# **Classificação de imagens na triagem de pacientes com catarata**

Douglas Jonatas do Carmo Dias¹; Ana Beatriz Pereira Sette2

¹ Centro Universitário de Lins. Engenheiro de Computação. Rua: José Bonifácio n°244 – Ribeiro; 16401-115. Lins, São Paulo, Brasil.

2 Universidade Federal de Viçosa. Doutora em Economia Aplicada. Departamento de Economia Rural, Avenida Purdue, s/nº, Edifício Edson Potsch Magalhães – Campus Universitário; 36570-900. Viçosa, Minas Gerais, Brasil.

**Classificação de imagens na triagem de pacientes com catarata**

# **Resumo**

A automação de processos cada vez mais crescente obriga a existência de aplicações deste tipo na área de visão computacional. Visão computacional é uma ferramenta que pode ser útil em todas as áreas em que estiver. O objetivo deste trabalho foi a aplicação de redes neurais convolucionais para a identificação de existência ou não de quatro classes de objetos em imagens. Para isso, utilizou-se técnicas como “fine-tuning” e “data augmentation”. Este trabalho deu grande importância à explicação de conceitos de visão computacional visando a facilitação da replicação dos procedimentos aqui aplicados. As principais redes neurais convolucionais utilizadas neste trabalho foram: VGG16, Xception e ResNet101. Como principais resultados, foram obtidos as matrizes de confusão destas redes neurais convolucionais e algumas métricas oriundas destas matrizes. Por fim, comparou-se os resultados obtidos em cada umas das redes e observou-se um melhor desempenho da Xception diante das outras redes neurais convolucionais. Esta rede apresentou métricas de acurácia, precisão, sensibilidade e “f1-score” superiores a 98%. As diferenças de resultados entre as redes foram analisadas e foram sugeridas novas linhas de ação para o melhoramento dos resultados destas redes.

**Palavras-chave:** redes neurais convolucionais; objetos; imagens; visão computacional.

# **Introdução**

Devido ao crescimento das divícias sociodemográficas e da perspectiva de vida mais longa, vários países ao redor do mundo começaram a notar sua população atingindo a idade adulta, aumentando assim a idade média da população e como consequência trazendo mudanças na carga de doenças para doenças não transmissíveis e deficiências. A maioria das principais causas de deficiência visual é a catarata (R. Bourne et al., 2021).

Carata é a causa mais prevalente de cegueira reversível no mundo, sendo responsável por aproximadamente 50% dos 50 milhões de casos. Por definição, se refere a opacidade do cristalino, podendo ser adquirida, o que inclui o processo de envelhecimento. As fontes causais da catarata não foram plenamente elucidadas, porém existem fatores de riscos que devem ser evitados, pois a catarata é uma questão de saúde pública intimamente relacionada com o aumento da expectativa de vida no Brasil e no mundo. (V. O. Domingues et al., 2016).

# **Material e Métodos**

Os materiais utilizados para o desenvolvimento do trabalho foram: Google Colab, o projeto foi desenvolvido em python utilizando as bibliotecas: matplotlib.pyplot, numpy, tensorflow, tensorflow\_hub e keras. Preprocessing. Image.

O modelo utilizado para fazer a classificação das imagens foi o efficientnetv2-b2-21k, é um modelo de classificação encontrado no TensorFlow Hub, tem vários modelos de classificação e para o desenvolvimento do projeto esse é o que se adequou melhor.

---------------------------------------------------------------------------------------------------------------------------

Esta seção é dividida em três partes:**Contextualização Teórica**

Inicialmente é importante que se estabeleça alguns conceitos importantes para o entendimento das CNNs. Nesta seção serão apresentados conceitos das seguintes camadas existentes em uma rede neural convolucional:

* “Convolutional layer” ou camadas convolucionais;
* “Activation layer” ou camada de ativação;
* “Pooling layers” ou camadas de “pooling”; e
* “Fully connected layer” ou camadas totalmente conectadas.

