UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

LEODÉCIO BRAZ DA SILVA SEGUNDO

CLASSIFICAÇÃO DE PERSONAGENS ANIMADOS USANDO REDES NEURAIS CONVOLUCIONAIS PROFUNDAS PRÉ-TREINADAS E FINE-TUNING

QUIXADÁ

2018

LEODÉCIO BRAZ DA SILVA SEGUNDO

CLASSIFICAÇÃO DE PERSONAGENS ANIMADOS USANDO REDES NEURAIS CONVOLUCIONAIS PROFUNDAS PRÉ-TREINADAS E FINE-TUNING

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Paulo de Tarso Guerra Oliveira

QUIXADÁ

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S581c Silva Segundo, Leodecio Braz da.
Classificação de Personagens Animados usando Redes Neurais Convolucionais Profundas Pré-treinadas e Fine-tuning / Leodecio Braz da Silva Segundo. – 2018.
59 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2018.
Orientação: Prof. Dr. Paulo de Tarso Guerra Oliveira.

1. Aprendizagem Profunda. 2. Transferência de Aprendizagem. 3. Rede Neural Convolucional. 4. Imagem - Classificação. I. Título.

CDD 004

LEODÉCIO BRAZ DA SILVA SEGUNDO

CLASSIFICAÇÃO DE PERSONAGENS ANIMADOS USANDO REDES NEURAIS CONVOLUCIONAIS PROFUNDAS PRÉ-TREINADAS E FINE-TUNING

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: _____/_____/_____.

BANCA EXAMINADORA

Prof. Dr. Paulo de Tarso Guerra Oliveira (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Criston Pereira de Souza
Universidade Federal do Ceará (UFC)

Profa. Dra. Ticiana Linhares Coelho da Silva
Universidade Federal do Ceará (UFC)

À minha mãe, por todo seu cuidado e dedicação durante toda a minha caminhada como estudante, meu Pai por todos os seus esforços para prover o sustento da nossa família e por ser a figura que me motiva a ser, e minha Irmã por ser minha fonte de inspiração em todos os momentos.

AGRADECIMENTOS

Agradeço primeiramente a Deus por todas a bençãos concedidas em minha vida.
Agradeço a minha Mãe e meu Pai a quem muito eu amor, por todo carinho e cuidado dedicado a mim e por sempre serem a minha referência como pessoa.
A minha irmã Leonara por ser a pessoa em quem me inspiro e ao meu cunhado Roberto por todas as vezes que estava disponível a me ajudar e me aconselhar, muito obrigado.
Aos meus amigos Alison, Erisson, Erly Júnior, Emy, Evandro, Jefferson, Matheus e Wendel que são irmãos que a vida me deu, agradeço por todas as histórias compartilhadas nesses vários anos de amizade.
A minha tia Alzenir e seu esposo Abinoer, por terem me acolhido e por toda ajuda nos momentos que precisei, meu muito obrigado.
Aos meus amigos que a faculdade me permitiu conhecer, toda a minha turma de Ciência da Computação, agradeço pelos momentos que passamos ao longo desses quatro anos de curso.
Agradeço também ao Décio por todo o apoio, e pelas coisas que me ensinou que foi de grande importância para este trabalho, obrigado.
Agradeço aos meu amigos, Beatriz, Jêsca, Robert, Sávio, Yuri e Vinícius que estão comigo desde o ensino médio, por todos os momentos de conversas, estudos e lazer que realizamos.
Agradeço ao meu orientador Paulo de Tarso por todas as conversas, todos os conselhos e orientações direcionadas a mim que me ajudaram muito durante esta jornada.
E a todos que fizeram parte da minha vida e que de alguma maneira contribuíram em minha formação, meu mais sincero obrigado.

“Nenhum esforço faz sentido, se você não acredita em si mesmo”

(Maito Gai)

RESUMO

Neste presente trabalho, o autor utiliza a transferência de conhecimento e analisa o uso da técnica de *fine-tuning* em redes neurais convolucionais para a classificação de personagens da série animada “Os Simpsons”. A transferência de conhecimento é realizada a partir do uso de redes neurais profundas já treinadas sobre um grandes conjuntos de dados de múltiplas classes e a técnica de *fine-tuning* consiste em realizar um congelamento de alguns pesos e modificar outros de um modelo já treinado, sobre um novo conjunto de dados. Estudos sobre essa técnica relatam que algumas representações aprendidas anteriormente por um modelo são adaptadas a um novo problema. Uma preocupação presente neste trabalho foi quanto a evitar o *overfitting* durante o processo de aprendizagem dos modelos. Para isso foram introduzidos hiperparâmetros como regularização e *dropout*. Duas arquiteturas de redes neurais foram utilizadas, o uso do *fine-tuning* nos modelos, sobre os dados, ocasionou uma melhora significativa nos resultados, melhorando o desempenho da classificação.

Palavras-chave: Aprendizado Profundo. Transferência de Conhecimento. Redes Neurais Convolucionais. Classificação de Imagens.

ABSTRACT

In this work, the author uses knowledge transfer and analyzes the use of the fine-tuning technique in convolutional neural networks for the character classification of the animated series “The Simpsons”. Knowledge transfer is carried out using deep neural networks already trained on large datasets of multiple classes, and the fine-tuning technique consists of carrying out a freezing of some weights and modifying others of an already trained model, on a new set of data. Studies about this technique report that some representations previously learned by a model are adapted to a new problem, which increases the performance of the model a little more. One concern present in this work was to avoid overfitting during the learning process of the models. For this, hyperparameters were introduced as regularization and dropout. Two neural network architectures were used, the use of fine-tuning in the models, on the data, caused a significant improvement in the results, improving the classification performance.

Keywords: Deep Learning. Knowledge Transfer. Convolutional Neural Networks. Image Classification.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

RNA Redes Neurais Artificiais

RNC *Redes Neurais Convolucionais*

SUMÁRIO

1 INTRODUÇÃO

Técnicas de aprendizagem profunda, principalmente por meio do uso de Redes Neurais Convolucionais (RNC) vêm sendo bastante utilizadas na área de visão computacional para diversas tarefas. O uso de RNC tem sido o estado da arte em visão computacional promovendo avanços significativos na área, a medida que estas arquiteturas fornecem bons resultados para classificação de objetos (BEZERRA, 2016).

O uso de redes neurais convolucionais tem sido útil em diversas aplicações de visão computacional, reconhecimento e processamento de imagens. O grande uso destas redes se dá pelo fato de que estudos recentes relatam que estas arquiteturas fornecem bons resultados para classificação de imagens, como em Krizhevsky *et al.* (2012) e Russakovsky *et al.* (2015), detecção de objetos em imagens, como em Szegedy *et al.* (2015), reconhecimento de cena (ZHOU *et al.*, 2014), e para muitos outros problemas.

Nesse cenário, diferentes trabalhos vêm sendo desenvolvidos utilizando aprendizagem profunda para problemas de visão computacional, como em Aguiar (2017), onde o autor propôs um sistema de auxílio ao diagnóstico médico para classificar câncer de mama em imagens histopatológicas, e em Hosang *et al.* (2015), onde os autores investigam o uso de redes convolucionais para a detecção de pedestres nas ruas.

Contudo, em visão computacional, a tarefa de classificar imagens é considerada ainda uma tarefa complexa (AGUIAR, 2017). Um dos maiores desafios em se utilizar modelos de aprendizagem profunda, se dá pelo fato de que esta tarefa demanda uma grande quantidade de informações contidas em bases de dados, isto é, se faz necessário um amplo conhecimento prévio sobre o problema de maneira geral. Por exemplo, para reconhecer um cachorro e um gato em uma foto, primeiramente é necessário que se saiba o que é um cachorro, o que é um gato e diferenciá-los.

Além da necessidade de uma grande quantidade de dados disponível, o que nem sempre é possível, o treinamento de redes neurais profundas também exige um certo poder computacional que muitas vezes é escasso ou limitado, o que dificulta realizar o treinamento de uma rede desde o início para um novo conjunto de dados.

Como alternativa para suprir a falta de uma grande quantidade de dados para treinar um modelo, a técnica de transferência de conhecimento tem-se mostrado bastante promissora. Esta técnica consiste em utilizar uma rede previamente treinada sobre uma grande quantidade de dados para que o conhecimento dela possa ser utilizado como entrada de treinamento para

um novo modelo sobre novos dados, reduzindo o tempo e processamento necessário para treinar esse modelo desde o início para os novos dados.

Um conceito bastante semelhante ao de transferência de conhecimento e que por vezes são utilizados quase que de forma intercambiável, é o conceito de *Fine-tuning*, que consiste em congelar algumas camadas de um modelo já treinado sob alguns dados e, em seguida, treiná-lo um pouco mais possivelmente sob dados diferentes. Essa técnica é bastante útil quando os dados que o modelo é submetido não são conhecidos pelo modelo da transferência de conhecimento, sendo assim, apenas algumas partes destes conhecimentos são aproveitadas, enquanto o modelo treina para os novos dados.

Uma preocupação durante o processo de aprendizagem da rede é quanto ao *overfitting*, que consiste da rede memorizar os dados de treino, aprendendo apenas padrões específicos demais. E quando submetido a um novo conjunto, não conseguir generalizar bem e assim, não realizar uma boa classificação.

Este trabalho tem como objetivo analisar o uso de transferência de conhecimento com duas diferentes arquiteturas de redes neurais profundas. Para isso realizou-se a classificação de imagens de personagens animados da base "The Simpsons Characters Data"[1]. Analisou-se também o efeito que a técnica de *fine-tuning* tem sobre a classificação, bem como o controle de *overfitting* por meio de hiper-parâmetros, como regularização e *dropout*.

Este trabalho está organizado da seguinte forma. No Capítulo 2 é apresentada a fundamentação teórica, onde são tratados os conceitos abordados neste trabalho que são importantes para a compreensão do leitor. No Capítulo 3 são apresentados os trabalhos relacionados com este tema de pesquisa, que serviram como inspiração e deram suporte a este trabalho. No Capítulo 4 é descrito a base de dados utilizada e o pré-processamento necessário, bem como os passos realizados para a execução deste trabalho, os experimentos que foram realizados, e o resultado obtido no teste. No Capítulo 5 são mostrados alguns experimentos adicionais que foram realizados com subconjuntos dos dados originais, visando analisar os desempenhos das redes. Por fim, no Capítulo 6 são apresentadas as conclusões obtidas neste trabalho.

[1] https://www.kaggle.com/alexattia/the-simpsons-characters-dataset

2 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo, são apresentados os fundamentos que baseiam este trabalho e que são necessários ao leitor, para que haja uma compreensão teórica do contexto em que este trabalho está inserido e das técnicas utilizadas.

Na Seção 2.1, é mostrado o conceito e o estado da arte da área de Visão Computacional. Na Seção 2.2, são apresentados os conceitos de Redes Neurais, Redes Neurais Profundas e Redes Neurais Convolucionais bem como o conceito de aprendizagem. Na Seção 2.3, são apresentados os conceitos de Transferência de Conhecimento e o de *fine-tuning*. Na Seção 2.4, são apresentadas as métricas para Avaliação de Desempenho para a classificação.

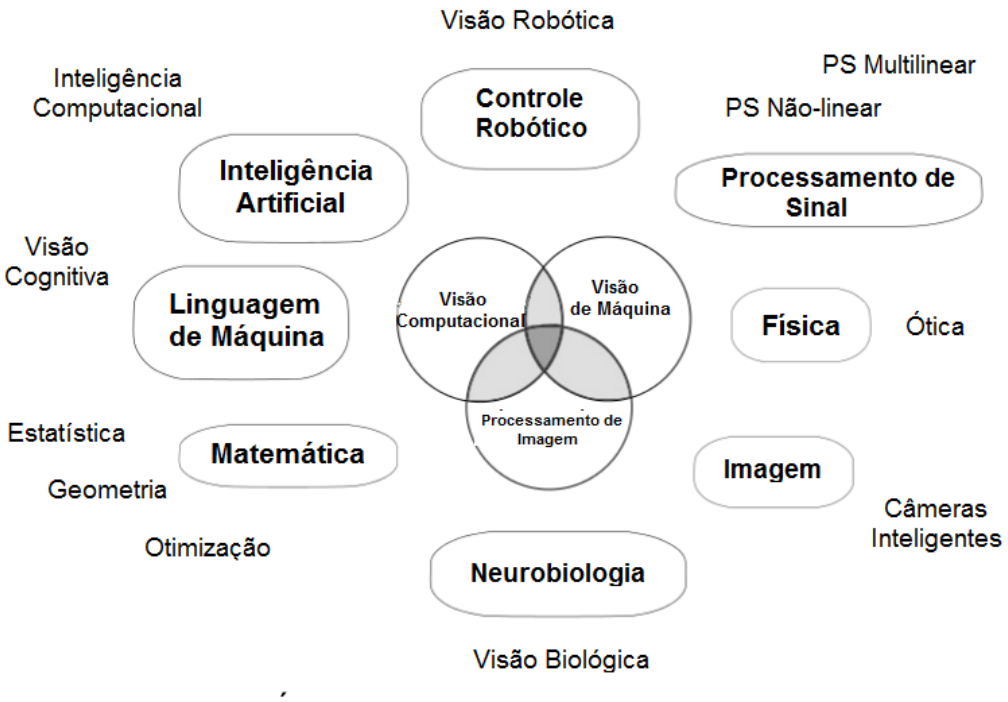
2.1 VISÃO COMPUTACIONAL

Os seres humanos têm a facilidade de perceber estruturas e reconhecer objetos a sua volta e sistemas de visão computacional buscam dar às máquinas a habilidade de reconhecer e interpretar imagens de maneira tão eficiente quanto os humanos (SZELISKI, 2010).

Visão computacional é o ramo da Ciência da Computação que busca construir descrições explícitas e significativas de objetos físicos a partir de imagens (BALLARD; BROWN, 1982).

A visão computacional pode ser aplicada em diferentes áreas de estudo como no campo de inteligência artificial, onde é bastante utilizada em tarefas como reconhecimento, identificação e detecção de objetos em imagens. A Figura 1 mostra os diversos campos que estão relacionados com visão computacional, como processamento de sinais e de imagens.

