convolucional a ser desenvolvido visando classificar uma mamografia quanto a presença ou não de câncer e também conceitos relacionados ao uso da TPU.

2.2.1 Aprendizado de Máquina e Aprendizado Profundo

De acordo com [9], aprendizado de máquina ou *machine learning* é um conjunto de métodos estatísticos que visa resolver diferentes tarefas, como regressão, classificação, detecção, entre outros. Para isso, são estimadas funções complexas, através de diferentes métodos de otimização.

Quanto a questão de aprendizado, é possível defini-lo como a capacidade de um computador melhorar sua performance, medida por um métrica pré-definida, para a realização de uma determinada tarefa com base em uma aumento na experiência, que pode ser entendido como aquela ganha ao passar por um banco de dados [9].

Existem dois tipos de aprendizado principais: o supervisionado e o não-supervisionado. O primeiro apresenta um conjunto de dados de entrada e saídas esperadas como experiência, visando que o modelo consiga prever uma saída com base em uma outra entrada que possa ser fornecida. O último apenas utiliza características do banco de dados, com o objetivo de aprender propriedades presentes no *dataset* [9].

Já o aprendizado profundo ou *deep learning*, uma sub-área de aprendizado de máquina, consiste em métodos nos quais aprende-se conceitos complexos a partir de outros mais simples, isso é feito através do uso de diferentes estruturas capazes de aprender organizadas hierarquicamente e em múltiplas camadas [9].

2.2.2 Redes Neurais

As redes neurais, uma das técnicas de aprendizado profundo, são sistemas que visam representar o funcionamento dos neurônios humanos para que se obtenham modelos capazes de aprender a resolver diferentes problemas. Isso ocorre através de estruturas simples que modelam o neurônio através de funções que recebem uma entrada e devolvem uma saída, a qual é utilizada para definir um erro, sendo este utilizado para atualizar os parâmetros internos dessa estrutura [10].

As redes neurais podem ser formadas por uma ou mais camadas, cada uma delas consiste por um conjunto de neurônios que realizam uma determinada função dependendo do seu tipo. Para evitar que apenas seja possível resolver problemas lineares, são utilizadas funções de ativação, explicadas em 2.2.2.1. Já para evitar o problema de *overfitting* principalmente em redes grandes e profundas utiliza-se o método de regularização chamado *dropout*, que para redes neurais corresponde a desativar de maneira aleatória uma taxa de neurônios a cada passo da aprendizagem.

Existem diversos parâmetros que podem ser configurados para o processo de aprendizagem em redes neurais. Dentre as existentes, pode-se destacar a função de perda, o otimizador, a taxa de aprendizagem, o tamanho do *batch* e as épocas. Cada uma delas é apresentada nas seções a seguir.

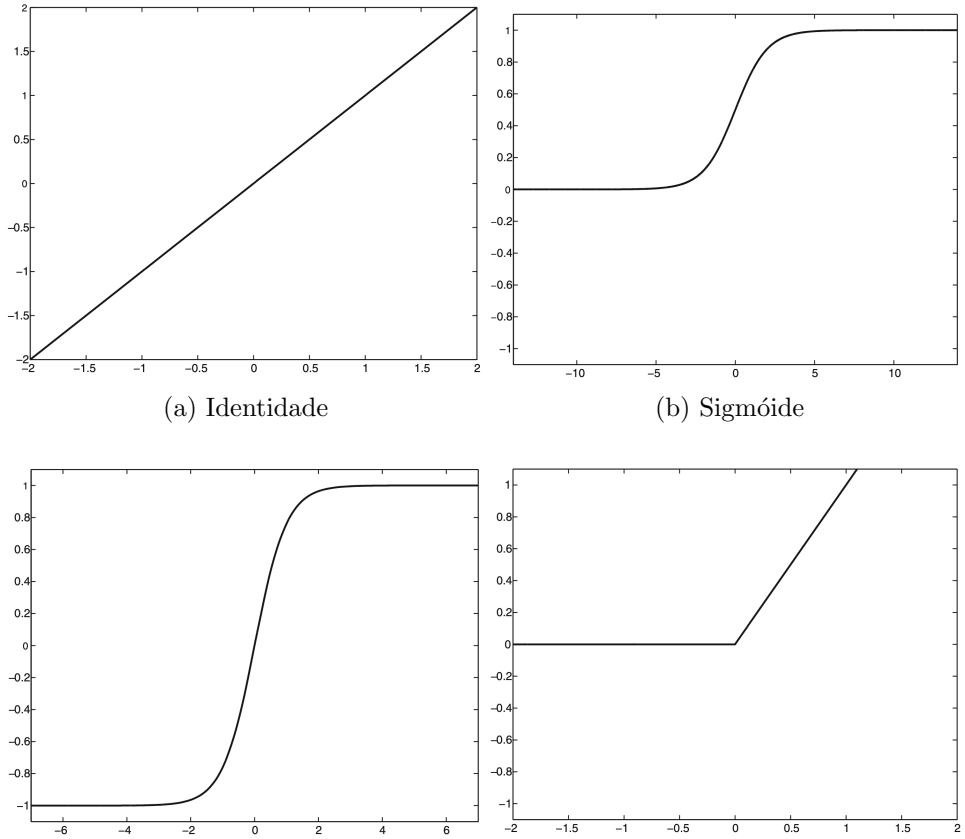
2.2.2.1 Função de Ativação

Utilizada na saída dos neurônios, trata-se de uma função em geral não-linear para permitir que o sistema modelo funções mais complexas. Existem diversas funções utilizadas, e em geral busca-se aquelas com uma derivada facilmente calculável, pois esse valor é utilizado durante o aprendizado. Um exemplo seria a função *swish*, que apresenta a equação presente em 2.1 e sua derivada em 2.2.

$$swish(x) = x \times sigmóide(x) \tag{2.1}$$

$$swish'(x) = swish(x) + sigmóide(x) \times (1 - swish(x)) \tag{2.2}$$

Outros exemplos de funções de ativação podem ser vistos na figura 3.



(a) Identidade (b) Sigmóide

(c) Tangente Hiperbólica (d) ReLU

Figura 3: Exemplos de funções de ativação
Fonte: [10]

2.2.2.2 Tamanho do *Batch*

Hiperparâmetro de uma rede neural que indica quantas amostras de treinos devem ser passadas antes da atualização dos parâmetros internos da rede neural, aumenta os requisitos de memória quando seu valor é aumentado, e torna-se computacionalmente ineficiente se seu valor for muito pequeno [10].

2.2.2.3 Épocas

Hiperparâmetro de uma rede que indica quantas vezes o modelo passará pelo *dataset* de treino durante o treinamento. A partir de certo valor, a depender do modelo e dos dados disponíveis, as melhorias na métrica de avaliação deixam de ser relevantes [10].

2.2.2.4 Função Perda

Função ou algoritmo responsável por comparar as saídas de uma rede com o valor esperado. Quanto mais discrepantes forem esses valores, pior é considerado o desempenho do sistema.

Considerando isso, deseja-se reduzir o máximo possível o valor da função perda, também conhecida como função erro. Obtém-se assim um problema de otimização, onde os parâmetros a serem ajustados são aqueles internos do sistema e a função perda é a que indica o resultado da comparação [10].

Importante notar que existe uma grande variedade dessas funções, e que elas estão ligadas à tarefa que se pretende resolver. Para um problema de classificação, pode-se utilizar a entropia cruzada por exemplo, representada na equação 2.3, onde y_i é a saída esperada, o_i é a saída obtida e k corresponde ao número de classes do problema.

$$L = -\sum_{i=1}^{k} y_i log(o_i) \tag{2.3}$$

A partir dessas funções, a cada passo do treinamento, que pode envolver uma ou mais amostras de dados é calculado o valor da função erro e em seguida visa-se minimizá-lo seguindo algum algoritmo otimizador, o qual será descrito a seguir.

2.2.2.5 Otimizador

Função ou algoritmo responsável por alterar os pesos durante o treinamento de um modelo, visando reduzir a função de perda. Tem esse nome por visar resolver o problema de otimização de obter o menor valor possível da função de perda.

A atualização ocorre após o cálculo do erro de uma rede, durante a retropropagação desse erro nas camadas anteriores da rede. Um exemplo de otimizador é o Adam, que além de utilizar a descida de gradiente também utiliza o conceito de momento para diminuir a chance do modelo deixar de aprender por estar em um mínimo local. [10]

2.2.2.6 Taxa de Aprendizagem

Também conhecida como *learning rate*, representa em que proporção os atributos da rede serão atualizados. Caso seu valor seja muito pequeno, a convergência para encontrar o mínimo valor da função perda é muito lenta, e se o seu valor for muito alto podem

ocorrer problemas como a divergência do valor ótimo da função perda ou ainda apenas encontrar mínimos locais.

Importante notar que podem-se adotar *learning rates* tanto constantes quanto variáveis conforme o estado do treinamento de um certo modelo [11].

Um exemplo de taxa de aprendizagem variável é do tipo *cyclic cosine* com *warmup*. Nele, o *warmup* indica que nas primeiras épocas o valor do *learning rate* vai aumentando linearmente com a época. A partir de um determinado momento, segue-se o *cyclic cosine*, onde os valores variam seguindo uma função cossenoidal da época, na qual pode-se variar a amplitude, o período e o valor base [12].

2.2.3 Redes Neurais Convolucionais

Também conhecidas como CNNs, são redes neurais que têm como diferencial a utilização de camadas de convolução, as quais realizam a operação convolução, que consiste em uma operação onde se obtém a saída a da soma da multiplicação elemento-a-elemento da matriz de entrada com uma matriz chamada de filtro ou *kernel* [13] conforme ilustrado na figura 4.

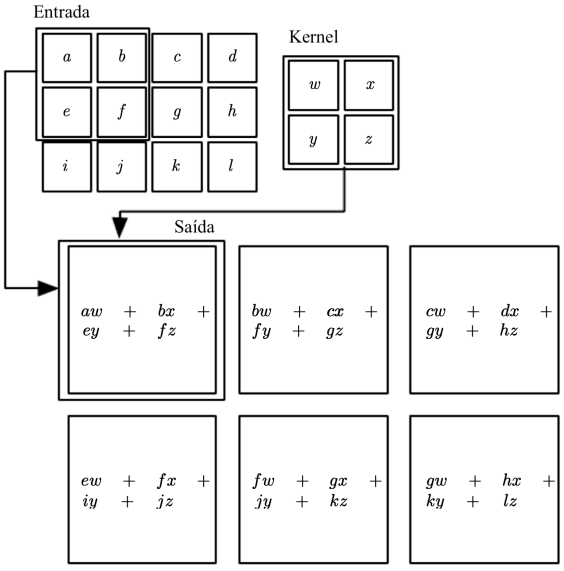


Figura 4: Operação de convolução
Fonte: [9]

A camada de convolução apresenta parâmetros configuráveis, dentre os quais destacam-se o tamanho dos filtros, o número de filtros, o *stride* e o *padding*. O tamanho dos filtros corresponde ao formato da matriz usada com *kernel*, por exemplo, na imagem 4 esse valor é 2 x 2.

O *stride* corresponde ao deslocamento realizado pelo filtro antes de realizar uma convolução, com o objetivo de reduzir a entrada para a camada posterior, como observa-se na figura 5.