Além disso, será apresentado o conceito de “fine-tuning” (que foi utilizado durante a implementação da VGG16, Xception e ResNet101) e uma breve explicação sobre os otimizadores utilizados nas CNNs.

As **camadas convolucionais** são o diferencial das redes neurais convolucionais. De forma direta, a convolução é uma operação matemática entre duas matrizes: uma matriz de dimensões maiores (normalmente, a imagem de entrada onde cada valor representa um “pixel”) e uma matriz de dimensões menores (chamada de filtro ou “kernel”). De forma bem genérica, pode-se dizer que na operação de convolução o “kernel” percorre a imagem de entrada realizando um conjunto de operações de multiplicação e soma.

Cada filtro, ou “kernel”, percorre toda a imagem gerando um mapa de características ou “feature map”. Estes mapas, produto de cada operação de convolução dos filtros com a imagem de entrada, captam, cada um, alguma característica marcante da imagem de entrada.

A operação de convolução pode gerar uma redução espacial da imagem de entrada. Esta redução pode acontecer devido a duas propriedade desta operação: “stride” e o “zeropadding”. O “stride” regula o número de “pixels” consecutivos que o “kernel” irá percorrer em cada operação. Enquando o “zero-padding” aumenta as dimensões da imagem de entrada antes da operação de convolução (completando as bordas com zeros) com a finalidade de impedir a redução da imagem de entrada pela operação de convolução (para mais informações sobre estes conceitos, consultar Joshi (2015)). A Figura 1 apresenta uma ilustração deste processo.

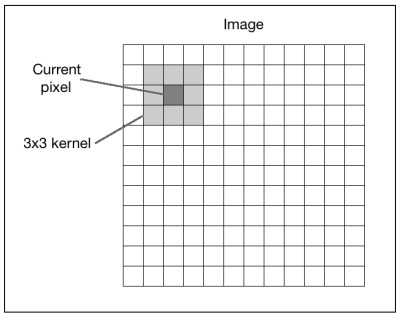


Figura 1: Operação de convolução entre matrizes

Fonte: Joshi (2015)

Uma rede neural é uma função parametrizada compostas por uma parcela linear e outra não-linear. A parcela linear é o produto das entradas pelos pesos mais o viés (“bias”). Enquanto a parcela não-linear é a função de ativação. A função de ativação recebe o sinal da parcela linear e faz uma operação não linear. A **camada de ativação** é onde fica a função de ativação. Na Figura 2 são apresentados exemplos de funções de ativação.