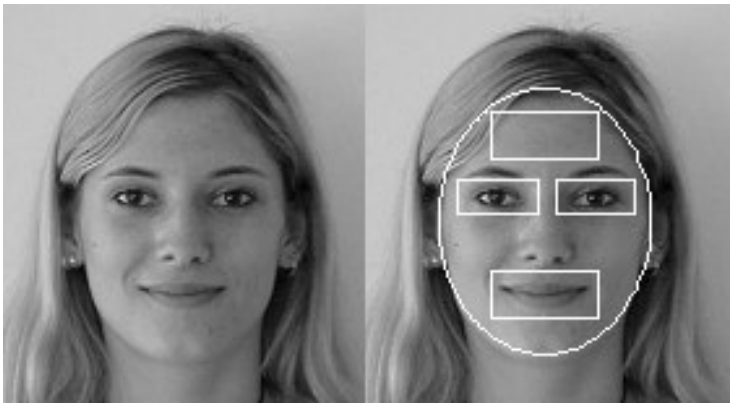
Figura 1 – Áreas relacionadas ao estudo de visão computacional



FONTE: Aguiar (2017)

A visão computacional pode também ser utilizada em diversas aplicações de diferentes áreas além da informática, como na medicina e segurança. Na medicina, qualquer área que esteja relacionada à análise visual de algo, como em diagnósticos por imagens, são potenciais campos de uso da visão computacional (RIOS, 2011). Na segurança, aplicações que utilizem reconhecimento de padrões para detectar boca, olhos e pele no processo de reconhecimento de faces, como é possível visualizar na Figura 2, auxiliam, por exemplo, a localizar criminosos por meio de câmeras de segurança.

Figura 2 – Exemplo de reconhecimento de face. Os traços indicam locais onde se esperam encontrar olhos, boca e pele. Reconhecer estas partes pode levar ao reconhecimento da face



FONTE: Marengoni e Stringhini (2009)

Redes neurais, em particular, redes neurais convolucionais aplicadas a problemas de visão computacional, têm-se obtido bons resultados em cenários como, classificação de imagens, reconhecimento e detecção de objetos, entre outros. Como no caso do desafio do *ImageNet* que tem desempenho melhor que de um ser humano.

2.2 REDES NEURAIS

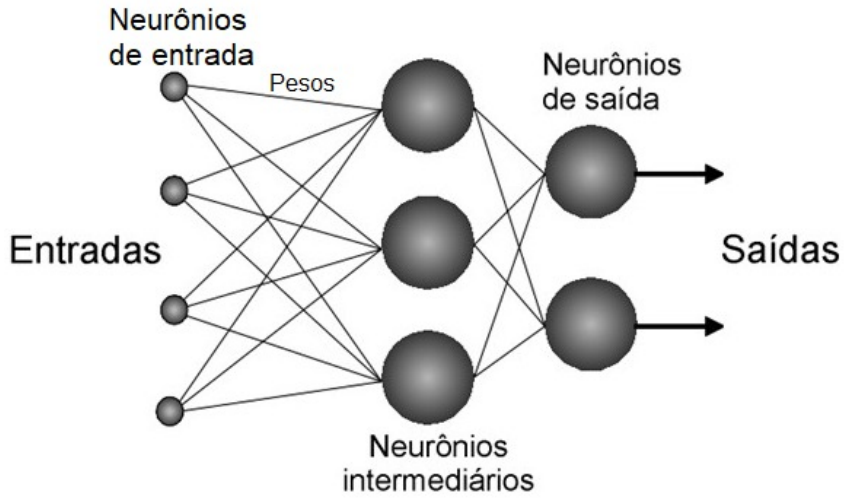
As Redes Neurais Artificiais (RNA), comumente denominadas como Redes Neurais, tiveram a sua motivação na neurociência, a partir do estudo do funcionamento do cérebro humano onde se percebeu a existência de conjuntos de neurônios que transferem informações entre si por meio de impulsos elétricos, através de junções formadas com outras células nervosas, conhecidas como sinapses.

De forma geral, uma rede neural é uma máquina projetada para modelar a maneira como o cérebro realiza uma tarefa particular ou determinada função de interesse. Uma RNA é composta por unidades de processamento também chamadas de “neurônios” que estão conectadas, realizando operações e armazenando conhecimento experimental. Este conhecimento é adquirido pela rede por meio de um **processo de aprendizagem** (HAYKIN, 2007).

Entre os neurônios de uma rede há agentes de conexão chamados de pesos sinápticos que são utilizados para armazenar o conhecimento adquirido. No processo de aprendizagem de uma rede, esses pesos sinápticos são modificados de forma ordenada a fim de alcançar o objetivo desejado.

A Figura 3 ilustra uma rede neural artificial. Os neurônios da camada de entrada se relacionam com os neurônios da camada intermediária, e cada neurônio desta camada se liga aos da camada de saída (CRESPO, 2017).

Figura 3 – Esboço de uma RNA



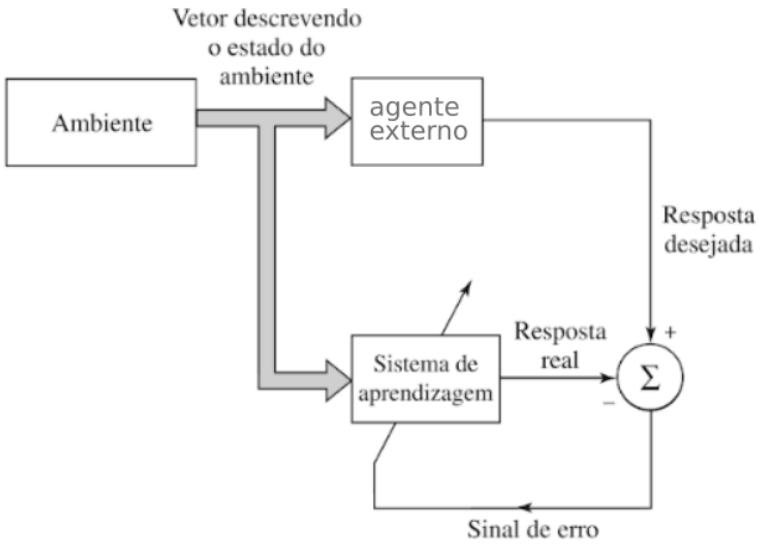
FONTE: Crespo (2017)

No processo de aprendizagem, parâmetros de uma rede neural são adaptados por meio de estímulos do ambiente no qual a rede está inserida. Existem muitos tipos de algoritmos de aprendizagem para redes neurais, estes algoritmos diferem entre si principalmente pelo modo como os pesos são modificados (HAYKIN, 2007).

No contexto de aprendizagem, pode-se destacar dois principais paradigmas que são eles: *Aprendizagem Supervisionada* e *Aprendizagem Não Supervisionada*.

Aprendizagem supervisionada consiste no processo de aprendizado onde os dados de treino passados na entrada possuem as informações de respostas. Os parâmetros dos algoritmos de aprendizado são então ajustados de acordo com os dados de treino e do *sinal de erro*. O sinal de erro é dado pela diferença entre a resposta desejada e a resposta do algoritmo. Assim, através de treinamento, o algoritmo aprende estes padrões e ao ser submetido a novos dados consegue prever as respostas com base nos padrões previamente encontrados (HAYKIN, 2007). A Figura 4 mostra um diagrama em blocos que ilustra esta forma de aprendizagem.

Figura 4 – Diagrama em blocos da aprendizagem supervisionada



FONTE: Adaptado de Haykin (2007)

Na aprendizagem não supervisionada não há um agente externo para supervisionar o processo de aprendizagem, como indicado na Figura 5. Neste processo, os dados de treino passados na entrada não possuem rótulos, ou seja, não possuem pré-definidamente os valores de resposta. Para a realização do processo são dadas condições para realizar uma *medida independente da tarefa* da qualidade da representação que a rede deve aprender, e baseado nesta medida, os parâmetros livres da rede são otimizados (HAYKIN, 2007).

Figura 5 – Diagrama em blocos da aprendizagem não supervisionada

FONTE: Haykin (2007)

Perceptron de Múltiplas Camadas (*Multilayer Perceptron*, MLP) é a denominação dada às redes neurais compostas de várias camadas de neurônios. Uma MLP consiste de um conjunto de unidades que constituem a *Camada de entrada*, onde ocorre a entrada dos valores e

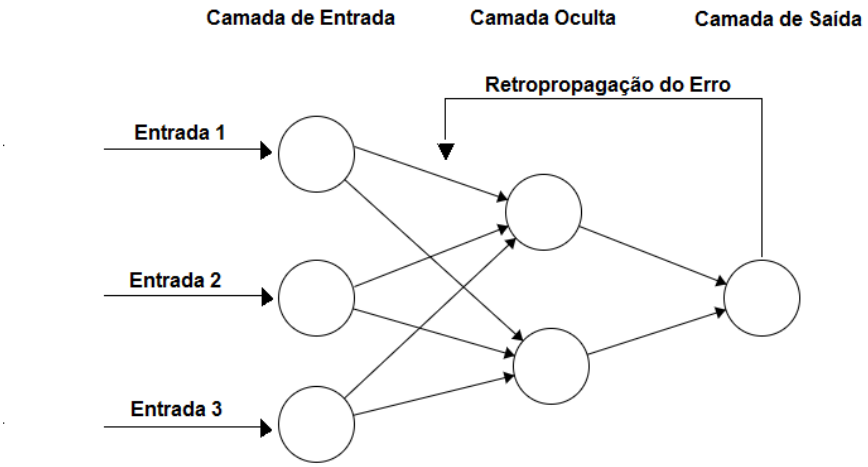
os repassa à rede, *Camadas intermediárias*, que são responsáveis por realizar o processamento dos valores de entrada e uma *Camada de saída*, onde o resultado proveniente das camadas anteriores é apresentado.

Em uma MLP, qualquer neurônio de uma camada pode estar ligado com qualquer neurônio da camada seguinte. Os *perceptrons* de múltiplas camadas têm sido aplicados com sucesso para resolver diversos problemas através do seu algoritimo de treinamento supervisionado *error backpropagation* (HAYKIN, 2007).

Neste algoritimo a aprendizagem consiste de dois passos por meio das camadas da rede: avançar na rede, a *propagação*, e retroceder na rede, a *retropropagação*. Na etapa de propagação, uma entrada é aplicada aos neurônios da rede e seu efeito se propaga ao longo da rede passando pelas suas camadas e então um conjunto de saídas é produzido com a resposta da rede. Nesta etapa os pesos da rede são mantidos *fixos*, isto é, sem alterações. Na etapa de retropropagação, os pesos são todos ajustados de acordo com uma regra de correção de erro. Um *sinal de erro* é obtido através da diferença entre o que é esperado pela rede e o que é obtido pelos neurônios de saída. Este sinal de erro é então propagado de volta na rede, ajustando os pesos atribuídos a cada neurônio para minimizar o erro e fazer com que a resposta da rede se aproxime da resposta desejada (HAYKIN, 2007). O ato de executar essas iterações, realizando os passos de propagação e retropropagação na rede, até processar uma quantidade de lotes suficiente para cobrir todo o conjunto de treino é chamado de *época*.

A Figura 6 ilustra a retropropagação de erro atualizando os pesos da rede.

Figura 6 – Atualização do pesos pelo *backpropagation*

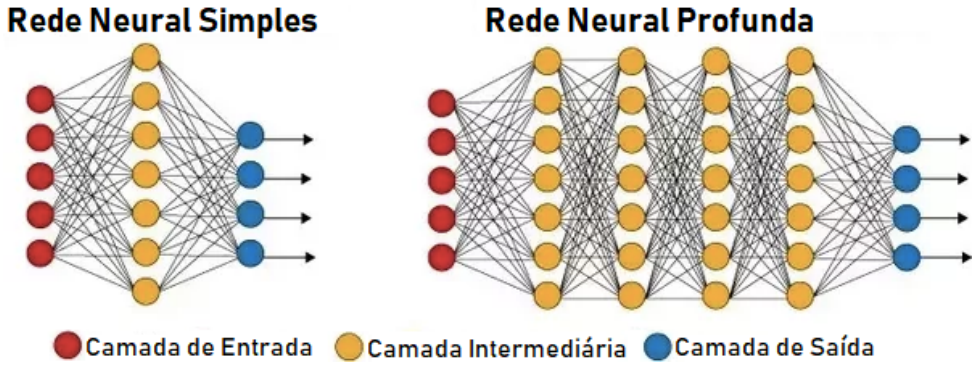


Fonte: Aguiar (2017)

Redes neurais profundas, usualmente denominadas como modelos de aprendizagem profunda estudam métodos que possibilitam ao computador construir conceitos complexos a partir de conceitos mais simples. O termo "profundo" se refere ao número de camadas intermediarias na rede neural, quanto mais camadas a rede tem, mais profunda e complexa ela é (CRESPO, 2017). A Figura 7 demostra a diferença entre uma rede neural simples e um rede neural profunda.

Uma perspectiva sobre aprendizagem profundo é que o conceito de profundidade permite que o computador aprenda determinada tarefa em várias etapas (GOODFELLOW *et al.*, 2016), ou seja, permite que modelos computacionais compostos por múltiplas camadas de processamentos aprendam representações de dados com múltiplos níveis de abstração (LECUN *et al.*, 2015).

Figura 7 – Diferença entre Rede Neural e Rede Neural Profunda. É possível notar uma maior quantidade de camadas intermediarias em uma Rede Neural Profunda



FONTE: Adaptado de Cenon Stack (2017)

Um conceito importante utilizado para a aprendizagem em redes neurais é o de *função de ativação*. Cada neurônio da rede inclui um função de ativação que é responsável por realizar uma transformação não linear nos dados de entrada, tornando a rede capaz de aprender e executar tarefas mais complexas.

Um exemplo de função de ativação é a função *sigmoide*, definida pela equação 2.1. Esta função converte entradas nos valores entre $-\infty$ e $+\infty$, cuja imagem é um intervalo de valores entre 0 e 1.