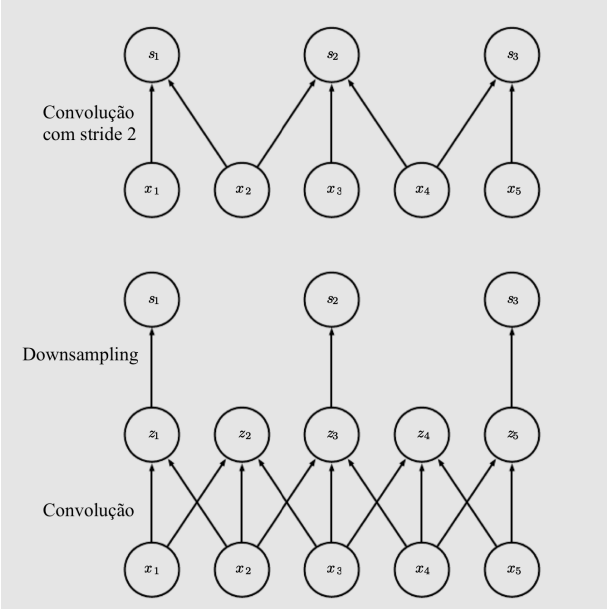


Figura 5: Operação de convolução com *stride* 2
Fonte: [9]

Já o *padding* pode ser utilizado para evitar que ocorra uma redução na entrada da camada seguinte e consiste em preencher os valores fora do domínio com um determinado valor, o qual pode ser fixou ou seguir uma regra como repetir os valores presentes nas bordas. Na figura 6 tem-se um *padding* com valor fixo 0.

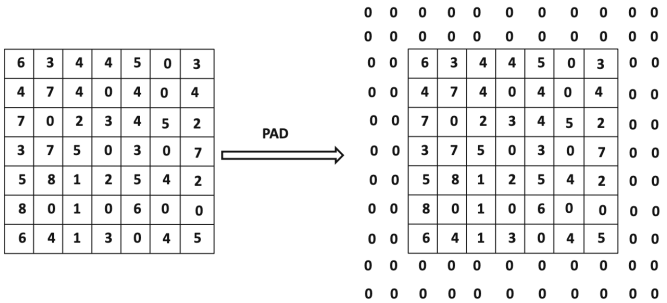


Figura 6: *Padding* com valor fixo 0
Fonte: [10]

Além da camada de convolução, é muito recorrente a presença de uma de *pooling*, a qual realiza o *downsampling* para uma camada posterior. Existem diversas maneiras de realizar o *pooling*, podendo-se obter a média de uma sub-área da imagem ou ainda o maior valor nela presente, esta última operação é chamada *max-pooling* e está representada na figura 7. Frequentemente, também utiliza-se uma função de ativação entre as camadas.

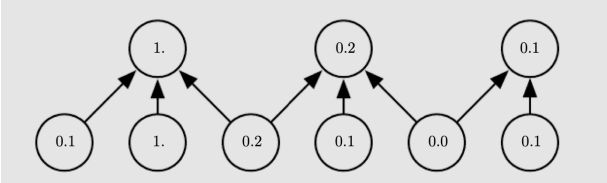


Figura 7: Exemplo de *max-pooling*
Fonte: [9]

As redes do tipo CNN foram criadas para entradas onde existe uma forte dependência espacial entre certas regiões da entrada, esta geralmente é representada em uma estrutura de grade ou matriz. Em razão disso, as redes convolucionais são muito utilizadas em imagens, mas também podem ser utilizadas para dados espaço-temporais [10].

Muito utilizadas no campo de visão computacional, nesses casos os filtros têm o objetivo de extrair características das imagens, sendo que pode-se organizá-las em diversas camadas, para ir extraindo características cada vez mais complexas.

2.2.4 Aumento de Dados

Também conhecido como *data augmentation*, essa técnica é muito comum na construção de redes neurais artificiais, visando aumentar a quantidade de dados de entrada e melhorar a generalidade do modelo, evitando assim problemas de sobreajuste [10].

Vale notar que podem-se utilizar diversas técnicas em imagens para esse processo, desde ajustes como rotação das imagens de entrada até alterações nas cores, por exemplo. No entanto, é preciso tomar cuidado com as transformações a serem realizadas para evitar que entradas sejam alteradas a ponto de serem classificadas com uma classe diferente da inicial.

2.2.5 TPU

A TPU (*Tensor Processing Unit*) é um ASIC (*Application-Specific Integrated Circuit*) criado pelo *Google* com o objetivo de acelerar as operações realizadas em redes neurais. Foi construído visando uma rápida operação em matrizes, sem a necessidade de acesso a memória durante uma multiplicação de matrizes depois que os dados foram carregados [14].

As TPUs foram criadas para realizar operações em matrizes, logo precisam estar conectadas a um *host* com uma CPU. Ele é responsável por realizar as demais tarefas, como carregar os dados do *Cloud Storage*, pré-processar os dados ou ainda enviá-los para

a TPU. Na figura 8 tem-se um esquema dessa conexão.

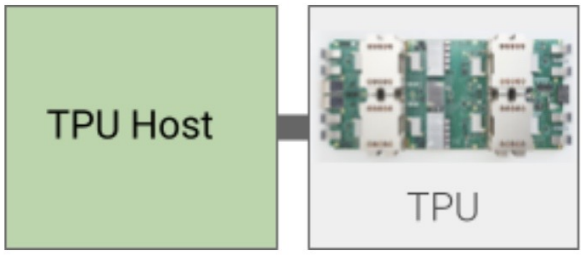


Figura 8: TPU associada a *host*
Fonte: [14]

Em razão disso, é preciso ter cuidado com os dados que serão colocados na TPU, pois pode existir um gargalo caso seja necessário constantemente enviar dados entre o *host* e a TPU [15].

Existem três versões de TPUs disponíveis para uso geral, a v2, a v3 e a v4. A quantidade de memória disponível para elas é variável, podendo ir desde 64GiB até 4TiB para a v2. Como melhoras relevantes entre as duas primeiras versões destacam-se que a v3 tem dois chips multiplicadores de matriz por *core*, além de ter o dobro de *High-Band Memory* [14]. A seguir, nas figuras 9 e 10 estão representadas as arquiteturas da TPUv2 e da TPUv3, respectivamente:

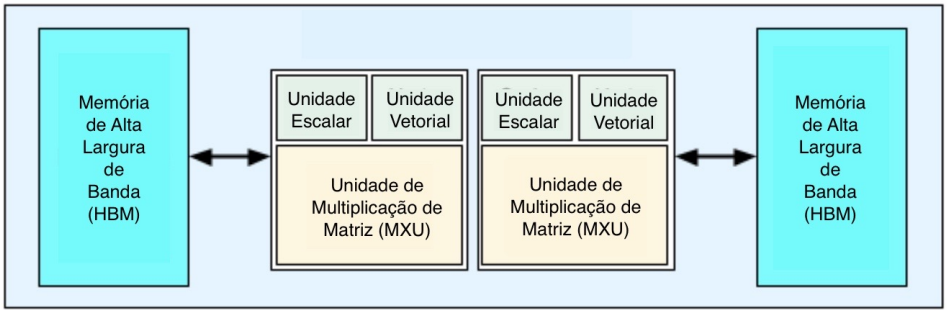


Figura 9: Arquitetura TPUv2
Fonte: [14]

Quanto a versão v4, ela ainda não está disponível para o público sem custo adicional e apenas tornou-se acessível no período final do projeto, logo não foi considerada a utilização dessa.

2.2.6 *TFRecords*

Formato para guardar uma sequência de binários, utilizado para a serialização de dados estruturados. Por se tratar de um arquivo binário, tem uma menor utilização de

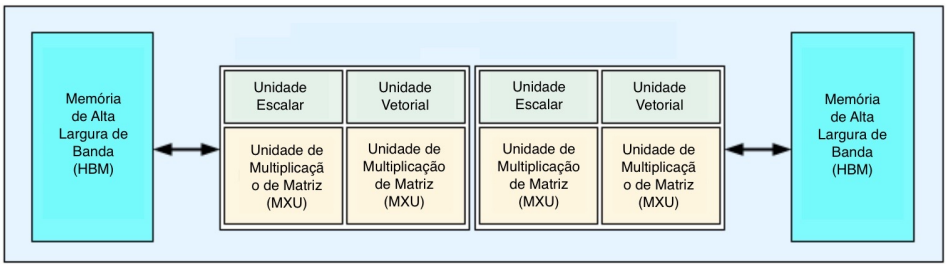


Figura 10: Arquitetura TPUv3
Fonte: [14]

espaço em disco e também permite um maior aproveitamento de arquiteturas que tiram proveito de um paralelismo de dados, como as TPUs, por exemplo.

A serialização dos dados permite juntar várias amostras de teste em um único arquivo, permitindo criar arquivos de entrada para a TPU de tamanhos que evitem um excesso de comunicação entre o CPU Host e a TPU em si, evitando assim o gargalo mencionado na seção sobre a TPU. Em geral, busca-se utilizar entradas na ordem de dezenas de megabytes [16].

Também por condensar a entrada e a saída em um único arquivo, permite-se que o processamento seja dividido em diferentes cores, tirando proveito do paralelismo da TPU.

2.2.7 *Transfer Learning*

É uma técnica de *machine learning* em que o modelo desenvolvido anteriormente para uma tarefa é utilizado como ponto de partida para um segundo modelo. Muito utilizado para problemas de visão computacional devido à quantidade de recursos necessários para o treinamento de um modelo do zero.

Dentre suas vantagens tem-se aproveitar um modelo que já treinou com uma quantidade bem maior de dados disponíveis e também uma melhora no tempo de treinamento já que não é preciso realizar todo o treinamento desde o início [10].

2.2.8 Treino Ponta-a-ponta

Também conhecido como *end-to-end training*, refere-se a um treinamento em que o modelo aprende diretamente dos dados de entrada a chegar à saída, sem a presença de etapas intermediárias como o treinamento isolado de cada componente intermediário [17].

2.3 Rede Neural Utilizada

Como mencionado a anteriormente, a rede neural que será adaptada é a desenvolvida por Petrini *et al* e descrita no artigo *Breast Cancer Diagnosis in Two-View Mammography Using End-to-End Trained EfficientNet-Based Convolutional Network* [5], ou seja, será a rede base para o projeto.

Quando completa, a rede tem como entrada duas mamografias redimensionadas no tamanho 1158 x 896 da vista CC e MLO e como saída se obtém dois possíveis resultados, um indicando a existência de câncer e outro não.

Ela é composta a partir de três processos de *transfer learning*, o primeiro é um classificador de *patch* que é inicializado pelos pesos do *ImageNet* [18].

Em seguida esse classificador é utilizado para inicializar os pesos para o classificador de vista única, o qual é treinado de ponta a ponta.

Finalmente, o classificador de vista única é utilizado tanto para a vista MLO quanto CC, treinando o sistema de ponta a ponta, e gerando o classificador de imagem inteira duas vistas.

A seguir serão descritos os principais componentes que a formam, já que eles deverão também estar presentes na versão final do trabalho e também a rede neural convolucional utilizada como base para o desenvolvimento dos demais componentes, a *EfficientNet*.