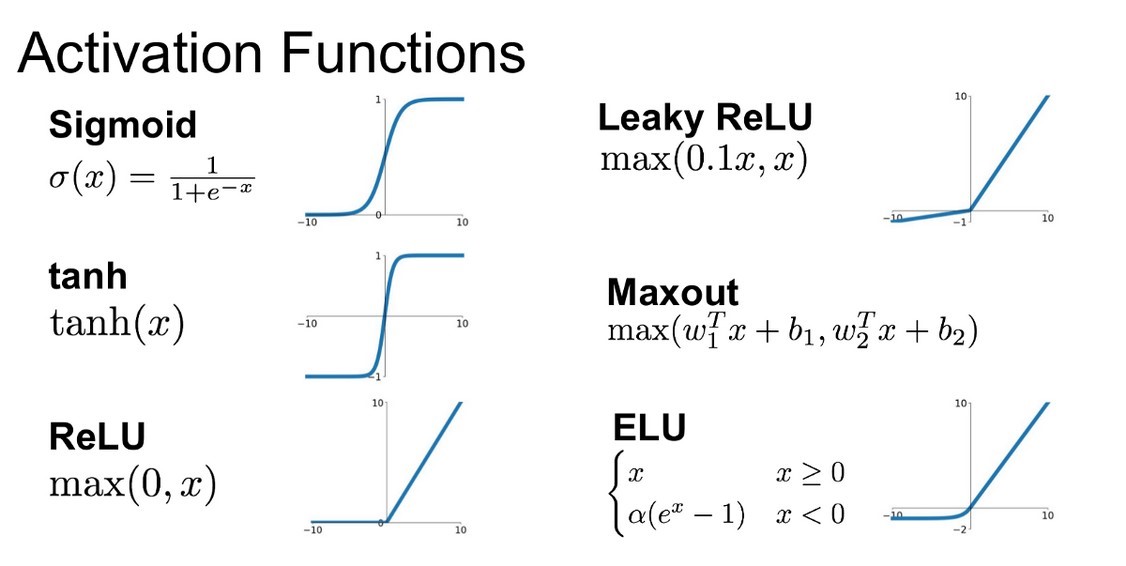


Figura 2: exemplos de funções de ativação

Fonte: Géron (2021)

As **camadas de “pooling”** são utilizadas para reduzir a dimensão da matriz de entrada. Elas segregam uma região da matriz de entrada e fazem uma operação matemática com os valores segregados. Esta operação pode ser a média destes valores ou a escolha do valor máximo. Uma função muito conhecida é a “max pooling” que escolhe o maior valor dentre os segregados. A Figura 3 ajudará a entender este conceito.

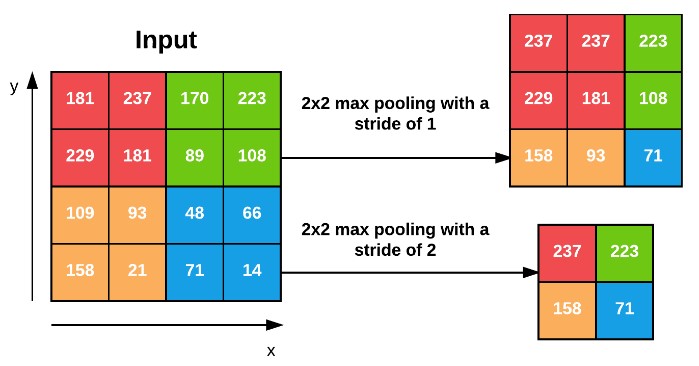


Figura 3: aplicação do “max pooling” com “stride” igua a 1 e 2

Fonte: Rosebrock (2021)

As **camadas totalmente conectadas** geralmente são alocadas no final da

arquitetura da rede neural. Elas capturam todos os sinais emitidos pela camada anterior e geralmente, após uma camada totalmente conectada, há uma camada de ativação encerrando a arquitetura da rede.

Arquiteturas famosas, das quais três foram utilizadas neste trabalho, costumam ser testadas em grandes “datasets” disponíveis na internet. Por exemplo, a VGG16 foi treinada em um “dataset” chamado “ImageNet”. Este “dataset” possui mais de um milhão de imagens de mil categorias diferentes. O processo de treinamento de uma rede neural convolucional do zero é algo muito custoso computacionalemnte, por isso os pesos obtidos durante o treinamento da VGG16 no “ImageNet” são informações importantes.

Por esta razão existe o **processo de “fine-tuning”**. No atual trabalho, foi realizada a classificação de quatro classes de imagens utilizando a VGG16. Os pesos da VGG16 prétreinada, porém, levaram em conta mil classes. Como se poderia reaproveitar esses pesos? O “fine-tuning” é o ato de literalmente cortar as últimas camadas da rede e substituir por camadas da escolha do autor com um classificador de número de classes igual ao número que se deseja classificar.

A Figura 4 apresenta o processo de “fine-tuning” da VGG16. A rede da esquerda representa a arquitetura inicial. A rede do meio representa a arquitetura inicial sem as camadas finais (este processo gera o que chama-se de “features”, essas “features” podem ser usadas como dados de entrada de regressores, por exemplo). E a rede da direita representa o processo de “fine-tuning”: as camadas iniciais pré-treinadas permanecem e as camadas finais são trocadas de acordo com o objetivo do usuário.