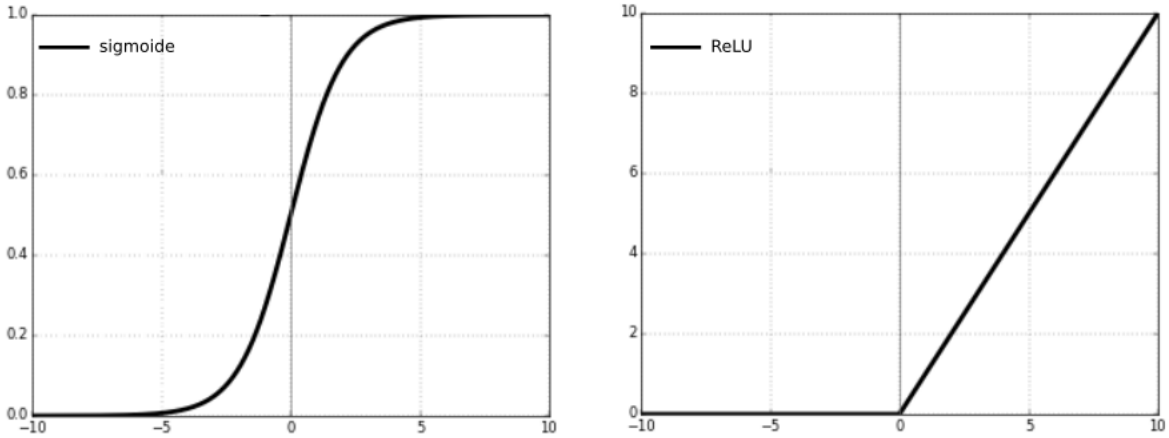
$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.1}$$

Uma outra função de ativação é a função *ReLU*, que é definida pela equação 2.2. A principal característica desta função é que ela não ativa todos os neurônios ao mesmo tempo, pois se a entrada de um neurônio for negativa, ela será convertida em zero e o neurônio não será ativado tornando a rede mais esparsa e eficiente para a computação.

$$ReLU(x) = max(0,x) \tag{2.2}$$

A Figura 8 ilustra os gráficos das duas funções de ativação citadas acima.

Figura 8 – Gráfico das funções Sigmoide e ReLU



Fonte: Adaptado de Sharma (2017)

A característica da *sigmoide* é que sua função produz uma curva em um formato de 'S' e que a sua saída sempre estará no intervalo (0,1). A função *sigmoide* é mais utilizada para classificação binária em modelos de regressão logística.

A vantagem da função *ReLU* é que a mesma é computacionalmente simples. A etapa de propagação e retropropagação são apenas uma simples instrução condicional. Essa é uma grande vantagem quando se lida com redes com muitos neurônios o pode reduzir significativamente o tempo necessário para o treinamento.

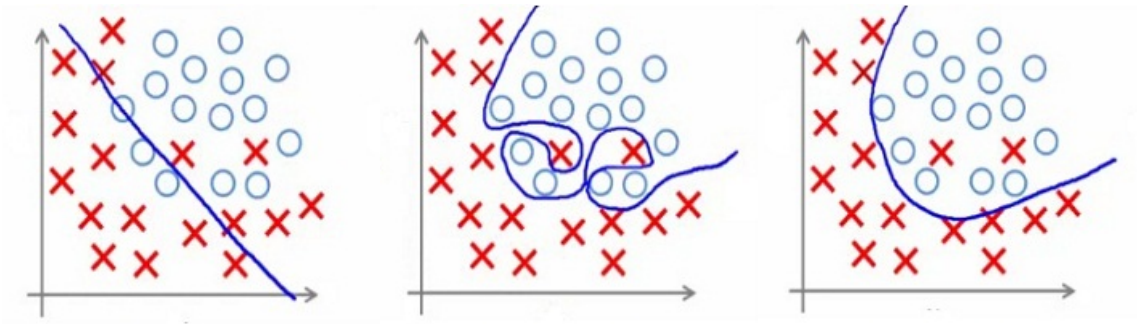
No processo de aprendizagem, é esperado que o erro no conjunto teste seja maior ou igual ao erro no conjunto de treino. Os fatores que determinam o quão bom é um algoritimo de aprendizagem são a sua capacidade de tornar baixo o erro no treinamento e tornar baixa a diferença entre o erro no treinamento e o erro no teste (GOODFELLOW *et al.*, 2016). Na aprendizagem, é esperado que o modelo se ajuste bem ao problema que lhe é passado, que seja generalizável e ao mesmo tempo específico em relação aos seus dados.

Quando um modelo não é capaz de obter um valor de erro suficientemente baixo no treinamento, é dito que aquele modelo possui um *underfitting*. Quando a diferença entre o erro no treinamento e o erro no teste é muito grande, é dito que aquele modelo possui um *overfitting*.

A Figura 9 mostra estes princípios de ajustes. A figura apresenta linhas que buscam separação entre os pontos. Se a linha não se ajusta adequadamente aos dados, como na Figura 9a, há um problema de *underfitting*. Se a linha se ajusta perfeitamente aos dados, como na Figura 9b, há um problema de *overfitting*. A Figura 9c mostra o que seria um ajuste apropriado ao traçar uma linha de aproximação nos dados.

Figura 9 – *Underfitting* e *Overfitting* em modelos de aprendizagem

(a) *Underfitting* (b) *Overfitting* (c) Ajuste apropriado



Fonte: Adaptado de Patel (2018)

Para impedir que um modelo aprenda padrões errôneos ou irrelevantes presentes nos dados de treinamento, uma solução é obter mais dados de treinamento, pois um modelo treinado com mais dados, naturalmente irá generalizar melhor. Mas quando não é possível obter mais dados, uma outra solução é utilizar a técnica da *regularização*, que consistem em adicionar restrições na complexidade de uma rede e penalizando seus pesos para obterem apenas valores pequenos, o que torna a distribuição de valores de peso mais regular (CHOLLET, 2017).

Regularização é qualquer modificação feita em um algoritmo de aprendizagem que se destina a reduzir seu erro de generalização (GOODFELLOW *et al.*, 2016). A regularização ajuda a resolver problemas de ajuste excessivo, estes ajustes excessivos fazem com que o modelo tenha um bom desempenho nos dados de treinamento, mas tenha um desempenho ruim nos dados de validação e teste.

O processo de regularização é chamado de regularização de peso (*weight regularization*) e é realizado adicionando à função de perda da rede um *custo* associado a grandes pesos. Este custo pode ser obtido através de duas técnicas comumente utilizadas: *regularização L1* e *regularização L2*.

Na regularização L1, o custo adicionado a um valor de perda (definido na seção 2.4) é proporcional a soma dos valores absolutos dos pesos (a norma l1 dos pesos).

$$C = perda + \lambda * \sum |w| \tag{2.3}$$

Na regularização L2, o custo adicionado é proporcional ao quadrado do valor dos

coeficientes de peso (a norma l2 dos pesos).

$$C = perda + \lambda * \sum w^2 \tag{2.4}$$

As equações 2.3 e 2.4 mostram as fórmulas para as regularizações L1 e L2, onde λ é o parâmetro de regularização que deve ser ajustado, pois sendo zero, não haverá mudanças e sendo um valor muito grande, no caso regularização L2, irá penalizar de maneira excessiva o modelo, causando *underfitting*.

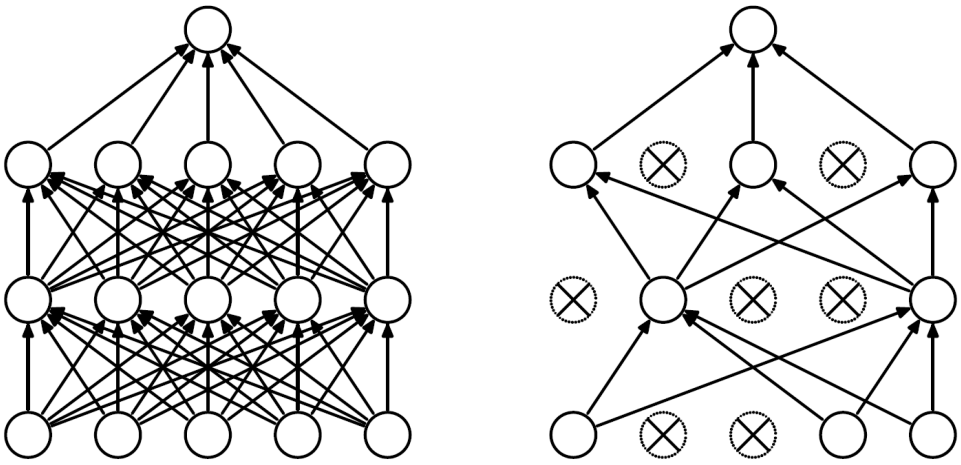
O efeito da regularização é fazer com que a rede prefira aprender pequenos pesos ou seja, ela pode ser vista como uma forma de compromisso entre encontrar pequenos pesos e minimizar a função de custo.

A regularização ajuda a reduzir o *overfitting* porque com o tamanho menor dos pesos, o comportamento da rede não mudará muito ao serem alteradas algumas entradas aleatórias. Isso previne que uma rede aprenda os efeitos de ruído local nos dados, descartando informações irrelevantes nos dados e aprendendo padrões vistos com frequência em todo o conjunto de treinamento.

Uma outra técnica bastante utilizada para prevenir o *overfitting* é a técnica de desligamento (*dropout*) (SRIVASTAVA *et al.*, 2014). O *dropout* aplicado a uma camada da rede consiste em, baseado em um determinado valor de probabilidade, aleatoriamente retirar alguns neurônios ou definir em zero a sua saída, fazendo com que estes neurônios não contribuam nem participem da propagação e retropropagação na etapa de aprendizagem.

A Figura 10 mostra o efeito da aplicação de *dropout* em uma rede. Na imagem a esquerda, a configuração original de uma rede neural padrão com 2 camadas ocultas. Na imagem a direita, a configuração após aplicação de *dropout* na rede à esquerda, unidades que se cruzavam foram descartadas.

Figura 10 – Efeito da aplicação do *dropout* em uma rede neural



Fonte: Srivastava *et al.* (2014)

Para a aplicação de *dropout* é necessário definir um hiperparâmetro, a probabilidade p, que controla a intensidade de *dropout*. Se $p = 1$ implica em nenhum neurônio de fora, e quanto menor o valor de p, maior a taxa de *dropout* (BEZERRA, 2016).

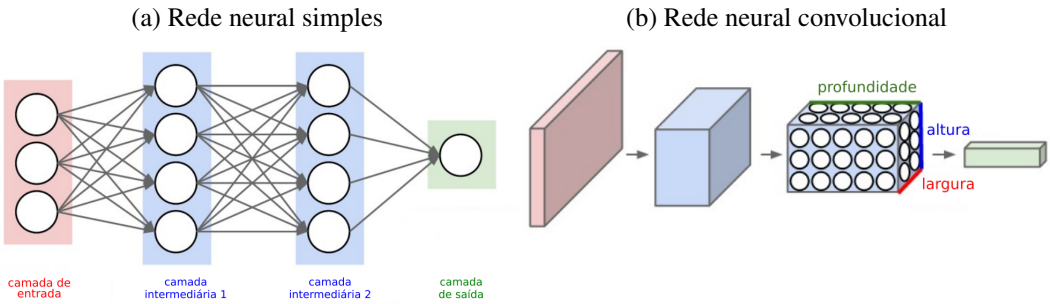
2.2.1 Redes Neurais Convolucionais

Redes Neurais Convolucionais (RNC) são um tipo de modelo de redes neurais profundas bastante utilizados e funcionam particularmente bem para problemas de visão computacional devido à sua capacidade de operar *convolucionalmente*, extraindo características de partes especificas de uma imagem e permitindo uma representação de maneira eficiente dos dados (CHOLLET, 2017).

A convolução opera sobre tensores de três dimensões, com duas dimensões (eixos) espaciais, altura e largura, bem como uma dimensão de profundidade (também chamado de canais) (CHOLLET, 2017). Tensores são matrizes multidimensionais. As redes convolucionais compreendem imagens como objetos tridimensionais, isso porque para uma imagem com codificação *RGB*, por exemplo, a dimensão do eixo de profundidade é 3, pois a imagem tem três canais de cor: vermelho, verde e azul.

A Figura 11 ilustra a organização dos neurônios nas redes neurais. A Figura 11a é uma rede neural simples de 2 camadas e a Figura 11b apresenta uma rede neural convolucional com seus neurônios organizados em três dimensões (largura, altura, profundidade).

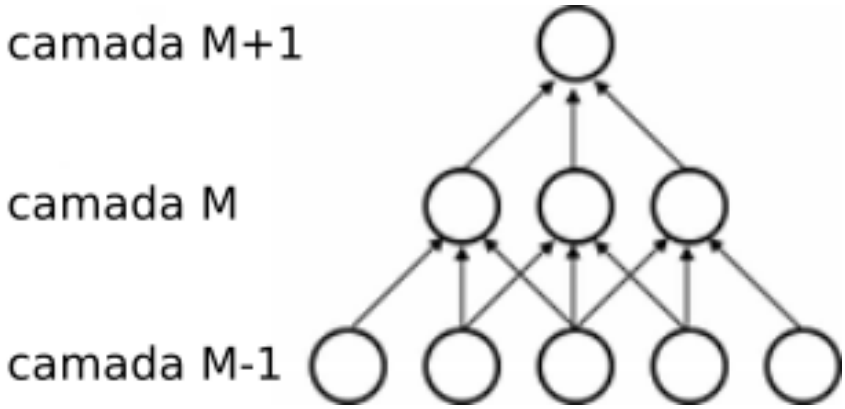
Figura 11 – Disposição dos neurônios em uma rede neural simples e em uma rede neural convolucional



Fonte: Adaptado de Karpathy (2015)

Em uma rede neural simples, como na Figura 3, cada neurônio na camada de entrada está conectada a todas as unidades da camada intermediária. Já uma RNC utiliza uma conectividade local, como na Figura 12, onde cada unidade na camada M-1 está conectada a uma quantidade limitada de unidades da camada M, ao invés de se conectar a todas as unidades da camada. Sendo assim há menos pesos para serem atualizados o que facilita o treinamento (ARAÚJO *et al.*, 2017).