2.3.1 *EfficientNet*

Trata-se de um conjunto de modelos de redes neurais convolucionais profundas que variam do B0 até o B7, onde entre cada uma ocorre o escalonamento tanto em profundidade, largura e resolução. A ideia é realizar esse aumento de maneira ótima, evitando aumentar demasiadamente o número de parâmetros a serem treinados.

Utiliza como bloco base o *MBConv*, que apresenta várias camadas incluindo as convolucionais, até outras como a de *batch noramlization* ou ainda de ativação. O objetivo dele é ser eficiente no tamanho ocupado sem prejudicar a acurácia obtida [19].

2.3.2 Classificador de *Patch*

Classifica pequenas regiões da imagem chamadas de *patches*. Estas correspondem a áreas que foram extraídas de ROIs (*regions of interest*), que são locais na imagem onde

existe uma estrutura que caracteriza a presença de câncer. Os *patches* também podem ser áreas que não representam a presença dessa doença.

Considerando essas características, o *patch* pode pertencer a cinco categorias: fundo, calcificação maligna, calcificação benigna, massa benigna e massa maligna. O modelo é construído com base em uma rede pré-treinada com as imagens naturais do *ImageNet*.

Como entrada tem-se um *patch* de tamanho fixo 224 x 224 e como saída obtém-se uma das cinco categorias que foram apresentadas. Importante notar que essa classificação em cinco classes é obtida ao adicionar três camadas no final do modelo, uma de *Average Pooling*, outra de *Flatten* e finalmente uma Densa. Elas são retiradas ao utilizar o *patch classifier* no classificador de imagem inteira de vista única.

2.3.3 Classificador de Imagem Inteira de Vista Única

Utiliza os pesos do classificador de *patch* para sua inicialização. A entrada é a imagem inteira de uma das vistas da mamografia para determinar se existe ou não a presença de câncer no paciente.

Usa o *patch classifier* já treinado como primeiro bloco, retirando as camadas finais mencionadas anteriormente. Em seguida adicionam-se blocos do tipo *MBConv*, para a seguir adicionar camadas de *Average Pooling, Flatten* e Densa, definindo assim dois possíveis resultados para a saída.

Com a arquitetura descrita anteriormente, utiliza-se como entrada uma imagem de uma das vistas (CC ou MLO) no tamanho 1158 x 896 e se obtém como saída dois possíveis resultados, um indicando a existência de câncer e outro não.

2.3.4 Classificador de Imagem Inteira de Duas Vistas

Usa o classificador de imagem inteira de vista única para a vista CC e para a vista MLO para inicializar este novo classificador. Em seguida realiza-se um treino completo utilizando as imagens das duas vistas para chegar ao resultado final.

Do classificador de imagem inteira de vista única, retiram-se as últimas camadas para cada vista, concatenando os mapas resultantes de cada classificador para então adicionar dois blocos do tipo MBConv. Para finalizar adicionam-se camadas de *Average Pooling, Flatten* e Densa, definindo dois possíveis resultados para a saída.

Com o sistema completo, a entrada consiste de imagens de uma mamografia em vista

CC e vista MLO, cada uma na resolução 1158 x 896 e como saída a classificação do par de imagens como benigna ou maligna para câncer de mama.

3 METODOLOGIA DO TRABALHO

Nesta seção são descritas as etapas em que o trabalho será desenvolvido, sendo que elas não necessariamente devem ocorrer em ordem sequencial. O objetivo é melhorar um sistema já existente. Com isso, o foco é inicialmente reconstruir o sistema com as ferramentas a serem utilizadas, tentando obter os resultados já conhecidos da rede base [5], para em seguida testar as alterações que visam melhorar o desempenho desse.

3.1 Definição do banco de dados

Para o trabalho pretende-se utilizar o mesmo *dataset* que foi usado em [5], pois ele apresenta a possibilidade de verificar se a reprodução ocorreu de maneira correta. Dentre as características relevantes desse banco de dados, está que ele possui rótulo verdadeiro provado por biópsia, além de apresentar as regiões indicativas de câncer segmentadas e classificadas em diferentes tipos.

Caso esteja disponível outro banco de dados com essas características e com mamografias do tipo FFSM (*Full-Field Digital Mammography*) ele poderia ser utilizado realizando-se primeiro o teste da rede em GPU e depois em TPU com maior resolução, mas até o momento da escrita da monografia não foi encontrado um banco com essas características.

Com isso, os testes serão realizados no dataset público *Curated Breast Image Subset of Digital Database for Screening Mammography* (CBIS-DDSM) [20], que é formado por um conjunto de arquivos no formato *Digital Imaging and Communications in Medicine* (DICOM).

É possível utilizar diferentes divisões de treino e teste, entretanto, será utilizada a divisão padrão já estabelecida, a qual usa 20% das imagens para teste e o restante para treinamento.

Um ponto importante a notar é que o CBIS-DDSM utiliza imagens SFM (*screen-film mammography*), obtidas pela digitalização de cada exame através do escaneamento, o que

pode afetar a qualidade dos resultados obtidos, inclusive não mostrando uma melhora considerável ao passar a utilizar TPUs.

Inicialmente será utilizada uma versão já preparada por Daniel Petrini quando ele desenvolveu a rede base [5] em *PyTorch*, onde as imagens estão no formato *Portable Network Graphics* (PNG) e na resolução a ser utilizada na rede para utilizar em uma GPU com 16 GB, ou seja, 1152x896. Uma vez funcionando com essa versão, utilizar-se-á o *dataset* com os dados em DICOM para assim ir aumentando a resolução das imagens de entrada.

Esse banco de dados apresenta 3570 regiões de interesse segmentadas, as quais são utilizadas para a geração dos *patches*. Na tabela 1 está presente o número de imagens de treino e testes no CBIS-DDSM para uma única vista e também o número de imagens que formam um par de vistas CC e MLO.

	Treino		Teste	
Categoria	Benigno	Maligno	Benigno	Maligno
Mamografias com uma vista	1347	1111	381	264
Mamografias com duas vistas	1180	968	324	222

Tabela 1: Número de imagens de vista única e com pares CC e MLO do CBIS-DDSM
Fonte: do autor

3.2 Classificador de *Patch*

Também chamado de *patch classifier*, é baseado no *EfficientNet* [19], em particular o modelo *EfficientNet-B0*, pré-treinado com os pesos do *ImageNet*. Ele utiliza pequenas regiões de 224x224 pixels da mamografia inteira para classificar em cinco categorias as regiões que serviram de entrada. As classes são: fundo, calcificação benigna, calcificação maligna, massa benigna e massa maligna.

Do modelo *EfficientNet-B0* retiram-se as últimas camadas, para adicionar uma versão própria dessas, formadas por uma camada de *Average Pooling*, outra de *Flatten* e finalmente uma Densa.

Para o modelo será utilizado o otimizador Adam com um taxa de aprendizagem com *warm-up* e do tipo *cyclic cosine* [12] por 30 épocas e com um delta da amplitude do *learning rate* de 2×10^{-4}, juntamente com um *delay* de *warm-up* de 4 épocas.

Importante notar que esses parâmetros seguem os utilizados no artigo da rede base [5].

O objetivo é inicialmente obter os resultados obtidos nele e espera-se uma acurácia em torno de 0,7554. Em seguida, se trabalhará para melhorar esse resultado utilizando *patches* de maior resolução.

3.3 Classificador de Imagem Inteira de Vista Única

O classificador de imagem inteira de vista única é criado a partir do classificador de *patch* já treinado, retirando-se as últimas camadas desse e adicionando dois blocos do tipo *MBConv* [19]. Em seguida, são colocadas novas camadas de *Average Pooling, Flatten* e uma Densa para permitir que o modelo tenha como saída apenas dois rótulos, indicando se existe ou não a presença de câncer.

Para o treinamento será utilizado o otimizador Adam com uma taxa de aprendizagem com *warm-up* e do tipo *cyclic cosine* e inicializado em 10^{-5} por 50 épocas e com um delta da amplitude do *learning rate* de 2×10^{-5} e um *batch size* de 4 juntamente com um *delay* de *warm-up* de 4 épocas.

Novamente esses parâmetros foram utilizados na rede base [5]. Com esses parâmetros espera-se aproximadamente replicar o resultado de 0,8033 de AUC (*Area Under Curve*).

Vale notar que neste classificador de imagem de vista única foi um local onde identificou-se uma restrição devido à quantidade de memória da GPU, em especial no *batch size* e na resolução da imagem de entrada, com esse recurso mais disponível ao usar uma TPU, espera-se aumentar esses valores.

3.4 Classificador de Imagem Inteira de Duas Vistas

O classificador de imagem inteira de duas vistas é criado usando dois classificadores de imagem inteira de vista única, um para receber a vista CC e outro a vista MLO. Nos dois casos são retirados os blocos*MBConv* e as últimas camadas.

Realiza-se então a concatenação da saída de cada uma das vistas e adicionam-se novamente blocos *MBConv* para, finalmente, adicionar as últimas etapas de *Average Pooling, Flatten* e uma Densa responsáveis pela classificação do par de imagens em maligno ou benigno para câncer de mama.

Para o treinamento inicialmente será testado o otimizador Adam com um *learning rate* com *warm-up* e do tipo *cyclic cosine* e inicializado em 10^{-5} por 100 épocas e com um

delta da amplitude da taxa de aprendizagem de $2x10^{-5}$ e um batch size de 6 juntamente com um *delay* de *warm-up* de 4 épocas. Importante notar que esses hiperparâmetros podem sofrer ajustes em razão dos resultados obtidos.

Novamente esses parâmetros foram utilizados na rede base [5] e espera-se obter um AUC em torno de 0,8418.

3.5 Transformação em *TFRecords*

Para a utilização de TPUs é recomendado a passagem dos dados para o formato *TFRecords*, pois este é serializável e permite um melhor uso do paralelismo das TPUs, logo é preciso fazer a transformação das imagens para esse formato. Além da imagem, é preciso adicionar o rótulo correspondente, para que assim os dados possam passar paralelamente pelos núcleos da TPU.

Outro ponto importante é colocar os dados em um *Google Cloud Bucket*, esse processo pode ser feito de duas maneiras. A primeira é manualmente, criando um *bucket* de maneira externa, permitindo uma melhor manipulação desse. A segunda forma é utilizando um ambiente de desenvolvimento que gerencie esse processo, com isso existem limitações impostas que devem ser respeitadas.

A princípio a segunda opção será utilizada para evitar custos adicionais por armazenamento. Caso a menor flexibilidade de uso torne-se um empecilho, pode-se facilmente criar um *bucket* manualmente e depois configurar a conexão com os diferentes modelos a serem criados.

3.6 Aumento da Resolução das Imagens

Utilizando o CBIS-DDSM inicialmente com os dados em DICOM, será feito o aumento gradativo das imagens de vista inteira, começando na resolução 1152 x 896 até serem observados problemas por falta de memória. A princípio espera-se conseguir utilizar imagens de aproximadamente o dobro de dimensões lineares, ou seja, 2304 x 1792.