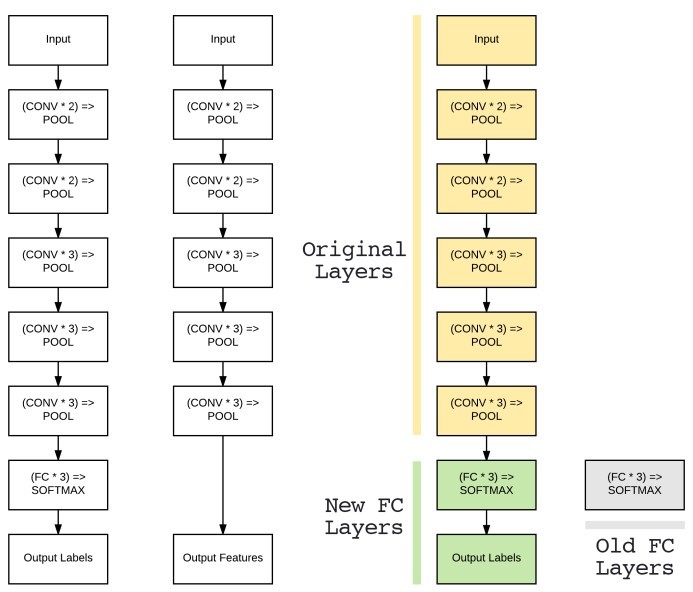


Figura 4: “fine-tuning” da VGG16

Fonte: Rosebrock (2017)

Este processo permite economia no treinamento da rede neural para a classificação de novas imagens, tendo em vista que o trabalho basicamente residirá em treinar as últimas camadas da rede. O processo de “fine-tuning” foi utilizado neste trabalho.

Após a aplicação do processo de “fine-tuning”, a nova CNN possui dois conjuntos de camadas: as que possuem valores de pesos e “bias” predefinidos devido ao pré-treinamento sofrido e as que possuem valores de pesos e “bias” aleatórios por terem sido recém enxertadas. Devido a esta peculiaridade, a fase de treinamento deste tipo de CNN costuma ser divida em duas partes. Na primeira parte, congela-se os pesos das camadas prétreinadas permitindo que apenas as camadas recém enxertadas sejam atualizadas. Na segunda parte permite-se que todos os pesos sejam atualizados.

Esta técnica de treinamento é importante, pois a camada recém enxertada é inicializada com valores de pesos e “bias” aleatórios. Se inicialmente fosse permitido que todas as camadas fossem atualizadas, os valores aleatórios das camadas recém enxertadas poderiam influenciar negativamente na atualização dos pesos das camadas pré-treinadas.

A atualização dos pesos e “bias” da rede neural é realizada através dos **otimizadores**. A função de um otimizador é atualizar os parâmetros da rede com a finalidade de reduzir os valores da função de perda. A função de perda é a diferença entre os valores preditos pela rede e os valores reais. A forma como um otimizador atualiza os parâmetros da rede é peculiar a cada otimizador.

O otimizador utilizado neste trabalho foi o SGD, “Stochastic Gradient Descent”. A equação de atualização dos pesos deste otimizador é dada por: *dL*

# w=w−n (1)

*dw* Onde,

* L: função de perda;
* n: “learning rate” ou taxa de aprendizado (definido pelo usuário); e  w: pesos da rede neural.

## Coleta e tratamento de dados

Os dados coletados para esse trabalho foram retirados do Kaggle. As imagens já estavam com os rótulos e as mesmas foram colocas no google driver, sendo possível fazer o carregamento para a área de trabalho do Google Colab.