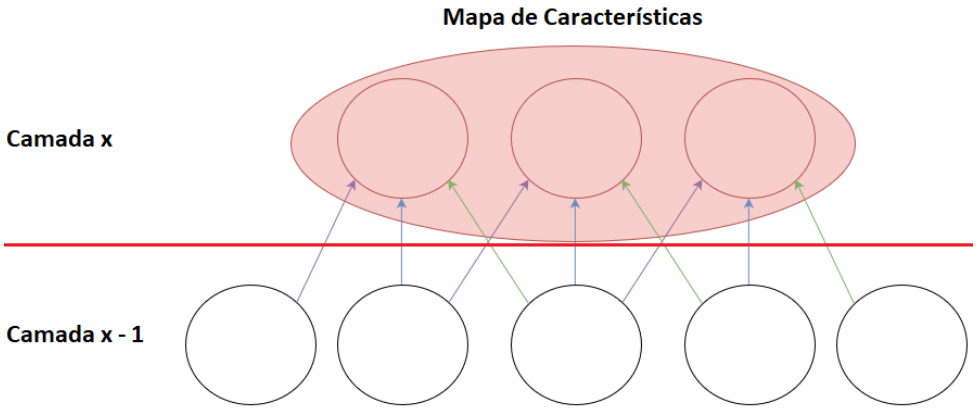
Figura 12 – Conexões entre camadas em uma RNC



Fonte: Adaptado de El-Sayed *et al.* (2013)

O campo formado pelas conexões de uma RNC, como ilustrado na Figura 13, pode, por exemplo, detectar características visuais relevantes como arestas, extremidades e bordas. Essas características vão sendo combinadas com as camadas seguintes para detecção de características mais complexas. Unidades da mesma camada são agrupados e formam os mapas de características.

Figura 13 – Mapa de características em uma RNC



Fonte: Crespo (2017)

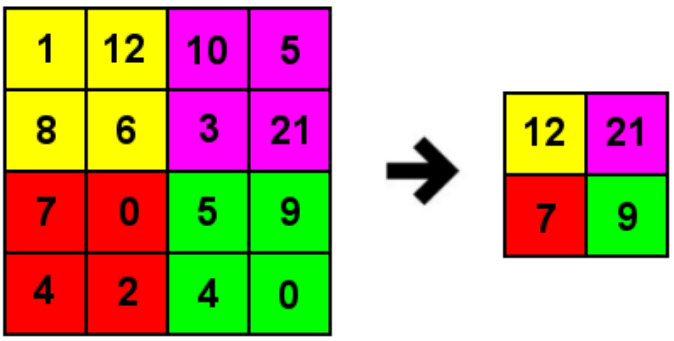
As camadas convolucionais consistem assim de um conjunto de filtros que se estendem por toda a profundidade da entrada. Durante o treinamento da rede, estes filtros são ajustados para que sejam ativados na presença de características relevantes identificadas na entrada (ARAÚJO *et al.*, 2017).

Um conceito importante em RNC é a camada de *pooling*, responsável por dividir o mapa de características de entrada em um conjunto de retângulos que não se sobrepõem e para cada sub-região da imagem, tem como saída um resultado de acordo com a métrica utilizada (CRESPO, 2017). O objetivo da função de *pooling* é reduzir de forma progressiva a dimensão do volume de entrada.

Há várias formas de *pooling* que podem ser aplicados, como selecionar o valor máximo (*max-pooling*) ou a média (*average pooling*). A operação de *max-pooling*, por exemplo, consiste em substituir os valores de uma região pelo valor máximo da mesma.

A Figura 14 mostra o resultado da aplicação de *max-pooling*, é possível notar que ocorreu uma diminuição no tamanho da entrada o que reduz o tamanho do mapa de características a ser processado, reduzindo os processamentos das próximas camadas (CHOLLET, 2017).

Figura 14 – Aplicação de max-pooling em uma imagem 4x4 utilizando um filtro 2x2

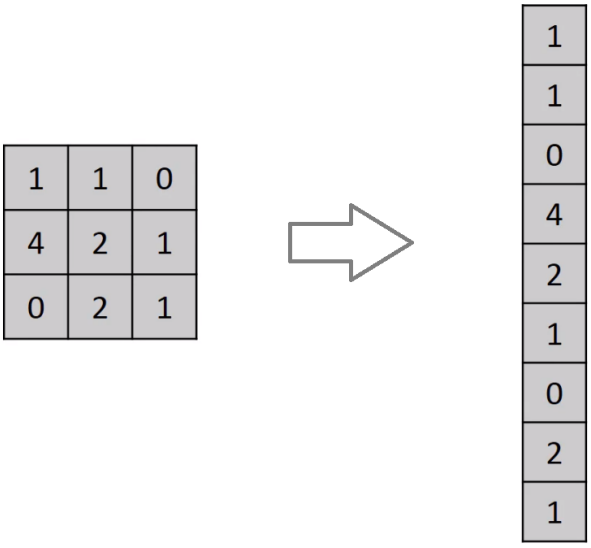


Fonte: Ferreira (2017)

Um outro tipo de camada presente em uma rede neural, são as camadas Densas (*Dense*) também chamadas de camadas totalmente conectadas (*fully connected layer*). Nessa camada cada neurônio está conectado a todos os neurônios da camada subsequente.

Além da camada densa, há também a camada *Flatten* que permite alterar a forma dos dados de determinada dimensão para o formato correto para uma camada densa a ser interpretada. Esta camada simplesmente permite que os dados sejam operados de uma forma diferente pois realiza uma remodelagem na forma (*shape*) dos dados de entrada. A Figura 15 mostra o resultado da aplicação da camada *Flatten* dada uma entrada.

Figura 15 – Aplicação da camada *Flatten*



Fonte: Živković (2018)

2.3 TRANSFERÊNCIA DE CONHECIMENTO

Modelos de aprendizagem profunda necessitam de uma grande quantidade de dados para serem treinados, o que dificulta a realização desta tarefa pois nem sempre essa grande quantidade de dados se encontra disponível ou acessível. Além disso, o custo, sobretudo de tempo e processamento, para se treinar um modelo desde o início é muito elevado.

A técnica de transferência de conhecimento (*knowledge transfer*) é utilizada com o objetivo de reusar parte do conhecimento já adquirido por um modelo, aproveitando-o com o intuito de resolver novas tarefas, reduzindo assim o tempo necessário para treinar um novo modelo ou uma rede neural de aprendizado profundo.

A transferência de conhecimento visa reduzir o número de parâmetros a serem aprendidos, através da reutilização de informações aprendidas por um modelo anterior em que essas informações são consideradas adequadas para uma nova tarefa (CARVALHO *et al.*, 2015).

Na tarefa de classificação de imagens, a transferência de conhecimento se dá por meio de vetores de características gerados por modelos já treinados com seus pesos configurados para reconhecer e extrair características de um determinado conjunto de dados. Assim, após submeter dados de imagens a um modelo já treinado com certos pesos, será gerado por ele vetores de características que podem ser passados a camadas seguintes da arquitetura ou a um novo modelo (AGUIAR, 2017).

Aguiar (2017) exemplifica a transferência de conhecimento como uma pessoa *A* que vê um objeto e o descreve para uma pessoa *B* que não está vendo o objeto. A pessoa *A* descreve os elementos presentes no objeto e a partir das características ditas, a pessoa *B* irá traçar um perfil dos elementos que ela conhece e das características que ela ouviu e associar a um objeto que ela conhece e que possua características semelhantes.

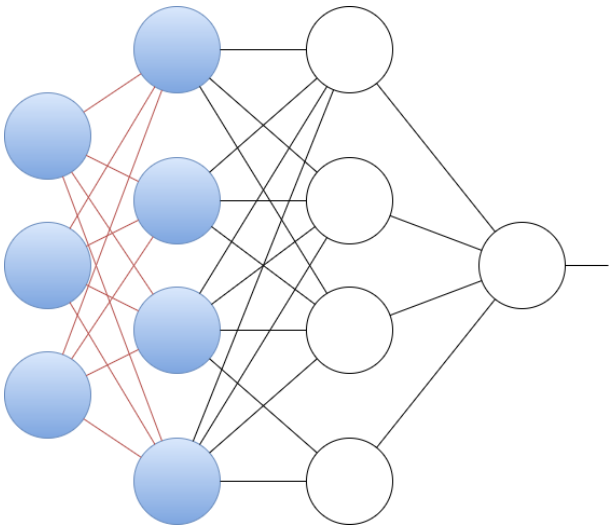
Um conceito similar ao de transferência de conhecimento é o de *ajuste fino* (*fine-tuning*). Ambos os conceitos vêm sendo amplamente utilizados, quase de forma intercambiável. A técnica de *fine-tuning* consiste em ajustar parâmetros de uma rede treinada para se adaptar a uma nova tarefa. Utiliza uma modelo já treinado com alguns dados e realiza um re-treino do modelo ou de parte do modelo em um novo conjunto de dados.

A técnica de *fine-tuning* consiste em *desafixar* ou liberar os pesos de algumas camadas de uma rede neural já treinada, para que seja treinado em conjunto com outras camadas recém adicionadas. Esta técnica ajusta as representações mais abstratas do modelo que está sendo reutilizado, a fim de torná-las mais relevantes para um novo problema em questão (CHOLLET,

2017).

A Figura 16 representa a arquitetura de uma rede neural e cada aresta entre os neurônios representa um peso. A parte composta por linhas vermelhas e neurônios azuis representa uma porção com os pesos *congelados* na rede já treinada, ou seja, seus pesos continuarão com os valores fixos. As linhas e neurônios não coloridos representam uma porção com os pesos *descongelados* ou seja, liberados na rede, que podem ser re-treinados.

Figura 16 – Arquitetura de uma rede neural com *fine-tuning*

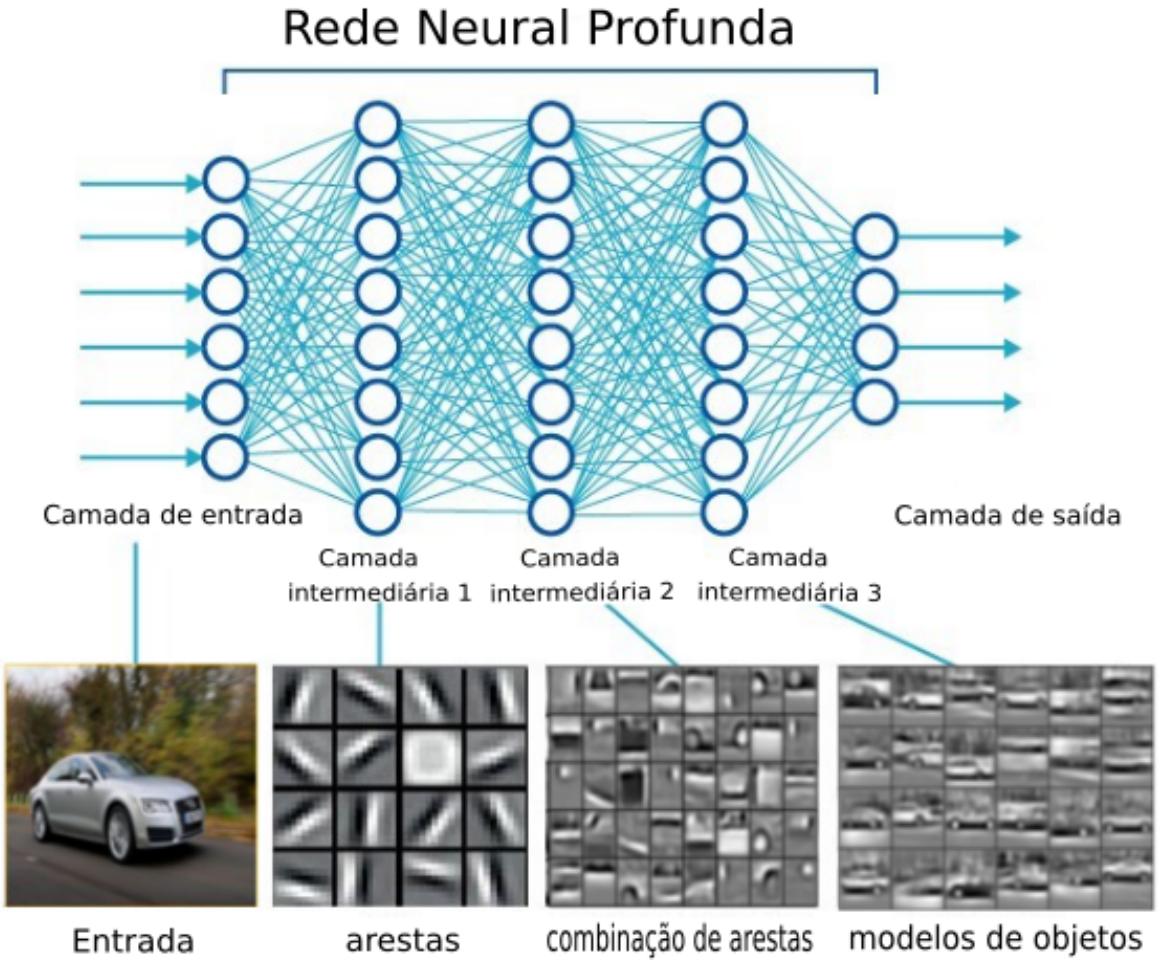


Fonte: Kishi (2017)

Em uma rede neural, as camadas iniciais aprendem características muitos gerais e, a medida que se avança na rede, as camadas tendem a aprender características mais complexas e padrões mais específicos para a tarefa em que está sendo treinada. Em geral, o *fine-tuning* é utilizado aproveitando-se dessas camadas iniciais, mantendo-as congeladas, e treinando somente as camadas posteriores para a tarefa em especifico.

A Figura 17 mostra uma representação gráfica de um modelo de aprendizagem profunda para o reconhecimento de imagem. No exemplo, a entrada é uma imagem e os neurônios progressivamente aprendem características mais complexas até que saia um sinal indicando o resultado.

Figura 17 – Cada camada da rede neural profunda reconhece arestas, formas geométricas e objetos complexos, dependendo da camada. Neste exemplo, um carro foi detectado.



Fonte: Adaptado de Lee *et al.* (2011)

O *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) (RUSSAKOVSKY *et al.*, 2015) foi um dos desafios precursores que permitiu a rápida evolução das redes neurais profundas (AGUIAR, 2017). Com o objetivo de obter bons resultados para este desafio, inúmeros estudos foram feitos e diversas técnicas de aprendizagem profunda foram investigadas. O desafio do *ImageNet* possui três aspectos de avaliação, detecção de objetos, localização de objetos e classificação de objetos em imagens. Sua base de dados possui milhões de imagens divididas em mil classes.