Importante relevar que as imagens do classificador de *patch* devem ser redimensionadas na mesma proporção, ou seja, se a resolução linear for duplicada, o tamanho de *patch* a ser utilizado também deve ser duplicado.

Uma vez encontrada a resolução máxima a ser treinada no CBIS-DDSM e ajustados

os hiperparâmetros da rede para funcionar na TPU, será possível obter os resultados para o CBIS-DDSM.

3.7 Obtenção dos Resultados

Conforme explicado anteriormente, inicialmente será reconstruída a rede base usando as tecnologias necessárias para rodar em uma TPU, com isso será utilizada uma GPU semelhante a utilizada no artigo [5] da rede base para obter os resultados iniciais para o banco de dados escolhido.

Uma vez obtidos os dados, as alterações na resolução da imagem de entrada serão ajustadas e se obterá os novos resultados para cada *dataset*, desta vez rodando em uma TPU. Completando assim os dados necessários para realizar a comparação quanto ao uso de diferentes resoluções de imagem de entrada.

4 ESPECIFICAÇÃO DE REQUISITOS

Nesta seção serão apresentados alguns dos requisitos para a realização deste trabalho.

4.1 Métricas para Avaliação de Resultados

Conforme os outros artigos do assunto [4] [5] as métricas para a avaliação do resultado serão a acurácia do modelo e a AUC (*Area Under Curve*), ambas muito utilizadas em problemas de classificação [14].

Esta última torna-se importante para um modelo classificador binário, neste caso a classificação consiste em identificar a presença ou não de câncer, pois indica quanto um modelo é capaz de distinguir entre classes.

Além disso, a AUC é obtida a partir *receiver operating characteristic curve* (ROC curve). Esta última utiliza a taxa de falsos positivos e verdadeiros positivos, duas outras medidas consideradas muito importantes para se obter um diagnóstico correto na área médica.

Conforme utilizado em [5], calculou-se o desvio padrão da AUC através da fórmula proposta por Hanley e McNeil [21].

Além dessas métricas, também pode-se verificar a diferença de tempo com o uso de uma TPU. Uma métrica que pode ser utilizada é o tempo médio que o modelo demora para completar uma época, ou seja, para apresentar todos os exemplos do banco de dados de treinamento uma única vez.

Outra análise interessante é capaz de surgir a partir do uso da matriz de confusão, que consiste em uma matriz onde cada elemento a_{ij} corresponde ao número de predições para a classe j dado que a classe correta era i .

Com esses dados é possível obter diferentes métricas, como por exemplo as taxas de falsos positivos e negativos de cada classe. Permitindo assim analisar se em alguma das

classes o classificador apresenta maior dificuldade ou ainda entre quais ocorre confusão.

4.2 Acesso à TPUs

Como já mencionado, a TPU é um ASIC projetado pelo Google e não está disponível para a compra do dispositivo em si, sendo apenas possível utilizar os seus serviços por meio de ambientes de desenvolvimento na nuvem.

As principais opções para isso são o *Google Colab*, o *Kaggle* e o *Google Cloud Platform*, todas essas pertencentes ao *Google*. Os primeiros dois possuem acesso gratuito à TPU com restrições de tempo. Entretanto, o *Google Colab* possui acesso apenas a TPUv2 gratuitamente, já o *Kaggle* tem TPUv3.

O *Google Cloud Platform* pode ser utilizado junto com o *Google Colab*, obtendo assim acesso à TPUv3, com isso entram custos para a utilização dessa, que variam conforme a região em que a TPU está sendo utilizada, começando a partir de $2.40/h [22].

Existe também a TPUv4, acessada exclusivamente pelo GCP, mas durante a redação desta monografia, apenas é possível utilizar um *pod* [22]. Isso acarretaria em um maior custo sem necessariamente fornecer poder computacional que seria realmente utilizado.

Considerando os custos e o acesso à mais nova versão da TPU que pode ser utilizada isoladamente disponível para o público em geral, optou-se por utilizar o *Kaggle*, apesar da restrição de tempo de uso de 20 horas semanais.

4.3 Processamento para uso de TPUs

Para aproveitar o paralelismo das TPUs é recomendado transformar o banco de dados em *TFRecords*. Com isso, cada arquivo desses não necessariamente precisa representar uma única imagem dos bancos originais. Evitando assim uma transferência de dados constante na rede, um dos possíveis gargalos ao utilizar TPUs, já que elas estão sempre conectadas a uma CPU *host* que precisa transmitir os dados a serem utilizados ao ASIC.

De uma maneira geral, esse excesso de comunicação descrito anteriormente deve ser evitado. Consequentemente, adicionar processos que sejam apenas possíveis de serem realizados em CPU não é recomendável. Isso causaria a necessidade de uma constante troca de informações entre a TPU e a CPU, potencialmente surgindo um gargalo de memória [15] mencionado.

Outro ponto importante é definir como a carga de trabalho deve ser dividida entre os diferentes *cores* da TPU. Os principais *frameworks* de aprendizagem de máquina já apresentas diferentes implementações para realizar esse trabalho de maneira simples para o usuário.

5 DESENVOLVIMENTO DO TRABALHO

Nesta seção são apresentadas as tecnologias utilizadas durante o desenvolvimento do trabalho, além de descrever o processo de desenvolvimento em si.

As tecnologias apresentadas principalmente são aquelas em que será feito o desenvolvimento, e não as que foram utilizadas originalmente na rede base [5].

5.1 Tecnologias Utilizadas

A rede base de [5] foi criada utilizando-se a linguagem *Python* junto com o módulo *PyTorch*. Para o uso nas TPUs neste trabalho, utilizou-se também *Python* em conjunto com a biblioteca *Tensorflow* e a API *Keras*. Para a codificação optou-se por utilizar o ambiente online do *Kaggle*, responsável também por gerenciar os *buckets* do *Google Cloud Storage*.

5.1.1 *Python*

A utilização de *Python* ocorreu devido à simplicidade da linguagem na escrita, além de possuir uma extensa documentação e também diversas bibliotecas disponíveis para as mais variadas finalidades, dentre elas para uso em *machine learning*, onde é uma das principais linguagens da área.

5.1.2 *Tensorflow*

A biblioteca *Tensorflow* é a biblioteca *open source* de inteligência artificial desenvolvida pelo *Google*, sendo uma das bibliotecas dessa categoria mais utilizadas no mundo. A grande disponibilidade de documentação e de ferramentas disponíveis nela, além da integração nativa com as TPUs, que também são produzidas pelo *Google*, foram os motivos pela escolha dessa [23].

5.1.3 *Keras*

A API *Keras* fornece uma interface de alto nível visando facilitar a criação de modelos de aprendizagem profunda. Seu foco em aumento de produtividade, ampla documentação e também integração bem estabelecida com o *Tensorflow* justificam a escolha do uso desta API [24].

5.1.4 *PyTorch*

O módulo *PyTorch* tem uma função equivalente ao *Tensorflow*, sendo também *open-source*. Sua agilidade para o desenvolvimento das redes é um de seus pontos fortes, além de sua ampla documentação. Para este trabalho foi necessário conhecer um pouco sobre esse *framework*, pois a rede utilizada como base foi construída utilizando-o [25].

5.1.5 *Kaggle*

A utilização do interpretador online do *Kaggle* [26] ocorreu devido à disponibilidade de TPU v3-8 (modelo v3 com 128GiB de memória) de maneira gratuita, além de já terem disponíveis os módulos para a utilização tanto do *Tensorflow*, quanto do *Keras* e também do *Google Cloud Storage*.

5.1.6 *Google Cloud Storage*

Por fim, tem-se a utilização dos *Google Cloud Storage*, o serviço de armazenamento de dados do *Google*. Seu uso decorre da necessidade de ter os dados presentes nos servidores do *Google* para que possam ser passados de maneira eficiente para as TPUs, as quais também estão disponíveis nesses servidores.

5.2 Projeto e Implementação

Nesta seção será descrito como foi o andamento do projeto, levando em consideração as partes do projeto conforme foi dividido na metodologia de trabalho.

O primeiro passo da implementação é a reprodução da rede de Petrini *et al* [5] no *framework Tensorflow*. Também são apresentadas as alterações necessárias para a utilização em TPU e o processamento necessário para utilizar os exames em diferentes resoluções.

Nas próximas subseções são apresentados de maneira detalhada os cuidados necessários durante a implementação de cada um dos três componentes da rede, além do processamento para aumento de dados e demais tratamento dos dados.

5.2.1 Redimensionamento das Imagens

A princípio utilizou-se a versão do CBIS-DDSM que foi disponibilizado por Petrini e que já tinha os *patches* e as imagens inteiras em formato PNG para a resolução usada originalmente, ou seja, 224 x 224 para o *patch* e 1152 x 896 para a imagem inteira. Com isso foi possível testar a reprodução em *Tensorflow* com exatamente os mesmos dados que a versão original [5].

5.2.1.1 Exames Selecionados para Redimensionamento

Entretanto, como já estavam redimensionadas, foi preciso buscar as imagens do CBIS-DDSM em outra fonte. Duas alternativas foram consideradas. A primeira era baixar diretamente o CBIS-DDSM e segunda era utilizar a versão disponibilizada no *Kaggle* [27].

A segunda opção foi escolhida por três motivos. O primeiro é que trata-se do ambiente utilizado para o desenvolvimento, logo um *dataset* já nele presente evita a necessidade de realizar o *upload* de um novo banco de dados. O segundo é a presença das imagens já em resolução maior e também com as máscaras indicando as regiões de interesse. E finalmente, através dos arquivos CSVs (*comma-separated values*) foi possível verificar que todos as imagens na versão de Petrini também estão na versão do *Kaggle*.

Visando uma menor variação nos dados utilizados para os testes, optou-se por apenas utilizar os mesmos exames que estavam presentes no CBIS-DDSM utilizado originalmente em [5].

5.2.1.2 Geração dos *Patches* Redimensionados

Importante notar que os dados para todos os três componentes devem ser redimensionados. Sendo que os *patches* devem manter a proporção em relação a imagem completa, logo se a resolução linear dobra para o exame completo o mesmo deve ocorrer para o *patch*.

Para isso, primeiro é preciso redimensionar as imagens completas e as máscaras com as regiões de interesse para a resolução desejada, e em seguida gerar os *patches* a partir disso.

A geração dos *patches* segue o especificado em [5] e consiste em obter o centro de massa da região de interesse, e variando esse ponto em 10% em todas as direções obter dez *patches* que contém essa região. Por fim, mais dez *patches* são escolhidos de modo a que não se sobreponham com a região de interesse.

Um exemplo de como são gerados os *patches* está presente na figura 11. A imagem a esquerda corresponde a mamografia completa e a direita está a máscara que indica onde existe uma lesão indicativa de câncer. Os quadrados com moldura amarela correspondem aos *patches* que seriam categorizados como de fundo e os obtidos a partir do quadrado com bordas brancas receberiam a classificação da lesão que contém.

O processo descrito gera um desbalanceamento nas classes e optou-se por não o corrigir, para manter a reprodução mais fiel e também por obterem-se resultados satisfatórios dessa maneira.