Tratando-se de qualquer modelo de machine learning ou deep learning uma das partes mais importantes no treinamento de qualquer modelo é a parte do pré-processamento dos dados de entrada. Sendo eles:

* Escolha de um conjunto de imagens já com os rótulos;

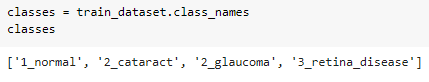


Figura 1: Rótulo das imagens

Fonte: Dados originais da pesquisa

Após a seleção do conjunto de imagens já com os rótulos é necessário definir o tamanho das imagens de acordo com o modelo utilizado, nesse caso as imagens terão um tamanho de 260x260 pixels

* Divisão das imagens para o treino;

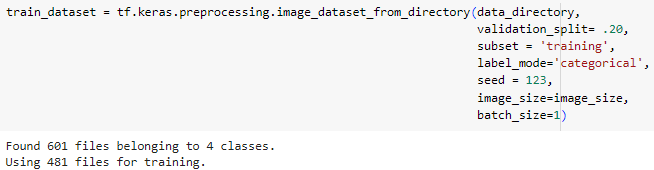


Figura 2: Treino do dataset

Fonte: Imagem proveniente da pesquisa

Para fazer o treinamento foi utilizado 80% das imagens (481) e ficando 20% (120) para o teste(validação) para isso foi utilizado o método split, a etiqueta desse modelo será por categoria, ou seja: [‘1\_normal’, ‘2\_cataract’, ‘2\_glaucoma’, ‘3\_retina\_disease’], toda vez que a função for chamada será composta pelas imagens, o tamanho das imagens serão 260x260 e o número de amostras processadas antes do modelo ser atualizado será 1.

* Pré-processamento e embaralhamento das imagens;

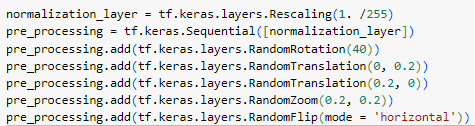


Figura 3: Adicionando e embaralhando as imagens

Fonte: Autor

Foi feito a transformação de dados (com normalization) que alinha os valores dos dados a uma escola comum, utilizado o rescaling para reduzindo a escala (pixels) das imagens de 0-255 para 0-1. Como será passado uma sequencial de camadas para a rede neural será utilizado o Sequential. Também serão adicionadas algumas imagens no pré- processamento, pois pode ser que as imagens separadas para o treinamento não sejam o suficiente, logo será necessário “criar” imagens para o mesmo, sendo adicionadas algumas imagens aleatórias com algumas modificações como: girando 40°, deslocando 20% das imagens no horizontal e vertical, aumentando o zoom em 20% e com o zoom reduzido em 20% e girando as imagens horizontalmente.

* Divisão das imagens para o teste.

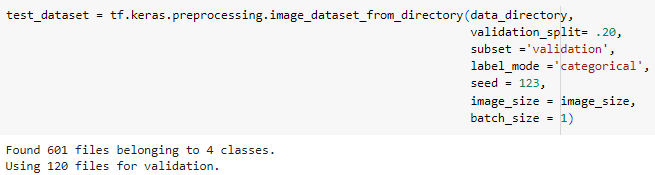


Figura 4: Imagens para o teste

Fonte: Criado pelo autor

O preprocessamento adotado neste trabalho englobou o embaralhamento das imagens antes da divisão em conjunto de treino, validação e teste, o redimensionamento da imagem para o formato requerido pelas redes neurais convolucionais e a transformação da imagem em um objeto do tipo “numpy.array”, que é uma espécie de representação de tensores em “python”.

Para o embaralhamento das imagens utilizou-se a função “train\_test\_split” do pacote “Scikit-Learn”. Essa função é bastante empregada para dividir um conjunto de dados em dois subconjuntos distintos: um conjunto de treinamento e um conjunto de teste. Essa divisão é essencial para avaliar o desempenho do modelo de forma realista e evitar problemas de “overfitting”. Ao chamar esta função, deve ser fornecido como parâmetro da mesma as variáveis independentes, “X”, e os rótulos, “y”. O resultado são quatro conjuntos de dados, dois de variáveis independentes, “X\_train” e “X\_test”, e dois de rótulos, “y\_train” e “y\_test”. Outro parâmetro importante desta função é o “random\_state”. Este parâmetro é opicional e recebe como entrada números inteiros. Ele nada mais é que um valor de semente aleatória que permite a repetição exata das divisões em treino e teste caso seja necessário rodar novamente o programa.