As arquiteturas de redes neurais profundas VGG-16 e VGG-19 (SIMONYAN; ZISSERMAN, 2014) obtiveram bons resultados para o desafio do *ImageNet* no ano de 2014, apresentando bom desempenho nas tarefas de localização e classificação de objetos.

Buscando trazer melhorias para a tarefa de reconhecimento de imagens, o Google

propôs a rede InceptionV3 (SZEGEDY *et al.*, 2016), uma arquitetura voltada para a resolução de problemas de visão computacional (AGUIAR, 2017). Esta arquitetura obteve um bom desempenho no desafio do *ImageNet* no ano de 2015. Em comparação com as redes VGG-16 a rede InceptionV3 possui menos parâmetros estimados pela rede, possuindo um melhor desempenho computacional.

Neste trabalho é feito o uso das arquiteturas VGG-16, por ser um modelo sequencial e mais simples de entender e visualizar, e InceptionV3 por apresentar uma melhor performance. Ambas as arquiteturas são carregadas com seus pesos pré-treinados para reconhecimento do conjunto de dados do *ImageNet* para a transferência de conhecimento e em seguida descongelados para a técnica de *fine-tuning*.

2.4 AVALIAÇÃO DE DESEMPENHO

As métricas para classificação são uma forma de avaliar se o modelo de classificação está adequado ao que foi proposto. Para mensurar a qualidade da classificação, utiliza-se o conceito de matriz de confusão como a representada na Tabela 1, que oferece uma medida efetiva do modelo de classificação ao exibir o número de classificações corretas e as classificações preditas para cada classe (MONARD; BARANAUSKAS, 2003).

Tabela 1 – Matriz de confusão para a classificação

		Classe Real y	
		Positivo	Negativo
h0(x) Resultado Previsto	Resultado positivo previsto	Verdadeiro Positivo (VP)	Falso Positivo (FP)
	Resultado negativo previsto	Falso Negativo (FN)	Verdadeiro Negativo (VN)

Fonte: Elaborado pleo autor.

Uma métrica bastante utilizada é a acurácia (*accuracy*), que consiste na proporção do que foi predito corretamente, seja positivo ou negativo, sobre o total de exemplos. É idealmente usada quando não há uma grande variação na proporção entre exemplos de cada classe no conjunto de dados. A acurácia indica o quão frequente o classificador está correto.

A fórmula da acurácia é representada pela equação 2.5.

$$acc = \frac{(VP + VN)}{Total} \tag{2.5}$$

Suponha um conjunto de 27 imagens de cães, gatos e coelhos onde os rótulos reais são representados pelo vetor *y* e os rótulos preditos por um classificador são representados pelo vetor *y_pred*.

- *y* = ['gato', 'gato', 'gato', 'gato', 'gato', 'gato', 'gato', 'gato', 'cão', 'cão', 'cão', 'cão', 'cão', 'cão','coelho', 'coelho', 'coelho', 'coelho', 'coelho', 'coelho', 'coelho', 'coelho', 'coelho', 'coelho', 'coelho', 'coelho']
- *y_pred* = ['gato', 'gato', 'gato', 'gato', 'gato', 'gato', 'gato', 'gato', 'gato', 'gato', 'cão', 'cão', 'cão', 'coelho', 'cão', 'cão', 'coelho', 'coelho', 'coelho', 'coelho', 'coelho', 'coelho', 'coelho', 'coelho', 'coelho', 'coelho', 'coelho']

A Tabela 2 mostra a matriz de confusão gerada para a classificação acima, onde a classe real é o *y* e a classe predita é o *y_pred*.

Tabela 2 – Matriz de confusão da classificação de cães, gatos e coelhos

		Classe Real		
		gato	**cão**	**coelho**
Classe Predita	**gato**	5	2	0
	cão	3	3	2
	coelho	0	1	11

Fonte: Elaborado pelo autor.

O valor de acurácia dessa classificação é $(5+3+11)/27 = 0.7037037037$. Ou seja, esse classificador obteve uma acurácia no valor de aproximadamente 70%.

O **coeficiente de perda** (*loss*) é a medida do quão longe as predições de um modelo são de seu rótulo real, ou seja, o valor de perda implica em quão bem ou quão mal o modelo se comporta após cada iteração de treinamento. O principal objetivo em um modelo de aprendizagem é minimizar o valor da função de perda, que aumenta à medida que a probabilidade prevista diverge do rótulo real.

Há muitos tipos e diferentes implementações de funções de perda, porém todas possuem o mesmo objetivo. A equação 2.6 representa a fórmula para a função de perda *Log-loss* que avalia o desempenho de um modelo de classificação medindo a similaridade entre distribuições de probabilidades entre uma classe predita e uma classe real.

Para calcular o *Log-loss*, o classificador deve atribuir uma probabilidade a cada classe, em vez de simplesmente fornecer a classe mais provável.

$$loss = \frac{-1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{i,j} * \log(P_{i,j}) \tag{2.6}$$

- N representa o número de amostras
- M o representa a quantidade de classes
- log representa a função logarítmica
- $y_{i,j}$ indica se a amostra i pertence a classe j
- $P_{i,j}$ indica a probabilidade da amostra i pertencer à classe j

A função *Log-loss*, funciona penalizando as classificações falsas e funciona bem para classificação de multi-classes. Um valor de *Log-loss* próximo a zero indica uma alta acurácia no conjunto de treino. No geral, minimizar o seu valor, fornece uma maior acurácia para o classificador.

3 TRABALHOS RELACIONADOS

Neste Capítulo são apresentados os trabalhos relacionados que foram utilizados como base para o presente trabalho.

O trabalho de Aguiar (2017) cita sobre o desenvolvimento de diversos trabalhos em sistemas de auxílio ao diagnóstico médico, que de maneira geral fornecem opiniões extraídas a partir de imagens médicas. O autor faz uma descrição e análise da aplicabilidade dos conceitos abordados em seu trabalho, tais como visão computacional, redes neurais profundas e a técnica de transferência de conhecimento.

Aguiar (2017) buscou realizar a classificação de nódulos em exames histopatológicos através de imagens da base de dados *BreaKhis* (SPANHOL *et al.*, 2016). O autor destaca o uso da técnica de transferência de conhecimento como alternativa para auxiliar no treinamento de redes neurais profundas e cita que por meio desta técnica sobre sua base de dados, pôde obter resultados promissores que podem ser bem explorados a fim de trazer melhorias para a classificação destes dados.

De maneira semelhante, no trabalho de Silva *et al.* (2018), os autores propuseram experimentos utilizando Redes Convolucionais para classificação e diagnóstico precoce da doença de Alzheimer a partir de imagens volumétricas de ressonância magnética da base de dados MIRIAD (*Minimal Interval Resonance Imaging in Alzheimer's Disease*). Os autores definiram uma arquitetura de rede neural convolucional com 4 camadas convolucionais e analisaram os resultados sobre os dados realizando dez experimentos com diferentes números de *épocas*.

Como em Silva *et al.* (2018) este trabalho faz o uso de redes neurais convolucionais, porém são utilizadas redes já conhecidas e utilizadas no meio acadêmico como a VGG-16 e InceptionV3. Assim como no estudo de Aguiar (2017), neste trabalho são utilizados conceitos de visão computacional, além da técnica de transferência de conhecimento com o objetivo reduzir o tempo necessário para treinar as redes neurais para classificar as imagens de nossa base de dados.

Em outro estudo envolvendo redes neurais para classificação de imagens, o trabalho de Ferreira (2017) tem o objetivo de detectar ervas daninhas em imagens de lavouras de soja e classificar essas ervas daninhas entre gramíneas e folhas largas, com o propósito de direcionar um herbicida específico ao tipo de erva daninha detectado.

Os autores construíram a base de dados a partir de imagens de uma plantação de soja, ao todo foram mais de quinze mil imagens do solo, soja e ervas daninhas. Os autores utilizaram redes neurais convolucionais e a treinaram para a base de dados, onde obtiveram uma precisão

de cerca de 98%.

Em todos os trabalhos relacionados citados neste Capítulo, é importante destacar o quão generalizável é esta área de aprendizado profundo, podendo ser utilizada em diferentes contextos e situações solucionando diferentes problemas.

4 CLASSIFICAÇÃO DE PERSONAGENS ANIMADO COM APRENDIZADO PROFUNDO

Neste Capítulo são apresentados os passos de implementação realizados neste trabalho, assim como informações sobre a base de dados utilizada, os pré-processamentos necessários. São apresentados também as tecnologias utilizadas e os experimentos que foram realizados.

Os códigos foram escritos utilizando a linguagem Python e documentados através da aplicação *jupyter notebook* [1]. Todos os procedimentos e implementações aqui descritos estão disponíveis para reprodução[2].

4.1 CONJUNTO DE DADOS E PRÉ-PROCESSAMENTO

A base de dados utilizada foi a base *The Simpsons Characters Data* disponível através da plataforma *Kaggle*[3]. A base consiste em imagens de personagens da série animada Os Simpsons (*The Simpsons*) da emissora *Fox Broadcasting Company*. A base de dados é composta por dois conjuntos: treino e teste. O conjunto de treino é formado por 20.933 imagens divididas não uniformemente em 42 classes que representam os personagens. E o conjunto de teste contém 938 imagens. A Figura 18 contém exemplos de imagens presentes na base de dados.

Figura 18 – Imagens de personagens do desenho Os Simpsons que estão presentes na base de dados

(a) Lisa Simpson

(b) Milhouse Van Houten

(c) Homer Simpson

(d) Nelson Muntz

Fonte: Elaborado pelo autor

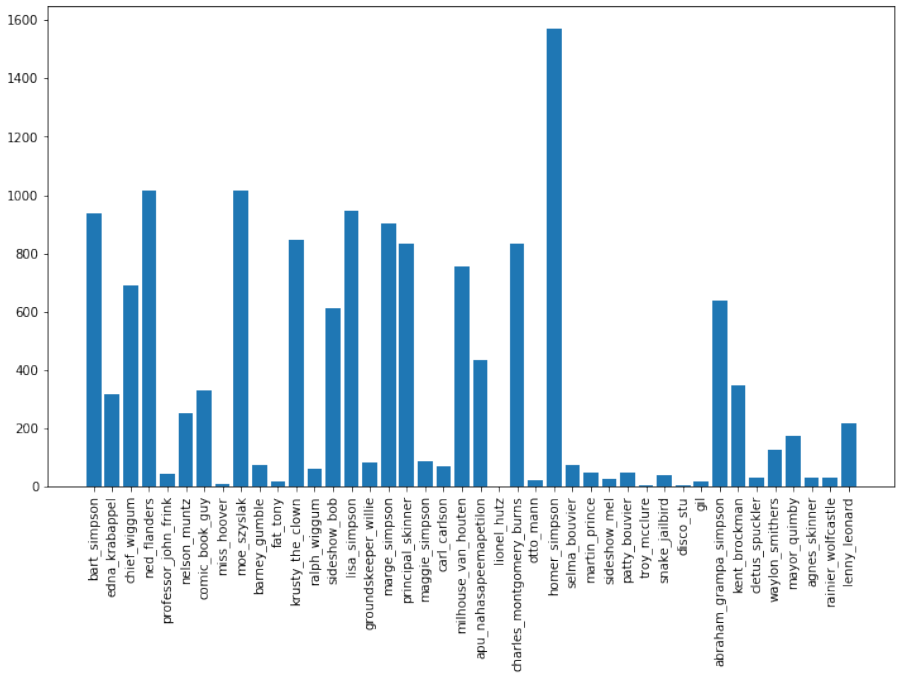
Como etapa de pré-processamento, o conjunto de treino original foi dividido em duas parte, um conjunto de treino e um conjunto de validação, na proporção de 70% para treino e 30% para validação. Em valores reais, o conjunto de treino passou a consistir de 14.633 imagens e o conjunto de validação de 6.300 imagens.

[1] http://jupyter.org/
[2] https://github.com/leosegundo/classification_simpsonsCharacters
[3] https://www.kaggle.com/alexattia/the-simpsons-characters-dataset

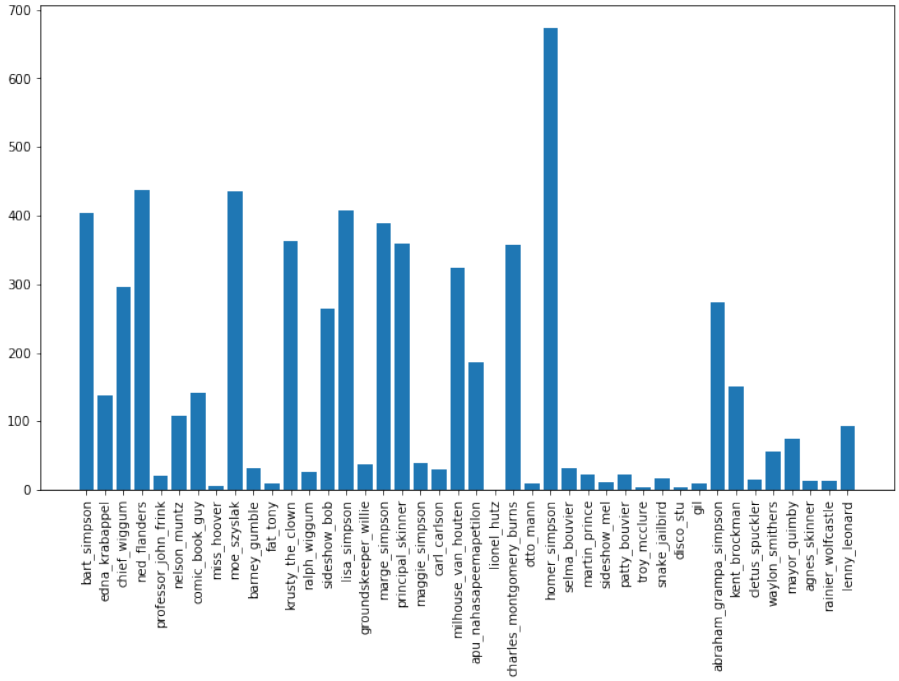
A Figura 19 mostra a distribuição das imagens em cada classe nos conjuntos de treino 19a e validação 19b.