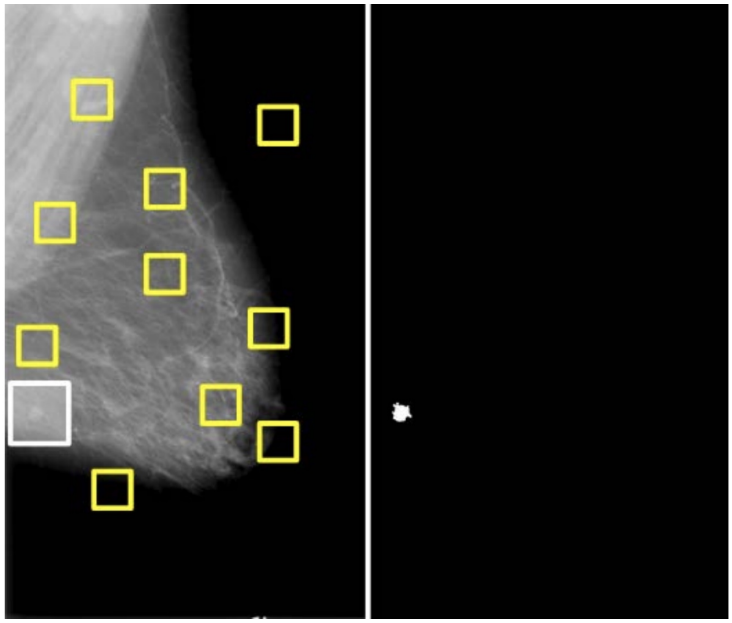


Figura 11: Exemplo de extração de *patches*
Fonte: [5]

Para a montagem dos dados de duas vistas, são selecionados os pares de vista CC e vista MLO, e se um deles apresentar o rótulo indicativo de presença de câncer de mama, o par recebe essa classificação. Caso contrário, o par de imagens é considerado como benigno.

5.2.2 Transformação em *TFRecords*

Tanto para os exames completos quanto para os *patches* o processo de passagem de para *TFRecords* é realizado da mesma maneira. Importante mencionar que ao utilizar duas vistas, é preciso serializar as duas imagens e depois adicionar o rótulo correspondente.

Para o processo de passagem de *TFRecords* seguiu-se as recomendações fornecidas pelo *Google* [28], criando-se arquivos com tamanho na ordem de dezenas de MBs e um número múltiplo de oito de arquivos gerados, facilitando assim a distribuição entre os diferentes núcleos da TPU.

De forma simplificada, obtinha-se uma lista com o nome dos arquivos e as respectivas classes, com a ordem dos arquivos aleatorizada. A aleatorização é importante mencionar, pois se todos os exemplos de uma mesma classe estão condensados em um único arquivo, o desempenho do modelo diminui.

Para cada caso, a imagem era lida, serializada e adicionada juntamente com a classe correspondente a uma estrutura de dados criada para a serialização, para que em seguida ocorra a escrita desses dados em um dos arquivos *TFRecord*, dessa forma, cada um deles possui diversas amostras. O conhecimento da estrutura criada foi posteriormente utilizada para a desserialização durante o processo de treinamento dos diferentes componentes.

5.2.3 Aumento de Dados

A princípio, o objetivo foi reproduzir completamente as técnicas de *data augmentation* usada em [5]. Com isso, foram buscadas rotinas equivalentes às utilizadas em *PyTorch* e que funcionam em tensores, o formato que é utilizado em *Tensorflow*. Uma vez com a reprodução feita, realizou-se uma comparação entre os resultados gerados e as mudanças geradas nas imagens são praticamente idênticas.

Entretanto, o processo de aumento de dados obtido inicialmente não foi necessariamente o utilizado, pois apesar da similaridade nos resultados obtidos, o impacto de sua utilização em *Tensorflow* e *PyTorch* é diferente.

Através de experimentos verificou-se que os resultados tornam-se mais similares ao trabalho original utilizando apenas *flips* horizontais e verticais aleatoriamente nas imagens. Logo, utilizou-se apenas essa técnica para todos os casos.

5.2.4 Classificador de *Patch*

Utilizou-se a implementação de *EfficientNet-B0* presente no *Keras* [29], a qual já permitia obter essa rede pré-treinada no *ImageNet*. Importante notar que existem diferenças com a versão utilizada originalmente em *PyTorch*, uma delas é que a entrada esperada na versão de *Keras* está no intervalo de 0 a 255, enquanto que em *PyTorch* está no intervalo de 0 a 1.

Outra diferença encontrada é que foi preciso adicionar manualmente uma camada de *Dropout*, pois ao utilizar a *EfficientNet* sem as camadas finais essa camada também era retirada. A não utilização do *dropout* tornava o classificador enviesado a apenas uma das classes, tornando o desempenho do modelo bastante inferior.

Por fim, utilizou-se também a função de ativação *softmax* ao invés da linear que foi originalmente utilizada. Essa mudança decorre do fato da implementação do *EfficientNet* do *Keras* [29] ter sido originalmente pré-treinada com essa função de ativação, resultando em um melhor resultado ao utilizá-la.

Na figura 12 está presente a arquitetura do componente:

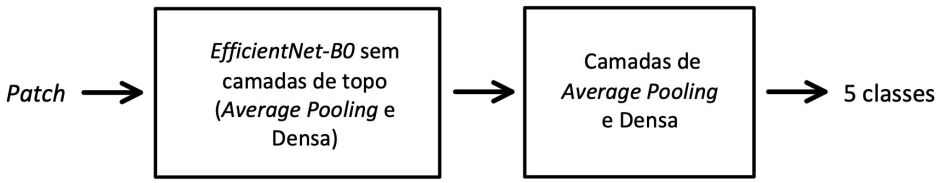


Figura 12: Arquitetura do classificador de *patches*
Fonte: do autor

Conforme mencionado na seção 3.2, a taxa de aprendizagem foi do tipo *cyclic cosine* com *warmup*, e durante 30 épocas obteve os valores apresentados na figura 13:

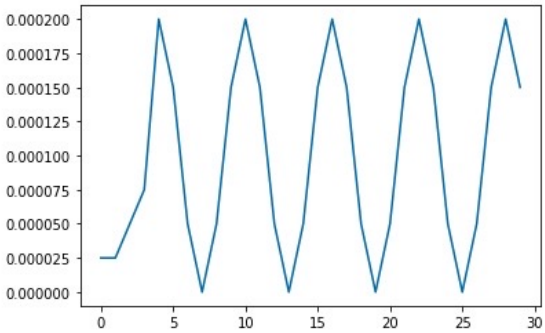


Figura 13: Taxa de aprendizagem em cada época do classificador de *patches*
Fonte: do autor

Como entrada utilizaram-se os *patches* do banco CBIS-DDSM na resolução 224 x 224 já criados por Petrini et al. [5]. E os hiperparâmetros da seção 3.2 foram todos mantidos da mesma maneira para a reprodução em GPU.

Ao passar a utilizar a TPU, apenas o *batch size* foi alterado, passando de 32 para 256. Foram mantidas as mesmas imagens utilizadas para o teste da reprodução, mas já em formato de *TFRecords*. Ao dobrar a resolução, realizou-se o processo descrito em 5.2.1.2 para criar as imagens de entrada.

5.2.5 Classificador de Imagem Inteira de Vista Única

Com o *patch classifier* treinado na etapa anterior como base, foram retiradas as camadas finais, obtendo assim apenas os *feature maps* treinados. Em seguida adicionaram-se, dois blocos *MBConv*, característicos da rede *EfficientNet*.

Utilizou-se a implementação dos blocos *MBConv* de [30]. Obtendo assim a arquitetura da rede esperada, com o número de *feature maps* e seus tamanhos iguais aos de Petrini *et al* [5]. A arquitetura deste componente pode ser vista na figura 14.

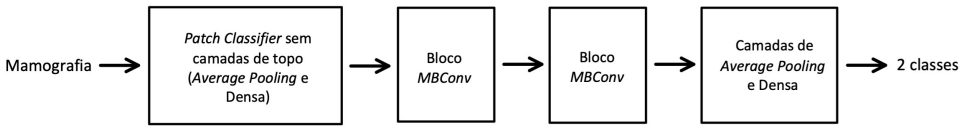


Figura 14: Arquitetura do classificador de vista única
Fonte: do autor

Para esse bloco, existem diversos parâmetros que podem ser configurados. Destacam-se o número de filtros de entrada, de saída, o tamanho do *kernel* e o *stride*. Para o componente desta seção, a quantidade de filtros é a mesma nos dois casos e vale 1280. O *kernel* utilizado é 3 x 3 com um *stride* de 2.

Conforme mencionado na seção 3.3, o *learning rate* também foi do tipo *cyclcic cosine* com *warmup*, e em 30 épocas obteve-se os valores apresentados na figura 15:

Para este componente foram selecionados os mesmos hiperparâmetros apresentados na seção 3.3. Um detalhe diferente é que utilizou-se um *dropout* de 0.8 ao invés de 0.9 utilizado originalmente.

Ao passar a utilizar a TPU, o *batch size* foi alterado, passando de 4 para 32 e o número de épocas foi reduzido para 30.

Mamografias já redimensionadas para 1152 x 896 por Petrini *et al.* [5] foram usadas

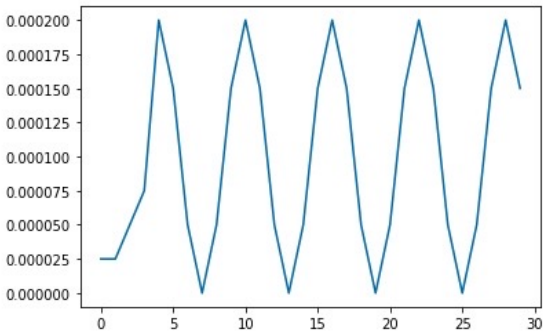


Figura 15: *Learning rate* em cada época de classificador de vista única
Fonte: do autor

para testar a reprodução e para os testes em TPU com a mesma resolução, mas serializadas em *TFRecords*. Já para a maior resolução usou-se como entrada o resultante do processo descrito em 5.2.1.1.

5.2.6 Classificador de Imagem Inteira de Duas Vistas

Utilizando dois classificadores de vista única, retirando as camadas finais e os blocos de *MBConv*, e concatenando os *feature maps* para em seguida adicionar dois blocos de *MBConv* e as camadas finais.

Importante notar uma ligeira diferença com o bloco utilizado no classificador, que tem todos os parâmetros iguais, exceto pelo número de filtros de entrada que passaram de 1280 para 2560.

O esquema do classificador completo está presente na figura 16:

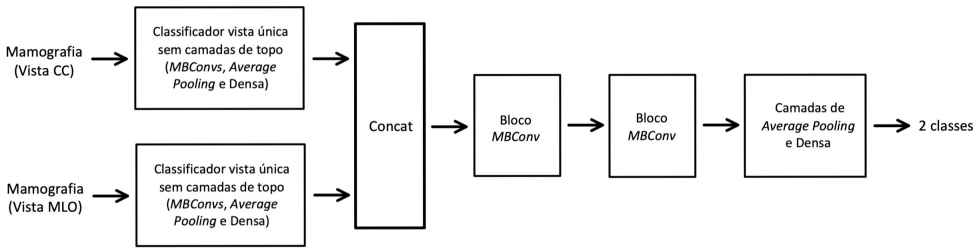


Figura 16: Arquitetura do classificador de duas vistas
Fonte: do autor

De maneira geral, os hiperparâmetros foram os apresentados na seção 3.4. Com exceção da taxa de aprendizagem, que optou-se pela variação desse seguindo a seguinte regra. Nas primeiras 20 épocas ele tem um valor de 10^{-4}, e todas as camadas são congeladas, com exceção da última. Com isso apenas a camada densa é treinada. Em seguida,

por mais 20 épocas, o *learning rate* passa para 10^{-6} e os dois blocos *MBConv* também passam a ser treinados.