Para a realização do redimensionamento e transformação do tipo do objeto foi desenvolvido um “pipeline”. Um “pipeline” é um tipo de objeto que realiza as transformações requeridas num dado de entrada de forma a abstrair a complexidade dessas transformações no corpo do código principal. Ou seja, no corpo principal do código, só é vista uma “caixa preta” onde se insere uma imagem e recebe-se um tensor de dimensões predefinidas. A Figura 5 apresenta a abstração gerada no corpo do código principal devido à criação do “pipeline” “SimpleDatasetLoader”. Observe que para se realizar as transformações utilizando o “SimpleDatasetLoader” basta utilizar o método “load” deste objeto.

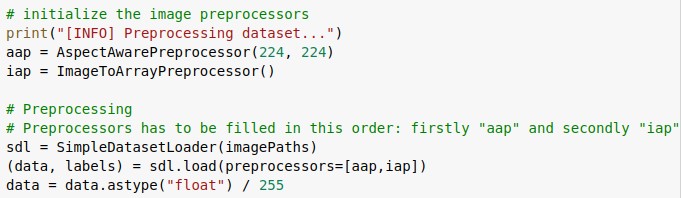


Figura 5: preprocessadores “aap” e “iap” e “pipeline” “sdl”

Fonte: Dados originais da pesquisa

Segundo Rosebrock (2017), de forma geral, quando se aplica “deep learning” para Visão Computacional, é bom que se tenha dentre 1000 a 5000 imagens por classe na fase de treinamento. Quando isso não acontece utiliza-se uma técnica chamada “data augmentation”. Esta técnica foi utilizada neste trabalho. Ela consiste na geração de imagens modificadas a partir de uma imagem original.

A Figura 6 é um exemplo do processo de “data augmentation”. Observe a imagem original a esquerda e as imagens “aumentadas” a direita. Os “datasets” utilizados neste trabalho (carros, drones, navios e aviões) possuem por volta de 800 imagens exemplos em cada classe na fase de treinamento, por esta razão foi realizado o processo de “data augmentation”.

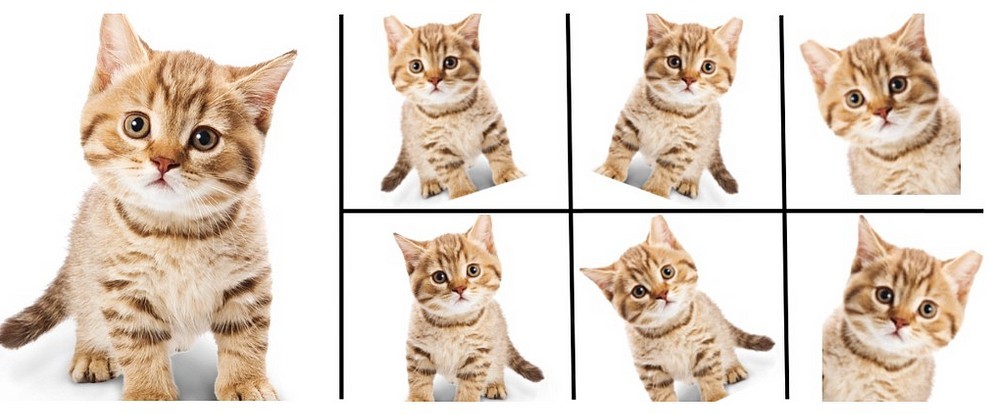


Figura 6: exemplo de “data augmentation”

Fonte: Raj (2018)