Figura 19 – Histograma de distribuição das classes no conjunto de treino e validação

(a) Conjunto de treino



(b) Conjunto de validação



Fonte: Elaborado pelo autor

4.2 CONSTRUÇÃO DO CLASSIFICADOR

Após a divisão dos dados entre treino e validação, para a realização dos experimentos optou-se por utilizar duas arquiteturas de Redes Neurais profundas para verificar o desempenho destas sobre a base de dados proposta.

As arquiteturas utilizadas foram a VGG-16, por ser um modelo sequencial e simples mas bastante popular e utilizado em competições da plataforma *Kaggle*[4], e a InceptionV3, pois trabalhos como o de Canziani *et al.* (2016) apresentam esta arquitetura como uma das melhores em termos de acurácia e consumo de recursos computacionais.

Foi utilizado a biblioteca *Keras*[5] para realizar o carregamento das imagens contidas na base de dados e gerar os rótulos das imagens conforme cada classe a qual ela pertence.

Para a construção do classificador, também por meio da biblioteca *Keras* foram carregadas os modelos das arquiteturas VGG-16 e InceptionV3, com seus pesos treinados para reconhecer o conjunto *Imagenet*. A Figura 20 mostra o carregamento da arquitetura VGG-16 treinada com pesos do conjunto *Imagenet*, através desta biblioteca.

Figura 20 – Exemplo de carregamento da arquitetura VGG-16 utilizando a bilioteca *Keras*

```
from keras import applications
model = applications.VGG16(include_top=False,input_shape=(height,width,3), weights="imagenet")
```

Fonte: Elaborado pelo autor

Após o carregamento dos modelos, foi adicionado uma camada de classificação, uma camada totalmente conectada com 42 neurônios, que representa a quantidade de classes, e foi atribuída a esta camada uma função de ativação *softmax*, isto representa que a saída da rede será uma distribuição de probabilidades sobre as 42 classes possíveis. A Figura 21 ilustra como uma camada densa com uma função de ativação *softmax* é adicionada ao modelo.

Figura 21 – Adicionando uma camada densa ao modelo

```
from keras.layers import Dense
model.add(Dense(42, activation="softmax"))
```

Fonte: Elaborado pelo autor

Todos procedimentos aqui descritos foram realizados em uma máquina virtual na

[4] https://www.kaggle.com/
[5] https://keras.io/

plataforma *Kaggle* cujas especificações estão relatadas na Tabela 3.

Tabela 3 – Especificações da máquina utilizada nos experimentos

Processador	Intel Xeon 2.30GHz
Memória RAM	15G
Sistema Operacional	Linux 4.9.0-5-amd64 #1 SMP Debian 4.9.65-3+deb9u2
Placa de vídeo	NVIDIA-SMI 390.25, Tesla K80
Memória	11441MiB

Fonte: Elaborado pelo autor

4.3 EXPERIMENTOS

Os experimentos realizados com os modelos (InceptionV3 e VGG-16) foram divididos em seis subseções que foram chamadas de **Cenário de Experimentação**, onde a proposta destes cenários é avaliar a performance das arquitetura e descrever a otimização de hiperparâmetros para melhorar a performance das redes em cada caso.

Ambas as arquiteturas foram carregadas com os pesos pré-treinados do conjunto *ImageNet*, sem considerar contudo sua última camada, que é responsável por classificar as 1000 classes do conjunto.

Para os experimentos foram fixados valores padrões, segundo Chollet (2017), para os parâmetros de regularização, *dropout* e taxa de aprendizagem (*learnig rate*) para ambas as arquiteturas.

4.3.1 Cenário de Experimentação I

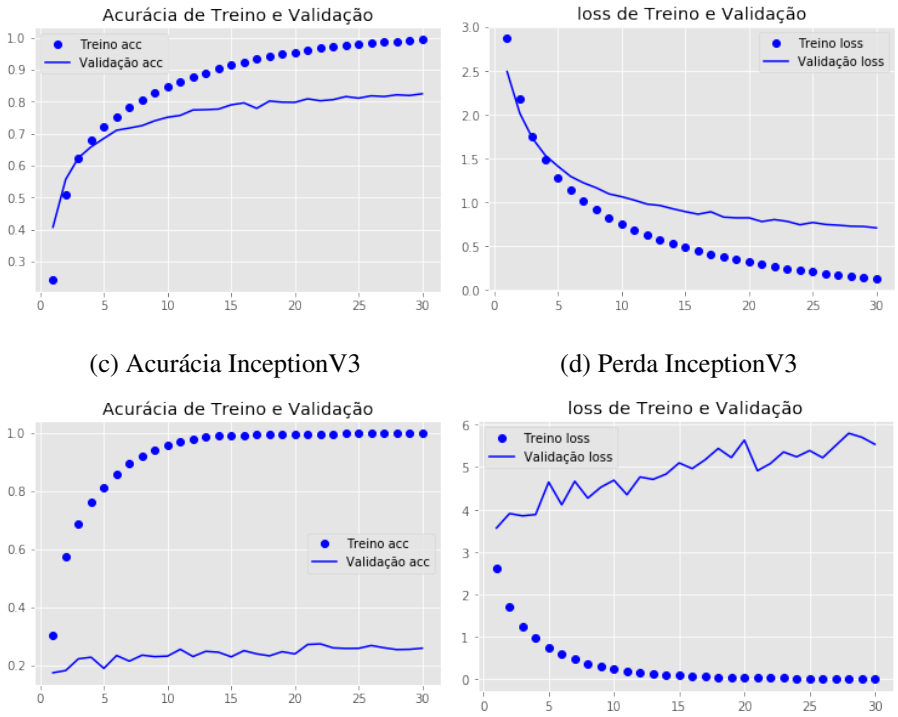
O objetivo deste primeiro cenário de experimentação foi analisar a performance das arquiteturas VGG-16 e InceptionV3 sem adição de nenhuma camada extra e sem aplicação de nenhuma técnica adicional, ao longo de 30 épocas.

As Figuras 22a e 22b apresentam a acurácia e perda da VGG-16 e as Figuras 22c e 22d apresentam a acurácia e perda da InceptionV3. Na Figura 22 é possível notar que ambas arquiteturas apresentaram *overfitting* que é indicado pela diferença entre os valores de acurácia e perda durante o treino e validação.

A Tabela 4 apresenta o resultado ao final deste cenário de experimentação.

Figura 22 – VGG-16 e InceptionV3

(a) Acurácia VGG-16 (b) Perda VGG-16



(c) Acurácia InceptionV3 (d) Perda InceptionV3

Fonte: Elaborado pelo autor

Tabela 4 – Resultado cenário de experimentação I

Arquitetura	Acurácia	Perda
VGG-16	82.4%	0.7
InceptionV3	25.9%	5.5

Fonte: Elaborado pelo autor

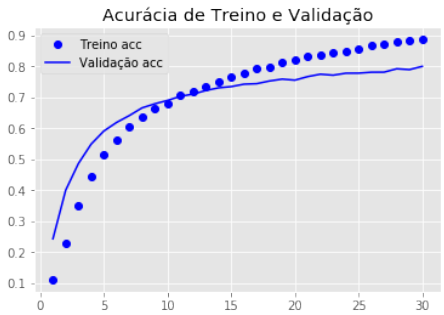
4.3.2 Cenário de Experimentação II

Com o intuito de prevenir e reduzir o *overfitting*, foi aplicado uma camada de *dropout* com um valor de probabilidade de 50%, que é um valor padrão citado por Chollet (2017). O objetivo deste cenário de experimentação foi o de analisar os resultados de ambas as arquiteturas após a aplicação desta camada, ao longo de 30 épocas.

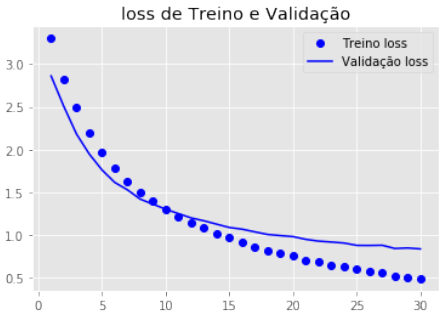
As Figuras 23a e 23b apresentam a acurácia e perda da VGG-16, e as Figuras 23c e 23d apresentam a acurácia e perda da InceptionV3. Note que somente na arquitetura VGG-16 é que foi possível notar uma redução considerável no *overfitting*. A Tabela 5 apresenta o resultado ao final deste cenário de experimentação.

Figura 23 – VGG-16 e InceptionV3 - *dropout*

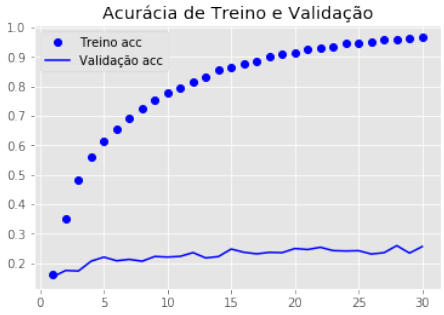
(a) Acurácia VGG-16



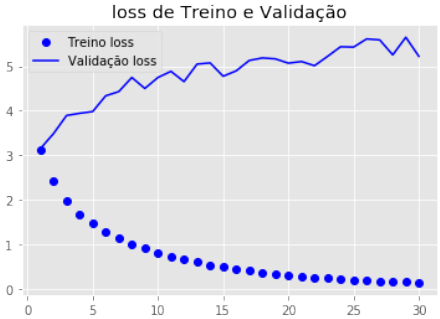
(b) Perda VGG-16



(c) Acurácia InceptionV3



(d) Perda InceptionV3



Fonte: Elaborado pelo autor

Tabela 5 – Resultado cenário de experimentação II

Arquitetura	Acurácia	Perda
VGG-16	79.9%	0.83
InceptionV3	25.6%	5.2

Fonte: Elaborado pelo autor

4.3.3 Cenário de Experimentação III

O objetivo deste cenário de experimentação foi o de verificar o efeito causado na performance das arquiteturas após adição de mais uma camada, a camada de regularização, ao longo de 60 épocas.

Esse aumento na quantidade de épocas de 30 para 60 se deu a fim de que as redes convergissem, como pode ser observado na Figura 24. Vale ressaltar que os valores definidos para regularização L1 e L2 foram de $1e-3$ (0.001), que é um valor padrão utilizado por Chollet (2017).

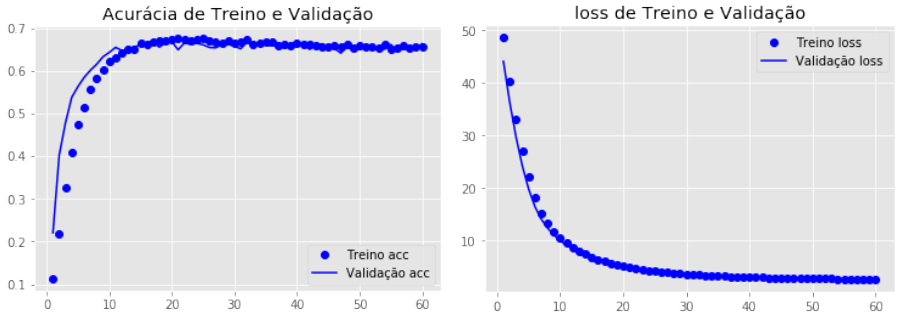
As Figuras 24a e 24b apresentam a acurácia e perda da VGG-16 e as Figuras 24c

e 24d apresentam a acurácia e perda da InceptionV3. A arquitetura InceptionV3 continua a apresentar *overfitting* acentuado enquanto a VGG-16 parece convergir, se sobressaindo bem para a base de dados proposta.

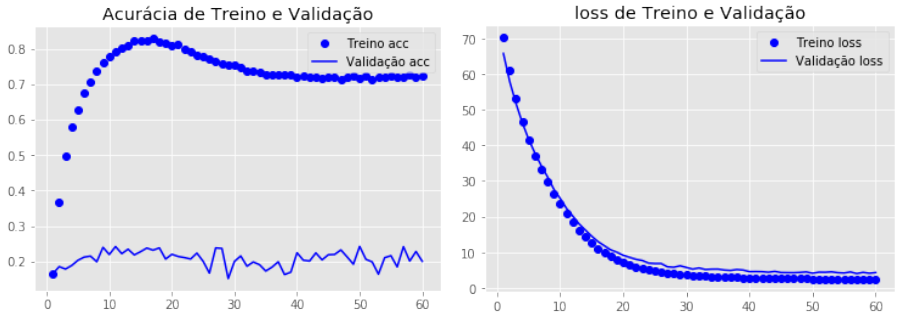
A Tabela 6 apresenta o resultado ao final deste cenário de experimentação, é importante frisar que apesar dos valores de acurácia e perda para a rede VGG-16 terem piorados em relação ao cenário anterior, ocorreu uma diminuição no *overfitting*, podendo ser notado pelo fato de que os valores de perda, para o treino e validação da rede, convergiram para um valor próximo.

Figura 24 – VGG-16 e InceptionV3 - regularização

(a) Acurácia VGG-16 (b) Perda VGG-16



(c) Acurácia InceptionV3 (d) Perda InceptionV3



Fonte: Elaborado pelo autor

Tabela 6 – Resultados do cenário de experimentação III

Arquitetura	Acurácia	Perda
VGG-16	66.3%	2.5
InceptionV3	20.0%	4.3

Fonte: Elaborado pelo autor

4.3.4 Cenário de Experimentação IV

O objetivo deste cenário de experimentação foi o de avaliar o impacto nos resultados após aplicação da técnica *fine-tuning*.

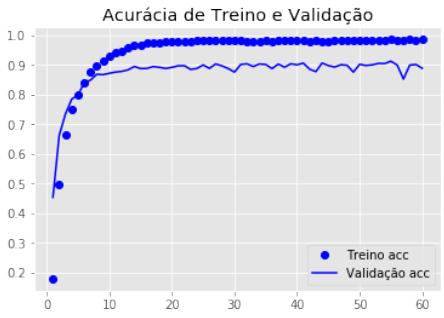
Assim, para a rede VGG-16, foi mantido fixos todos os pesos até os do penúltimo bloco de camadas, os pesos do último bloco de camadas foram então descongelados, como recomendado por Chollet (2017). Para a InceptionV3 foram mantidos os pesos fixos das primeiras 249 camadas e o restante foram descongeladas, como referenciada na documentação da API Keras (CHOLLET, 2015).