Essa alteração foi feita devido a demora em rodar todo o processo por 100 épocas e também por não estar atingindo um resultado satisfatório seguindo o processo descrito na seção 3.4.

Para o teste da reprodução foram utilizadas as mamografias em vista CC e MLO com a resolução de 1152 x 896. São as mesmas utilizadas para o treinamento do classificador de vista única, mas agora em pares, sendo que se uma delas apresentava o rótulo maligno, o par recebia esse *label*. No caso dos exames de 2304 x 1792, cada par de exames eram encontrados e só então serializados.

Novamente para o teste de utilização com TPU, apenas o *batch size* foi alterado de 3 para 24.

5.2.7 Ajustes para uso da TPU

Além da utilização dos arquivos em formato *TFRecord*, outros detalhes foram necessários para o uso da TPU. Um primeiro cuidado foi utilizar uma implementação de *EfficienteNet* que fosse capaz de rodar nesse sistema. De uma maneira geral, as redes mais utilizadas apresentam implementações com essa característica, mas existem casos em que essa compatibilidade não ocorre.

Outro ponto importante é que o tamanho dos *batches* deve ser constante devido ao compilador que é utilizado para rodar na TPU, logo, é preciso remover o último *batch* caso ele não esteja completo.

Recomenda-se também o uso de um *batch size* múltiplo de oito, pois essa característica reduz a chance de ocorrerem gargalos devido a memória. O motivo é a estrutura do susbsistema de memória das TPUs que consegue funcionar de maneira mais eficiente seguindo essa recomendação.

Além disso, visando utilizar a TPU o maior tempo possível, é recomendável realizar o *prefetch* das imagens de entrada, ou seja, enquanto ocorrem os cálculos de gradiente, a próxima amostra a ser usada já deve ser buscada. Parte desse processo de carregamento é feito em CPU, para apenas depois as imagens serem enviadas para a TPU.

Finalmente, após se conectar com uma TPU, é preciso definir uma estratégia para o uso das TPU. Para o projeto utilizou-se uma que replica os pesos entre os diferentes *cores*

e os atualiza de maneira síncrona.

É preciso sempre compilar o modelo de modo que esteja dentro do escopo dessa estratégia para que possa ser carregado no dispositivo desejado. Caso isso não seja feito, o modelo rodará na CPU disponível, podendo inclusive apresentar um desempenho diferente em razão disso.

6 TESTES E AVALIAÇÃO

Nesta seção, o objetivo é apresentar os testes e a avaliação dos resultados dos diferentes componentes desenvolvidos e comparar as métricas obtidas.

Conforme apresentado na seção 4.1 deste trabalho, para a avaliação dos resultados são utilizadas as métricas acurácia e AUC. Também mostra-se o tempo médio por época e a matriz de confusão.

Os resultados estão apresentados para cada componente indicando sempre como cada resultado pode ser analisado.

Para cada um dos componentes, existem quatro versões a serem comparadas. A primeira é a versão original em *PyTorch* e os dados são retirados de [5]. A segunda corresponde a reprodução feita em *Tensorflow* rodando em uma GPU, usando a resolução original, 224x224 para o *patch* e 1152x896 para a imagem inteira.

A terceira ainda utiliza a resolução original, mas o acelerador escolhido é a TPU. Finalmente, dobra-se a resolução linear, ou seja, 2304x1792 para o exame completo e 448x448 para o *patch*, obtendo assim os resultados em TPU.

6.1 Resultados do Classificador de *Patch*

Para o treinamento, a GPU e a TPU utilizando imagens de mesma resolução apresentaram comportamento semelhante. Com isso a variação na acurácia para a partição de validação apenas da TPU está presente nas figura 17.

Importante ressaltar que em TPU, além de obter o mesmo resultado para a mesma resolução como será mostrado a seguir, também houve uma melhora significativa no tempo de treinamento, com tempo médio de uma época sendo de 19s, enquanto em GPU esse tempo é, em média, 363s. Ou seja, o uso da TPU diminui em 19 vezes o tempo gasto.

Na figura 18 está presente a evolução da acurácia para o treinamento com *patches* de

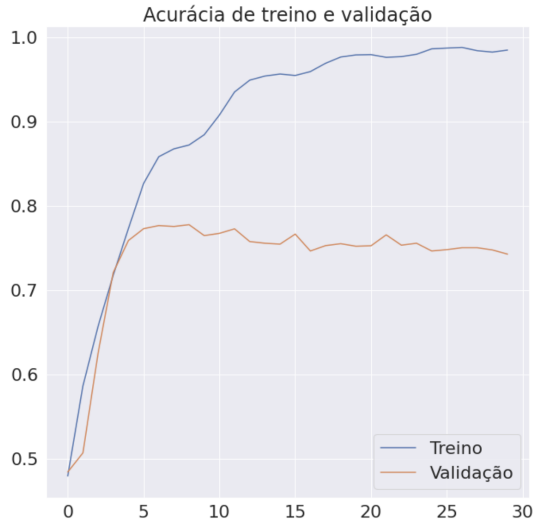


Figura 17: Acurácia durante treinamento do classificador de *patches* de 224 x 224
Fonte: do autor

tamanho 448 x 448. É interessante notar que a acurácia de validação convergiu em um valor mais alto do que na resolução original. Para este caso, o tempo de treino por época foi de 50s, mais de duas vezes o tempo na resolução inicial, porém menos que quatro vezes, valor esperado já que o número de *pixels* quadruplicou.

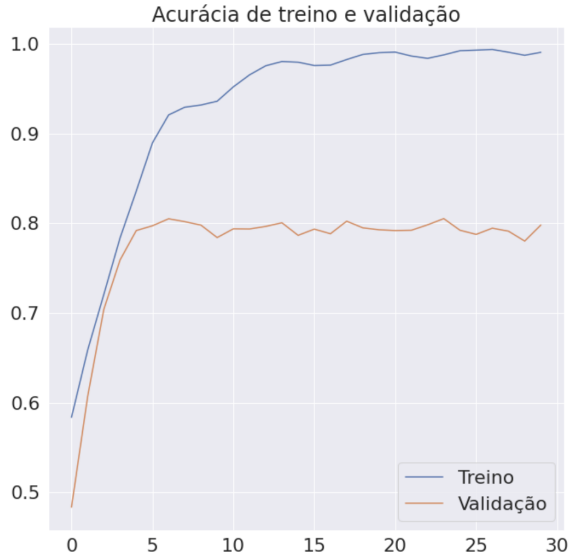


Figura 18: Acurácia durante treinamento do classificador de *patches* de 448 x 448
Fonte: do autor

Considerando os quatro experimentos citados em 6, são apresentados na tabela 2 os resultados para o banco de dados *CBIS-DDSM*. Os valores obtidos são os melhores para cada caso em *single run*.

Em primeiro lugar, foi preciso verificar se a reprodução funcionou através do teste em uma GPU, onde obteve-se uma acurácia de 74%, similar aos 75% obtidos na versão

Implementação	Resolução	Acurácia
Petrini *et al* [5] em *PyTorch*	224 x 224	75,54%
GPU em *Tensorflow*	224 x 224	74,04%
TPU em *Tensorflow*	224 x 224	76,37%
TPU em *Tensorflow*	448 x 448	79,52%

Tabela 2: Resultados do classificador de *patches* para CBIS-DDSM
Fonte: do autor

original, indicando que essa etapa funcionou corretamente.

Para o caso com o dobro da resolução linear, foi observado um aumento na acurácia de em torno de 4%. Através das matrizes de confusão pode-se verificar em qual classe houve uma melhora na classificação. Na figura 19 está presente essa matriz para a resolução original e na 20 para o dobro da resolução linear.

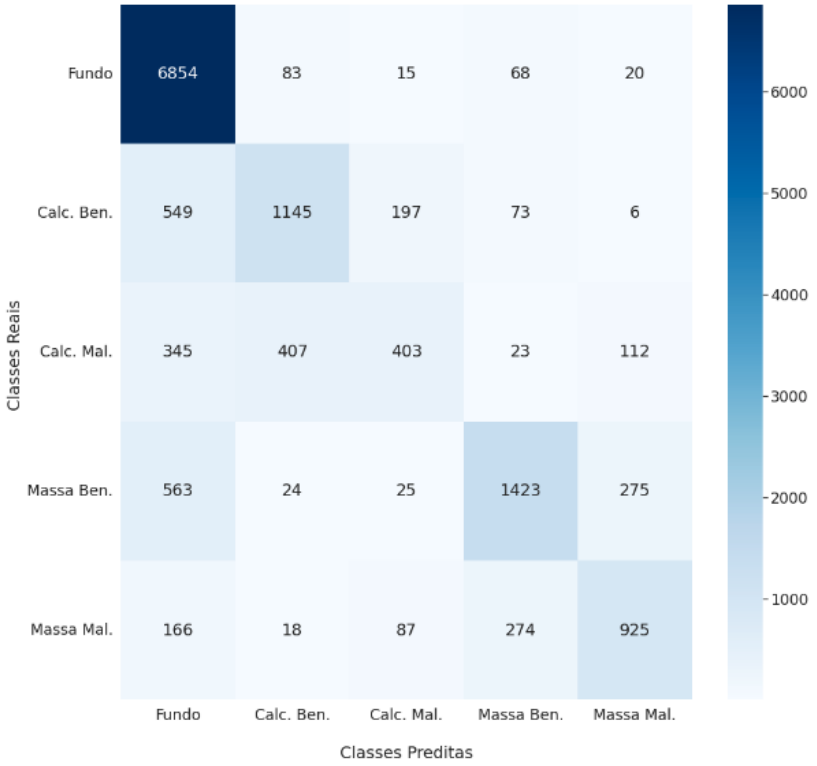


Figura 19: Matriz de confusão de classificador de *patches* de 224 x 224
Fonte: do autor

Figura 20: Matriz de confusão de classificador de *patches* de 448 x 448
Fonte: do autor

Através da análise das matrizes de confusão, pode-se observar que a melhora na classificação ocorreu principalmente nos casos de microcalcificação, sendo que houve um aumento mais expressivo na identificação da classe calcificação benigna. Isso indica que um menor redimensionamento facilita identificação desse tipo de câncer.

6.2 Resultados do Classificador de Imagem Inteira de Vista Única

Novamente os resultados aqui apresentados são os melhores obtidos para cada caso em uma única execução. Para o treinamento, pode-se verificar a evolução da acurácia para os dois últimos casos descritos no início da seção 6 nas figuras 21 e 22, em razão do resultado de GPU e TPU serem bastante similares na mesma resolução.