## YOLO

Fazer um navio é um projeto complexo. Quando se diz “complexo” se quer dizer que as atividades do projeto não são consecutivas, ou seja, atividades anteriores e posteriores, ao serem alteradas, influenciam uma nas outras. Uma forma de resolver um projeto complexo é através de uma espiral de projeto. A cada volta nesta espiral, há um maior detalhamento de cada uma das etapas. Após muitas voltas na espiral, chega-se próximo da solução ótima do projeto complexo, ou seja, a solução onde alterações em uma atividade não geram mudanças significativas em outras atividades. A Figura 7 apresenta um modelo de espiral de projeto utilizada em projetos de navios.

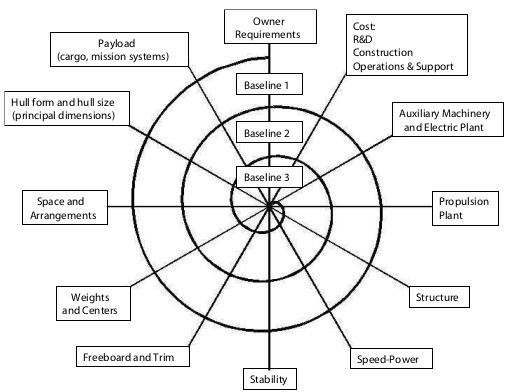


Figura 7: Espiral de projeto

Fonte: Lamb (2003)

Por que se falar em navios quando o objetivo deste trabalho é o projeto de uma rede neural convolucional para a identificação de objetos? Pois a aplicação da YOLO (através do “framework” “Darknet”) foi como uma primeira volta na espiral de projeto com a finalidade de se conhecer o caminho que seria percorrido.

Inicialmente, ressalta-se que diferentemente das outras redes neurais utilizadas, durante a implementação da YOLO, através da “Darknet”, não foi escrito nenhum código. Bastou a mera configuração de alguns parâmetros e arquivos deste “framework”. A Figura 8 apresenta a arquitetura da YOLO.

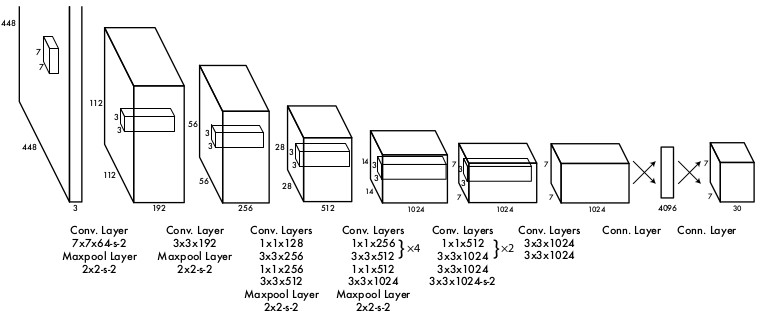


Figura 8: Arquitetura da YOLO

Fonte: Redmon, Divvala, Girshick e Farhadi (2015)

As principais alterações realizadas nesta arquitetura visavam a capacitação da mesma de identificar apenas uma classe de objetos. Ressalta-se que na aplicação deste “framework” utilizou-se apenas o banco de dados de imagens de drones.

A YOLO foi pré-treinada no banco de dados de imagens PASCAL VOC, “visual object classes challenge”. Este banco de dados de imagens contém 20 categorias de objetos. O procedimento realizado neste trabalho chama-se “transfer learning”. Neste procedimento inicia-se o treinamento para a identificação de uma nova classe a partir dos pesos utilizados no pré-treinamento (“transfer learning” é o nome do processo de aproveitamento dos pesos de uma rede neural pré-treinada, enquanto “fine-tuning” é o processo de substituição das últimas camadas de uma rede neural por outras camadas).