As Figuras 25a e 25b apresentam os gráficos de a acurácia e perda da VGG-16 e as Figuras 25c e 25d apresentam os gráficos de a acurácia e perda da InceptionV3. A Tabela 7 apresenta os resultados ao final deste cenário de experimentação.

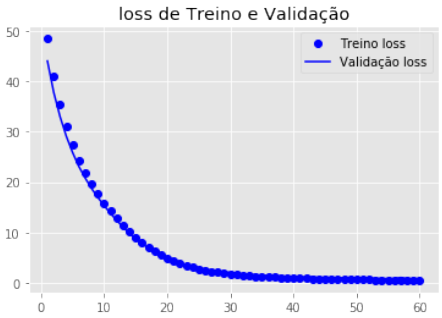
Note que novamente VGG-16 apresentou resultados superiores ao da InceptionV3 tanto em relação a acurácia, como a perda.

Figura 25 – VGG-16 e InceptionV3 - *fine-tuning* I

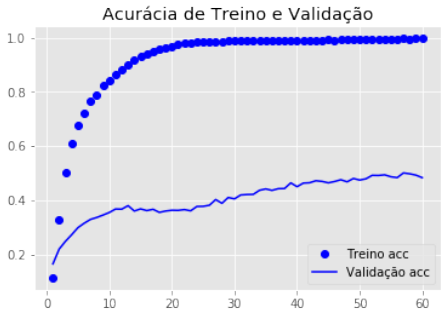
(a) Acurácia VGG-16



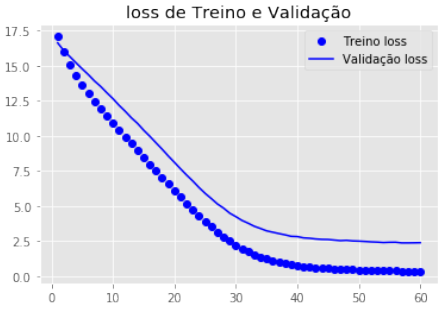
(b) Perda VGG-16



(c) Acurácia InceptionV3



(d) Perda InceptionV3



Fonte: Elaborado pelo autor

Tabela 7 – Resultados do cenário de experimentação IV

Arquitetura	**Acurácia**	**Perda**
VGG-16	88.8%	0.84
InceptionV3	48.3%	2.37

Fonte: Elaborado pelo autor

4.3.5 Cenário de Experimentação V

A fim de fornecer mais capacidade para a rede, o objetivo deste cenário V e o do próximo, o cenário VI, foi analisar o efeito sob os resultados ao realizar o *fine-tunig* descongelando mais blocos de camadas das redes VGG-16 e InceptionV3.

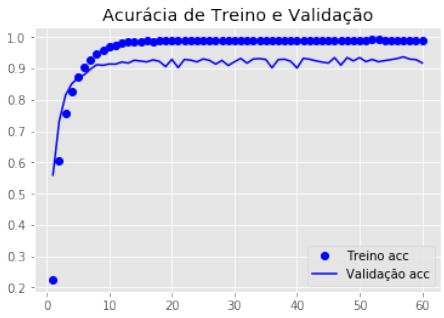
Neste cenário, foram desafixados os pesos dos dois últimos blocos de camadas da rede VGG-16 e para a rede InceptionV3 foram mantidos os pesos fixos das primeiras 229 camadas desafixando os pesos das restantes. Para este experimento, as configurações dos cenários anteriores foram repetidas, havendo apenas esta alteração para o *fine-tunig*.

As Figuras 26a e 26b apresentam os gráficos de a acurácia e perda da rede VGG-16 e as Figuras 26c e 26d apresentam os gráficos de acurácia e perda da rede InceptionV3.

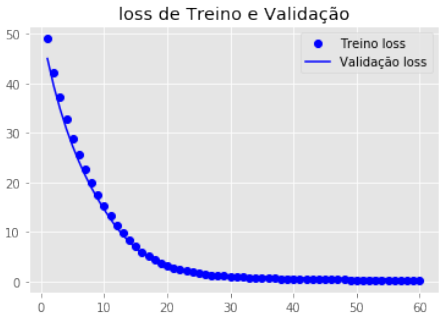
A Tabela 8 apresenta os resultados ao final deste cenário de experimentação.

Figura 26 – VGG-16 e InceptionV3 - *fine-tuning* II

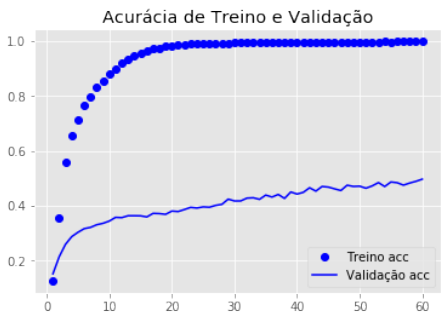
(a) Acurácia VGG-16



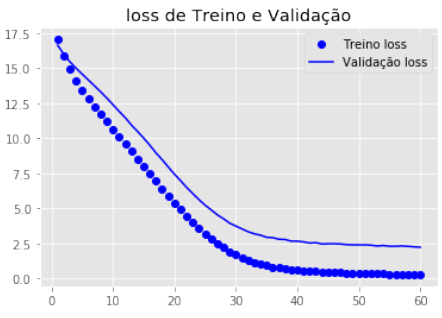
(b) Perda VGG-16



(c) Acurácia InceptionV3



(d) Perda InceptionV3



Fonte: Elaborado pelo autor

Tabela 8 – Resultados do cenário de experimentação V

Arquitetura	**Acurácia**	**Perda**
VGG-16	91.7%	0.59
InceptionV3	49.7%	2.22

Fonte: Elaborado pelo autor

4.3.6 Cenário de Experimentação VI

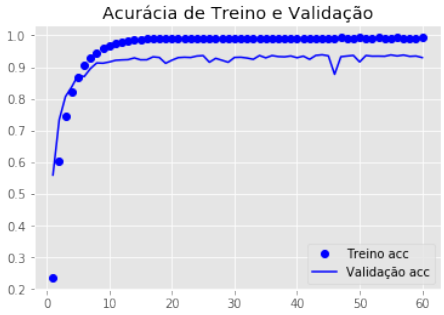
Neste cenário, foram desafixados os pesos dos três últimos blocos de camadas da rede VGG-16 e desafixados os pesos a partir da camada 197 para a rede InceptionV3.

As Figuras 27a e 27b apresentam os gráficos de a acurácia e perda da rede VGG-16 e as Figuras 27c e 27d apresentam os gráficos de acurácia e perda da rede InceptionV3.

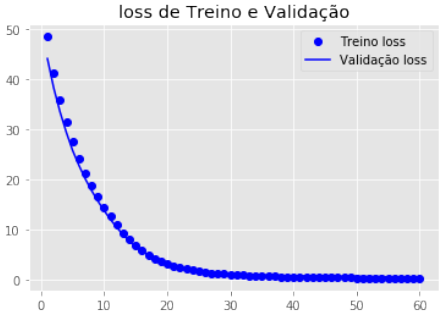
A Tabela 9 apresenta os resultados ao final deste cenário de experimentação.

Figura 27 – VGG-16 e InceptionV3 - *fine-tuning* III

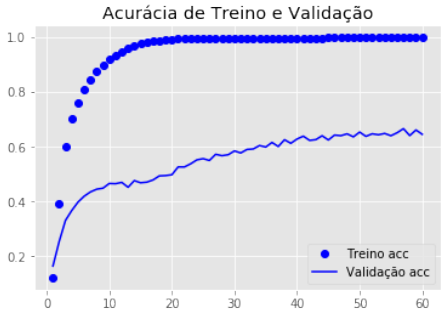
(a) Acurácia VGG-16



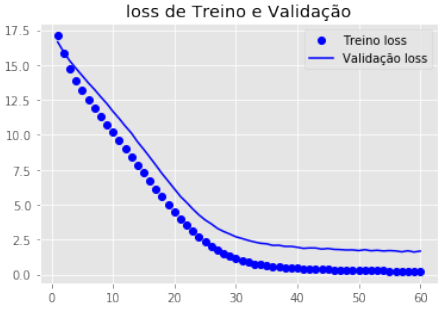
(b) Perda VGG-16



(c) Acurácia InceptionV3



(d) Perda InceptionV3



Fonte: Elaborado pelo autor

Tabela 9 – Resultados do cenário de experimentação VI

Arquitetura	**Acurácia**	**Perda**
VGG-16	93%	0.52
InceptionV3	64.5%	1.67

Fonte: Elaborado pelo autor

4.4 RESULTADOS E DISCUSSÕES

Ao analisar os resultados obtidos durantes todos os experimentos, é notório o fato de que a arquitetura VGG-16, apesar de ser mais simples e menor, em termo de profundidade, obteve resultados superiores ao da arquitetura InceptionV3.

É importante observar que a arquitetura InceptionV3 parece precisar que mais camadas sejam descongeladas para o *fine-tuning*, do que a rede VGG-16, pois somente no último cenário, a InceptionV3 dá uma perspectiva de melhora.

Para o teste, foi selecionada a arquitetura que obteve melhor resultado nos treinos, sendo assim o conjunto de teste foi submetido à rede VGG-16 com taxa *dropout* de 50%, regularização no valor de 0.001 e *fine-tuning* nos últimos três blocos. A Tabela 10 mostra os resultados de acurácia e perda do modelo proposto para o conjunto de teste.

Tabela 10 – Resultado da rede VGG-16 no conjunto de teste

Arquitetura	**Acurácia**	**Perda**
VGG-16	98%	0.29

Fonte: Elaborado pelo autor

Uma observação feita é que o modelo obteve um bom resultado para o conjunto de teste o que significa que o modelo proposto consegue generalizar bem sobre os dados, não ficando restrito apenas ao conjunto de treino.

5 EXPERIMENTOS ADICIONAIS

Com o objetivo de continuar avaliando o desempenho das arquiteturas, foram propostos alguns experimentos adicionais com algumas alterações principalmente em relação a base de dados.

5.1 BASE ADAPTADA

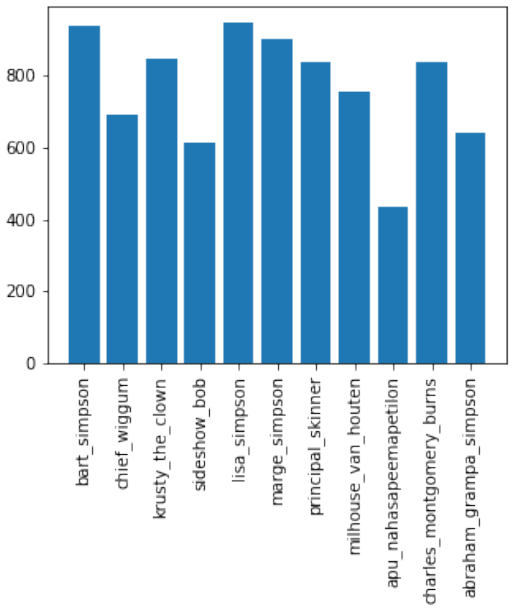
Foi proposta uma filtragem na base de dados original, para construir uma nova base a partir da mesma, que foi chamada de base balanceada. Esse nome “balanceada” se dá pelo fato de terem sido escolhidas classes que continham quantidades de imagens nos valores entre 400 e 1000, conforme a Figura 19a.

A nova base passou, portanto, a ser formada apenas por 11 classes e não mais por 42, além de ser considera uma classe bem mais balanceada em relação a inicial. Em valores, a nova base passou a ter 8.436 imagens no conjunto de treino, 3.622 imagens no conjunto de validação e 499 imagens no conjunto de teste.

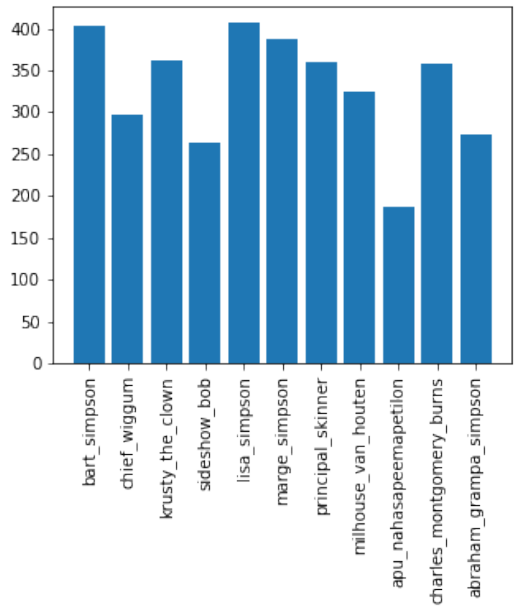
A Figura 28 mostra a distribuição das imagens em cada classe da nova base de dados nos conjuntos de treino 28a e validação 28b.

Figura 28 – Histograma de distribuição das classes da nova base no conjunto de treino e validação

(a) Conjunto de treino



(b) Conjunto de validação



Fonte: Elaborado pelo autor

5.2 EXPERIMENTOS

Os experimentos adicionais com os modelos propostos foram divididos em mais dois **Cenários de Experimentação**, com a proposta de avaliar a performance das arquiteturas para a nova base de dados proposta.

As configurações das arquiteturas foram mantidas iguais às do Capítulo 4.

5.2.1 Cenário de Experimentação VII

O objetivo deste cenário foi analisar o efeito da base balanceada em relação à base inicial desbalanceada. Para isto, foram repetidas as configurações do experimento do cenário VI sobre esta nova base de dados.

As Figuras 29a e 29b apresentam os gráficos de a acurácia e perda da rede VGG-16 e as Figuras 29c e 29d apresentam os gráficos de acurácia e perda da rede InceptionV3. A Tabela 11 apresenta os resultados ao final deste experimento. É importante ressaltar neste cenário, a diminuição considerável do *overfitting* para a arquitetura InceptionV3.