O tempo de treinamento por época médio foi de 376s para a menor resolução com GPU e de 22s para esse tamanho de imagem utilizando TPU. Com esses valores, o *ASIC* obteve um ganho comparável com o do componente anteriormente apresentado, sendo 17 vezes mais rápido que a GPU. Ao dobrar a resolução linear, esse tempo passou a ser 71s, ou seja, uma aumento de mais de quatro vezes no tempo para percorrer todo o *dataset*

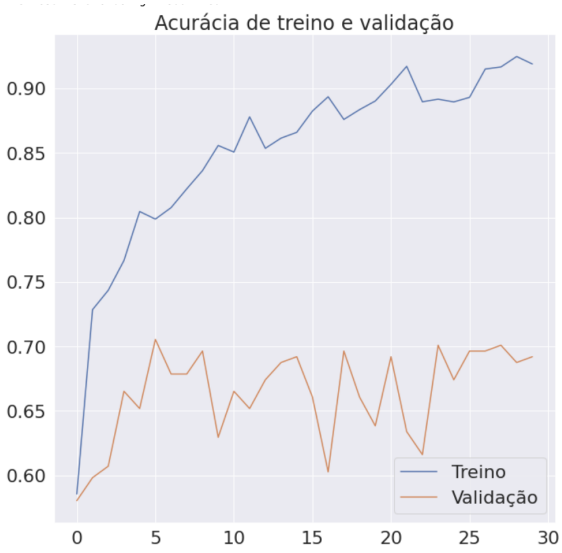


Figura 21: Acurácia durante treinamento do classificador de vista única de 1152 x 896 em TPU

Fonte: do autor

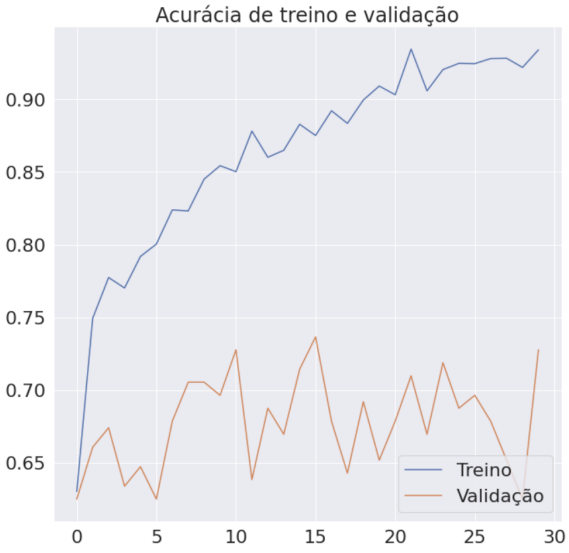


Figura 22: Acurácia durante treinamento do classificador de vista única de 2304 x 1792 em TPU

Fonte: do autor

com a resolução maior.

Para os casos descritos na seção 6, os resultados obtidos estão presentes na tabela 3.

Com base nos dados apresentados, pode-se notar que a reprodução funcionou corretamente. As AUCs da versão original e as em *Tensorflow* são muito similares para a mesma resolução, sendo que uma está contida no intervalo da outra.

Considerando o aumento de resolução, ocorre apenas um ligeiro aumento na AUC não permitindo concluir definitivamente que ocorreu uma melhora na classificação.

Implementação	**Resolução**	***Area Under Curve***
Petrini *et al* [5] em *PyTorch*	1152 x 896	$0,8033 \pm 0,0183$
GPU em *Tensorflow*	1152 x 896	$0,8143 \pm 0,0179$
TPU em *Tensorflow*	1152 x 896	$0,8007 \pm 0,0184$
TPU em *Tensorflow*	2304 x 1792	$0,8154 \pm 0,0178$

Tabela 3: Resultados do classificador de vista única para CBIS-DDSM
Fonte: do autor

Nas figuras 27 e 28 são apresentadas as matrizes de confusão para diferentes tamanhos de imagem de entrada.

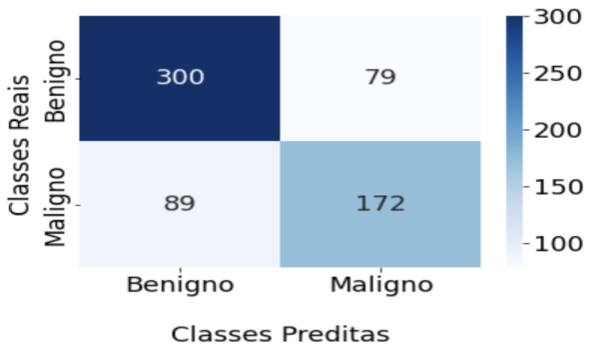


Figura 23: Matriz de confusão do classificador de vista única de 1152 x 896 em TPU
Fonte: do autor

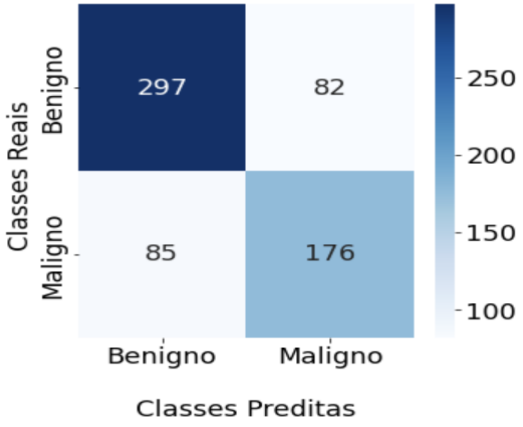


Figura 24: Matriz de confusão do classificador de vista única de 2304 x 1792 em TPU
Fonte: do autor

Para as matrizes apresentadas, nota-se que os resultados novamente são muito semelhantes. A acurácia em cada caso, foi de 73,75% para a menor resolução e 73,91% para a maior, valores bastante próximos. Esses dados são indicativos de que não houve uma

melhora no resultado obtido com o menor redimensionamento das imagens.

6.3 Resultados do Classificador de Imagem Inteira de Duas Vistas

Novamente a primeira análise é em quanto ao tempo. Para GPU na resolução original, o tempo médio por época foi de 73s e na TPU foi de 17s, ou seja, a última foi 4,29 vezes mais rápida que a primeira. Com a maior resolução testada, esse tempo passou a ser 28s.

O avanço do treinamento pode ser acompanhado pela acurácia tanto na partição de treino e validação para os diferentes casos testados nas figuras 25 e 26.

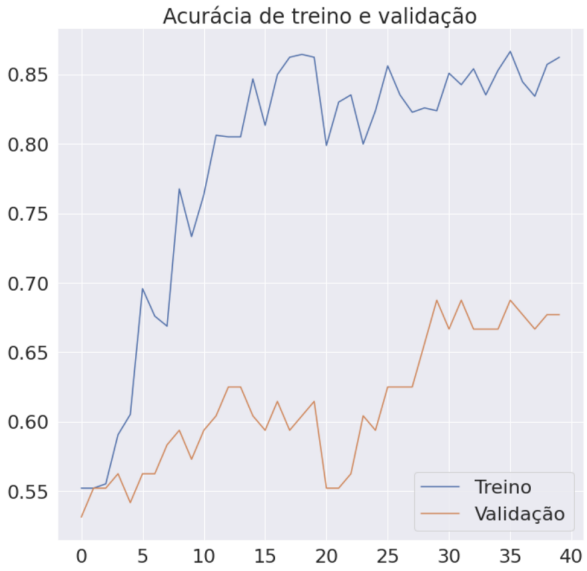


Figura 25: Acurácia durante treinamento do classificador de duas vistas de 1152 x 896 em TPU

Fonte: do autor

Na época 20 é importante relembrar que ocorreu uma alteração na taxa de aprendizagem e também nas camadas que estavam sendo treinadas. Conforme observável nos gráficos, essa mudança permitiu que o modelo continue aprendendo.

Comparando os casos em TPU e com diferentes resoluções, nota-se que ocorreu a convergência da acurácia de validação para um valor ligeiramente maior se comparada à original.

A métrica escolhida para o classificador de duas vistas é a *Area Under Curve* da curva ROC, e para cada um dos casos definidos em 6 o resultado está presente na tabela 4.

De maneira similar ao classificador de uma vista, o resultado da reprodução em *Ten-*

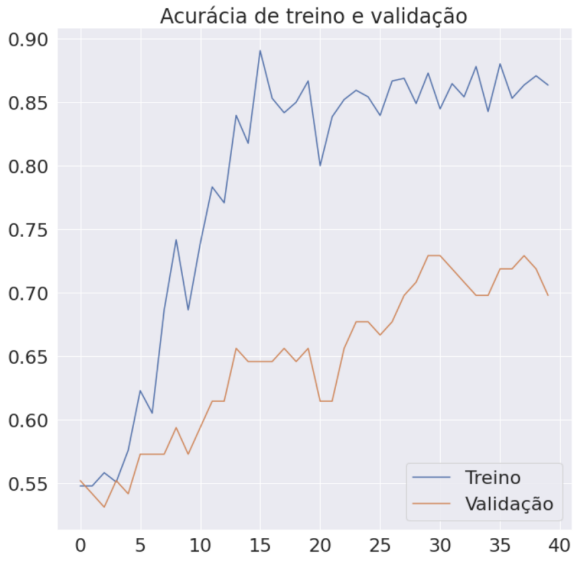


Figura 26: Acurácia durante treinamento do classificador de duas vistas de 2304 x 1792 em TPU

Fonte: do autor

Implementação	**Resolução**	***Area Under Curve***
Petrini *et al* [5] em *PyTorch*	1152 x 896	$0,8418 \pm 0,0258$
GPU em *Tensorflow*	1152 x 896	$0,8498 \pm 0,0227$
TPU em *Tensorflow*	1152 x 896	$0,8327 \pm 0,0264$
TPU em *Tensorflow*	2304 x 1792	$0,8466 \pm 0,0264$

Tabela 4: Resultados do classificador de duas vistas para CBIS-DDSM
Fonte: do autor

sorflow apresenta a métrica escolhida como parte do intervalo a ser considerado do trabalho original [5] em *PyTorch*. E o mesmo ocorre para a GPU e TPU com a mesma resolução.

Com a imagem de entrada maior, aparentemente há um leve aumento na AUC, mas não é um valor significativo o suficiente para concluir que existe uma melhora na classificação.

Nas figuras 27 e 28 são apresentadas as matrizes de confusão para os diferentes redimensionamentos aplicados.

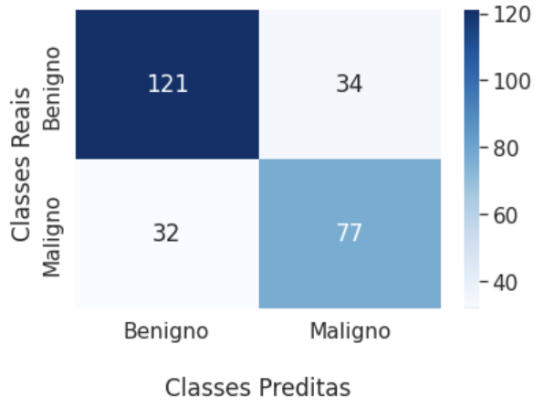


Figura 27: Matriz de confusão do classificador de duas vistas de 1152 x 896 em TPU
Fonte: do autor

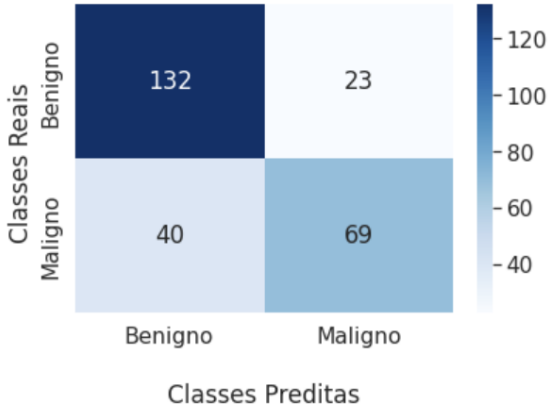


Figura 28: Matriz de confusão do classificador de duas vistas de 2304 x 1792 em TPU
Fonte: do autor

Com os dados apresentados e sabendo que a acurácia para a resolução 1152 x 896 foi de 75% e para 2304 x 1792 foi 76,13%, novamente não se pode afirmar que houve melhora na classificação devido à similaridade nos resultados.

De maneira resumida, existe uma melhora considerável com a utilização de uma maior resolução apenas para o classificador de *patches*, em especial na classificação de microcalcificações.

Para o classificador de vista única e o de duas vistas, apesar de uma métrica ligeiramente melhor, o aumento não é significativo o suficiente para afirmar que se obteve um melhor classificador ao utilizar imagens de entrada maiores.

O aumento não significativo pode ser devido ao CBIS-DDSM ser um banco de dados com mamografias escaneadas, com isso, já ocorreu uma perda de informação suficiente-

mente grande para não tirar proveito da maior resolução,

Outro ponto importante a mencionar é que o uso de TPU acelera o processo de treinamento consideravelmente. No pior caso o ganho foi de quatro vezes, chegando a ser 19 vezes mais rápida que uma GPU para os modelos desenvolvidos. A diminuição no tempo de treinamento facilita o desenvolvimento dos modelos, pois permite reduzir o tempo entre os testes e, consequentemente, o de desenvolvimento.

7 CONSIDERAÇÕES FINAIS

7.1 Conclusão do Projeto de Formatura

O objetivo deste projeto era determinar se é possível melhorar a capacidade de classificação um modelo baseado em uma rede neural convolucional utilizando uma TPU, em particular explorando a maior memória desse dispositivo. Essa característica permite a utilização de imagens de entrada de maior resolução.

Foram elaborados e ajustados modelos para verificar essa hipótese, sempre levando em consideração que o desenvolvimento para o uso da TPU apresenta detalhes que devem ser considerados para um bom funcionamento.

Apesar dos cuidados a serem tomados, não foi necessariamente a etapa mais complicada do projeto, pois uma vez com um modelo funcional em *Tensorflow* a adaptação para uso desse *ASIC* não apresenta grandes complicações. A passagem de *PyTorch* para *Tensorflow* terminou apresentando uma maior quantidade de obstáculos a serem ultrapassados do que a adaptação.

7.2 Contribuições

Com base nos resultados obtidos, obtém-se uma melhora na classificação de *patches*, mas ao utilizar as mamografias completas a melhora obtida não foi significativa. Com isso, não se pode concluir que o uso de uma maior resolução melhora a classificação para a arquitetura utilizada e com o banco de dados escolhido.

Apesar de não garantir uma melhora na classificação, o uso da TPU não foi completamente em vão. A velocidade obtida com a utilização dela pode ser de grande utilidade, principalmente conforme um aumento no tamanho das redes e também na quantidade de dados utilizadas.

7.3 Perspectivas de Continuidade

Para este trabalho testou-se em um banco de dados do com mamografias escaneadas, uma possibilidade de continuidade seria realizar o teste um banco com imagens inerentemente digitais. Pois no processo de escaneamento já ocorre a perda de informações independentemente do redimensionamento posterior dos exames.

Além disso, foi necessário o redimensionamento das imagens de entrada e utilizaram-se técnicas tradicionais de amostragem. Atualmente existem redes neurais que realizam o redimensionamento, sendo a utilização dessas uma alternativa de continuação do trabalho.

Finalmente, outra possibilidade seria a utilização da vista complementar da mamografia para auxiliar o modelo no processo de classificação.

REFERÊNCIAS

[1] SUNG, H. et al. Global cancer statistics 2020: Globocan estimates of incidence and mortality worldwide for 36 cancers in 185 countries. *CA: A Cancer Journal for Clinicians*, v. 71, n. 3, p. 209–249, 2021. Disponível em: <https://acsjournals.onlinelibrary.wiley.com/doi/abs/10.3322/caac.21660>. Acesso em: 11 abr. 2022.

[2] WILD, C. P.; WEIDERPASS, E.; STEWART, B. W. *World Cancer Report: Cancer Research for Cancer Prevention*. Lyon, França, 2020. Disponível em: <https://publications.iarc.fr/586>. Acesso em: 15 abr. 2022.

[3] RODRIGUEZ-RUIZ, A. et al. Stand-alone artificial intelligence for breast cancer detection in mammography: Comparison with 101 radiologists. *Journal of the National Cancer Institute*, 2019. Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6748773/>. Acesso em: 15 fev. 2022.

[4] SHEN, L.; MARGOLIES, L. R.; ROTHSTEIN, J. H. Deep learning to improve breast cancer detection on screening mammography. *Scientific Reports*, 2019. Disponível em: <https://www.nature.com/articles/s41598-019-48995-4>. Acesso em: 8 dez. 2022.

[5] PETRINI, D. G. P. et al. Breast cancer diagnosis in two-view mammography using end-to-end trained efficientnet-based convolutional network. *IEEE Access*, v. 10, p. 77723–77731, 2022. Disponível em: <https://ieeexplore.ieee.org/document/9837037>. Acesso em: 3 jan. 2022.

[6] ONCOGUIA. 2014. Disponível em: <http://www.oncoguia.org.br/conteudo/mamografia/6797/842/>. Acesso em: 15 mar. 2022.

[7] PEIXOTO, J. E.; CANELLA, E.; AZEVEDO, A. C. *Mamografia: da prática ao controle*. Rio de Janeiro, Ministério da Saúde, Instituto Nacional de Câncer, 2007. Disponível em: <https://www.inca.gov.br/sites/ufu.sti.inca.local/files//media/document//mamografia-pratica-controle-2007.pdf>. Acesso em: 15 mar. 2022.

[8] JALES, R. M. *Fundamentos da propedêutica por imagem da mama*. 2016. Disponível em: <https://drpixel.fcm.unicamp.br/conteudo/fundamentos-da-propedeutica-por-imagem-da-mama>. Acesso em: 15 mar. 2022.

[9] GOODFELLOW, I. J.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[10] AGGARWAL, C. C. *Neural Networks and Deep Learning*: A textbook. Cham: Springer, 2018. 497 p. ISBN 978-3-319-94462-3.

[11] WU, Y. et al. Demystifying learning rate policies for high accuracy training of deep neural networks. In: *2019 IEEE International Conference on Big Data (Big Data)*. [s.n.], 2019. p. 1971–1980. Disponível em: <https://ieeexplore.ieee.org/document/9006104>. Acesso em: 10 abr. 2022.

[12] GOTMARE, A. et al. *A Closer Look at Deep Learning Heuristics: Learning rate restarts, Warmup and Distillation*. arXiv, 2018. Disponível em: <https://arxiv.org/abs/1810.13243>. Acesso em: 3 jan. 2022.

[13] LECUN, Y. et al. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, v. 1, n. 4, p. 541–551, 1989. Disponível em: <https://ieeexplore.ieee.org/document/6795724>. Acesso em: 22 jan. 2022.

[14] GOOGLE. *Cloud TPU Documentation*. 2022. Disponível em: <https://cloud.google.com/tpu/docs/>. Acesso em: 18 mar. 2022.

[15] WANG, Y. E.; WEI, G.-Y.; BROOKS, D. *Benchmarking TPU, GPU, and CPU Platforms for Deep Learning*. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1907.10701>. Acesso em: 3 abr. 2022.

[16] TENSORFLOW. *Use TPUs*. 2022. Disponível em: <https://www.tensorflow.org/guide/tpu>. Acesso em: 28 mar. 2022.

[17] GLASMACHERS, T. *Limits of End-to-End Learning*. arXiv, 2017. Disponível em: <https://arxiv.org/abs/1704.08305>. Acesso em: 5 abr. 2022.

[18] DENG, J. et al. Imagenet: A large-scale hierarchical image database. In: IEEE. *2009 IEEE conference on computer vision and pattern recognition*. 2009. p. 248–255. Disponível em: <https://ieeexplore.ieee.org/document/5206848>. Acesso em: 12 jan. 2022.

[19] TAN, M.; LE, Q. V. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2019. Cite arxiv:1905.11946Comment: Published in ICML 2019. Disponível em: <http://arxiv.org/abs/1905.11946>. Acesso em: 22 jan. 2022.

[20] LEE, R. S. et al. *Curated Breast Imaging Subset of DDSM*. The Cancer Imaging Archive, 2016. Disponível em: <http://dx.doi.org/10.7937/K9/TCIA.2016.7O02S9CY>. Acesso em: 8 jan. 2022.

[21] HANLEY, J. A.; MCNEIL, B. J. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, v. 143, n. 1, p. 29–36, 1982. PMID: 7063747. Disponível em: <https://doi.org/10.1148/radiology.143.1.7063747>. Acesso em: 21 jan. 2022.

[22] GOOGLE. *Cloud TPU Pricing*. 2022. Disponível em: <https://cloud.google.com/tpu/pricing>. Acesso em: 28 mar. 2022.

[23] ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Disponível em: <https://www.tensorflow.org/>. Acesso em: 18 mar. 2022.

[24] CHOLLET, F. et al. *Keras*. 2015. Disponível em: <https://keras.io>. Acesso em: 18 mar. 2022.

[25] PASZKE, A. et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. Disponível em: <https://pytorch.org/>. Acesso em: 18 mar. 2022.

[26] KAGGLE. *How to Use Kaggle*. 2022. Disponível em: <https://www.kaggle.com/docs/tpu>. Acesso em: 18 mar. 2022.

[27] OWSEF. *CBIS-DDSM: Breast Cancer Image Dataset, Version 1*. 2020. Disponível em: <https://www.kaggle.com/datasets/awsaf49/cbis-ddsm-breast-cancer-image-dataset>. Acesso em: 8 jun. 2022.

[28] TENSORFLOW. *TFRecord and tf.train.Example*. 2022. Disponível em: <https://www.tensorflow.org/tutorials/load_data/tfrecord/>. Acesso em: 21 abr. 2022.

[29] JIN, H. et al. *Keras: Deep Learning for humans*. GitHub, 2022. Disponível em: <https://github.com/keras-team/keras/blob/v2.10.0/keras/applications/efficientnet.py>. Acesso em: 21 abr. 2022.

[30] GROOS, D. et al. *EfficientNet Keras (and TensorFlow Keras)*. GitHub, 2022. Disponível em: <https://github.com/qubvel/efficientnet/blob/master/efficientnet/model.py>. Acesso em: 18 abr. 2022.