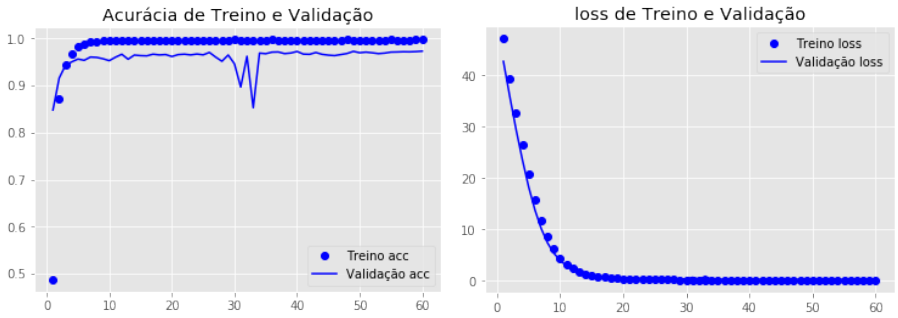
Tabela 11 – Resultados do cenário de experimentação VII das redes VGG-16 e InceptionV3 sobre a nova base

Arquitetura	**Acurácia**	**Perda**
VGG-16	97.3%	0.17
InceptionV3	75.4%	1.0

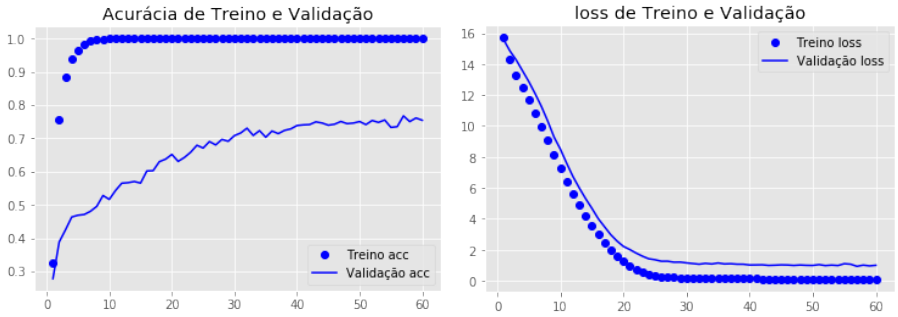
Fonte: Elaborado pelo autor

Figura 29 – VGG-16 e InceptionV3 sobre o conjunto balanceado - *fine-tuning* III

(a) Acurácia VGG-16 (b) Perda VGG-16



(c) Acurácia InceptionV3 (d) Perda InceptionV3



Fonte: Elaborado pelo autor

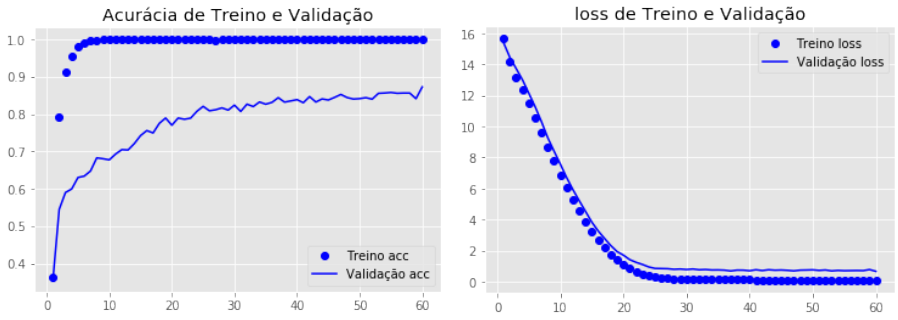
5.2.2 Cenário de Experimentação VIII

Como no cenário anterior a rede VGG-16 continua a superar a rede InceptionV3, neste experimento optou-se por dar mais capacidade apenas a rede InceptionV3 e analisar o comportamento da rede ao desafixar os pesos de mais um bloco de camadas, sendo assim, foram mantidos os pesos fixos das primeiras 165 camadas desafixando os pesos das restantes. Para este experimento, as mesmas configurações anteriores foram mantidas.

A Figura 30 apresenta os gráficos de acurácia e perda da rede InceptionV3 e Tabela 12 apresenta os resultados ao final deste experimento. É importante notar que há uma diminuição ainda mais evidente do *overfitting*.

Figura 30 – InceptionV3 sobre o conjunto balanceado - *fine-tuning* IV

(a) Acurácia (b) Perda



Fonte: Elaborado pelo autor

Tabela 12 – Resultado do cenário de experimentação VIII da InceptionV3 sobre o conjunto balanceado

Arquitetura	**Acurácia**	**Perda**
InceptionV3	87.3%	0.66

Fonte: Elaborado pelo autor

5.3 RESULTADOS E DISCUSSÕES

Analisando os resultados obtidos, pode-se observar que ao causar um balanceamento nas classes da base de dados houve uma melhora significativa na acurácia dos modelos, principalmente para a rede InceptionV3 onde, no cenário de experimentação VI, seu resultado havia sido 64.5% de acurácia e 1.67 de perda, no experimento com esta nova base, sobre as mesmas configurações, a Tabela 11 relata uma acurácia de 75.4% e perda de 1.0.

Como a rede VGG-16 obteve um melhor resultado no primeiro experimento, o novo conjunto de teste foi submetido a mesma. Este conjunto também foi submetido à rede InceptionV3 deste último experimento. A Tabela 13 mostra os resultados de acurácia e perda das redes VGG-16 e InceptionV3 para o novo conjunto de teste.

Tabela 13 – Resultado das redes VGG-16 e InceptionV3 no novo conjunto de teste

Arquitetura	**Acurácia**	**Perda**
VGG-16	99%	0.085
InceptionV3	89.1%	0.50

Fonte: Elaborado pelo autor

Em ambas as arquiteturas houve uma melhora tanto na acurácia quanto no valor de perda quando comparado ao resultado presente na Tabela 9, isso mostra a influência que o desbalanceamento nos dados tem sobre os resultado.

Um outra observação feita é, conforme a Figura 30, a rede InceptionV3 precisou que mais camadas fossem descongeladas para obter resultados satisfatórios e assim se aproximar dos resultados obtidos pela rede VGG-16.

6 CONCLUSÕES

Neste trabalho foi utilizado a transferência de conhecimento a partir de redes neurais profundas treinadas sob o conjunto de dados ImageNet e teve como objetivo analisar o uso da técnica de *fine-tuning* e como esta influencia nos resultados sobre a base de dados proposta.

A partir destes experimentos foi permitido concluir que mantendo os hiperparâmetros fixos, a inclusão de fine-tunning melhorou a acurária da classificação de personagens. Essa melhora pôde ser vista de forma crescente ao longo deste trabalho. A partir do cenário base, os cenários seguintes obtiveram melhorias sejam relacionadas à acurácia e perda ou em relação a diminuição do *overfitting*. Desde os resultados obtidos no cenário de experimentação I, na seção 4.3.1, até os resultados obtidos no cenário de experimentação VI, na seção 4.3.6, os valores de acurácia obtiveram uma melhora de cerca de 11% para a rede VGG-16 e de cerca de 149% para a rede InceptionV3. O mesmo vale para os valores de perda, onde para a rede VGG-16 houve um decréscimo de cerca de 25% e para a rede InceptionV3 houve uma redução de cerca de 80%.

Outra conclusão que obteve-se foi que para as configurações proposta, em todas os experimentos as rede VGG-16 superava a rede InceptionV3 em termos de acurácia de perda, porém um fator interessante notado pelo autor é que em termos de tempo necessário para um experimento, a InceptionV3 era bem mais rápida que a VGG-16.

Sobre os experimentos adicionais realizados, no cenário VII, na seção 5.2.2, notou-se uma melhoria em termos de acurácia e perda das redes, principalmente a InceptionV3, causada pela nova base de dados proposta, dado a característica de suas classes serem balanceadas. Outra análise feita foi em relação a rede InceptionV3 com mais camadas de *fine-tuning* sobre a nova base de dados, obtendo os melhores resultados para este modelo e concluindo que a InceptionV3 precisou que mais camadas fossem descongeladas e treinadas para obter resultados satisfatório.

Como trabalhos futuros pretende-se explorar mais ainda as técnicas de *fine-tuning*. Uma opção é treinar redes partidos das *features* produzidas nas primeiras camadas de uma rede pré-treinada, pois os padrões nos cartoons são mais simples. Pretende-se também, aplicar estes experimentos sobre outras bases como de fotografias ou de imagens médicas.

REFERÊNCIAS

AGUIAR, D. G. N. **Transferência de conhecimento utilizando aprendizado profundo para classificação de imagens histopatológicas**. 2017.

ARAÚJO, F. H.; CARNEIRO, A. C.; SILVA, R. R.; MEDEIROS, F.; USHIZIMA, D. Redes neurais convolucionais com tensorflow: Teoria e prática. **SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. III Escola Regional de Informática do Piauí. Livro Anais-Artigos e Minicursos**, Sociedade Brasileira de Computação, v. 1, p. 382–406, 2017.

BALLARD, D. H.; BROWN, C. M. **Computer Vision**. 1st. ed. [S.l.]: Prentice Hall Professional Technical Reference, 1982. ISBN 0131653164.

BEZERRA, E. Introdução à aprendizagem profunda. **Tópicos em Gerenciamento de Dados e Informações. Porto Alegre: Sociedade Brasileira de Computação**, p. 57–86, 2016.

CANZIANI, A.; PASZKE, A.; CULURCIELLO, E. An analysis of deep neural network models for practical applications. **arXiv preprint arXiv:1605.07678**, 2016.

CARVALHO, M. C. *et al.* **Transfer schemes for deep learning in image classification**: Esquemas de transferência para aprendizado profundo em classificação de imagens. [sn], 2015.

CENON STACK. **Log Analytics With Deep Learning And Machine Learning**. [S.l.], 2017. Disponível em: <https://www.xenonstack.com/blog/data-science/log-analytics-with-deep-learning-and-machine-learning>. Acesso em: 23 Maio 2018.

CHOLLET, F. **Fine-tune InceptionV3 on a new set of classes**. 2015. Disponível em: <https://keras.io/applications/>. Acesso em: 11 nov. 2018.

CHOLLET, F. **Deep Learning with Python**. 1st. ed. Greenwich, CT, USA: Manning Publications Co., 2017. ISBN 1617294438, 9781617294433.

CRESPO, S. A. L. H. N. **Reconhecimento de tumores cerebrais utilizando redes neurais convolucionais**. Universidade Federal do Pampa, 2017.

EL-SAYED, M. A.; ESTAITIA, Y. A.; KHAFAGY, M. A. Automated edge detection using convolutional neural network. **Editorial Preface**, Citeseer, v. 4, n. 10, 2013.

FERREIRA, A. d. S. **Redes Neurais Convolucionais Profundas na Detecção de Plantas Daninhas em Lavoura de Soja**. Dissertação (Mestrado), 2017.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. Disponível em: <http://www.deeplearningbook.org>. Acesso em: 11 nov. 2018.

HAYKIN, S. **Redes neurais**: princípios e prática. [S.l.]: Bookman Editora, 2007.

HOSANG, J.; OMRAN, M.; BENENSON, R.; SCHIELE, B. Taking a deeper look at pedestrians. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2015. p. 4073–4082.

KARPATHY, A. **Convolutional Neural Networks for Visual Recognition**. 2015. Disponível em: <http://cs231n.github.io/convolutional-networks/>. Acesso em: 15 nov. 2018.

KISHI, S. **How to make Fine tuning model by Keras**. 2017. Disponível em: <http://marubon-ds.blogspot.com/2017/08/how-to-make-fine-tuning-model.html>. Acesso em: 21 nov. 2018.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2012. p. 1097–1105.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436, 2015.

LEE, H.; GROSSE, R.; RANGANATH, R.; NG, A. Y. Unsupervised learning of hierarchical representations with convolutional deep belief networks. **Communications of the ACM**, ACM, v. 54, n. 10, p. 95–103, 2011.

MARENGONI, M.; STRINGHINI, S. Tutorial: Introdução à visão computacional usando opencv. **Revista de Informática Teórica e Aplicada**, v. 16, n. 1, p. 125–160, 2009.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. **Sistemas inteligentes-Fundamentos e aplicações**, v. 1, n. 1, p. 32, 2003.

PATEL, A. **Chapter-7 Under-fitting, over-fitting and its solution**. 2018. Disponível em: <https://medium.com/ml-research-lab/under-fitting-over-fitting-and-its-solution-dc6191e34250>. Acesso em: 21 nov. 2018.

RIOS, L. R. S. **Visão computacional**. Departamento de Ciência da computação—Universidade Federal da Bahia, Salvador, BA,, 2011.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M. *et al.* Imagenet large scale visual recognition challenge. **International Journal of Computer Vision**, Springer, v. 115, n. 3, p. 211–252, 2015.

SHARMA, S. **Activation Functions: Neural Networks**. 2017. Disponível em: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>.

SILVA, I. R.; SOUZA, R. G. de; SILVA, G. S.; OLIVEIRA, C. S. de; CAVALCANTI, L. H.; BEZERRA, R. S.; ROBERTA, A. d. A.; SANTOS, W. P. dos. **Utilização de Redes Convolucionais Para Classificação e Diagnóstico da Doença de Alzheimer**. 2018.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2014.

SPANHOL, F. A.; OLIVEIRA, L. S.; PETITJEAN, C.; HEUTTE, L. A dataset for breast cancer histopathological image classification. **IEEE Transactions on Biomedical Engineering**, IEEE, v. 63, n. 7, p. 1455–1462, 2016.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. **The Journal of Machine Learning Research**, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2015. p. 1–9.

SZEGEDY, C.; VANHOUCKE, V.; IOFFE, S.; SHLENS, J.; WOJNA, Z. 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.

SZELISKI, R. **Computer vision**: algorithms and applications. [S.l.]: Springer Science & Business Media, 2010.

ZHOU, B.; LAPEDRIZA, A.; XIAO, J.; TORRALBA, A.; OLIVA, A. Learning deep features for scene recognition using places database. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2014. p. 487–495.

ŽIVKOVIĆ, N. M. **Introduction to Convolutional Neural Networks**. 2018. Disponível em: <https://www.codeproject.com/Articles/1232042/Introduction-to-Convolutional-Neural-Networks>. Acesso em: 21 nov. 2018.