Por se tratar de um “framework”, não foi possível identificar exatamente quais camadas utilizavam os pesos pré-treinados. Por fim, ressalta-se que a YOLO realiza o que se chama de “object detection”. Um dos dados de entrada deste “framework” são as coordenadas das posições de interesse de cada imagem. A saída da YOLO é o que chamamos de imagem anotada, ou seja, é a imagem com uma “bounding box” ao redor do que ela identificou como o objeto de interesse.

Como foi dito inicialmente, o objetivo da aplicação da YOLO através da “Darknet” foi a realização de uma primeira volta na espiral de projeto. As métricas de acurácia da YOLO, devido ao tipo de resultado gerado por ela (imagens anotadas com a “bounding box”), são diferentes das métricas geradas pelas outras redes neurais utilizadas. Por esta razão, a aplicação deste “framework” servirá apenas como um exemplo inicial. Na seção de apresentação de resultados serão apresentados alguns resultados da YOLO, mas as comparações de métricas serão feitas apenas entre as outras CNNs utilizadas.

Uma vez dada a primeira volta na espiral do projeto, iniciou-se a implementação de um programa em “python” que recebeu imagens como dados de entrada e deveria retornar a classificação correta destas imagens de acordo com suas “labels”. Nesta próxima parte do trabalho foram utilizados os bancos de dados de imagens de drones, carros, navios e aviões.

## “VGG16”

A primeira rede neural convolucional utilizada foi a VGG16. A arquitetura desta rede neural pode ser vista na Figura 9. Ressalta-se novamente que esta rede neural foi prétreinada pelo “dataset” “ImageNet”. A imagem abaixo da arquitetura da rede representa a rede original, antes do processo de “fine-tuning”.

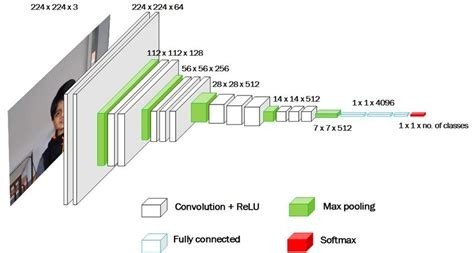


Figura 9: Representação da arquitetura da VGG16

Fonte: Khan (2019)

Abaixo apresenta-se a Tabela 1 com o nome de cada uma das camadas da VGG16.

Tabela 1. Camadas da VGG16

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| No | Nome | No | Nome | No | Nome | No | Nome |
| 0 | InputLayer | 6 | MaxPooling2D | 12 | Conv2D | 18 | MaxPooling2D |
| 1 | Conv2D | 7 | Conv2D | 13 | Conv2D | 19 | Flatten |
| 2 | Conv2D | 8 | Conv2D | 14 | MaxPooling2D | 20 | Dense |
| 3 | MaxPooling2D | 9 | Conv2D | 15 | Conv2D | 21 | Dense |
| 4 | Conv2D | 10 | MaxPooling2D | 16 | Conv2D | 22 | Dense |
| 5 | Conv2D | 11 | Conv2D | 17 | Conv2D |  | |

Fonte: Dados originais da pesquisa

Além das camadas já apresentadas no breve resumo teórico, tem-se na VGG16 outras camadas ainda não citadas. É válido ressaltar que o termo “Dense” se refere às camadas totalmente conectadas.

As camadas “Flatten” ainda não foram abordadas. Antes porém de falarmos sobre as camadas “Flatten” é importante que seja explicada a idéia de tensor. Até o momento temos falado de operações de matrizes. As matrizes são tensores de segunda ordem. Para um entendimento mais prático, tenhamos em mente que um escalar, um vetor e uma matriz são tensores de ordem zero, um e dois respectivamente.

As imagens em escala de cor cinza são formadas por pixels de zero a 255. Elas são representadas por tensores de segunda ordem, ou seja, matrizes. Já as imagens coloridas são representadas por três matrizes, uma em cada uma das escalas “red”, “green” e “blue”. As imagens coloridas são tensores de terceira ordem. A Figura 10 busca exemplificar o conceito de tensores.

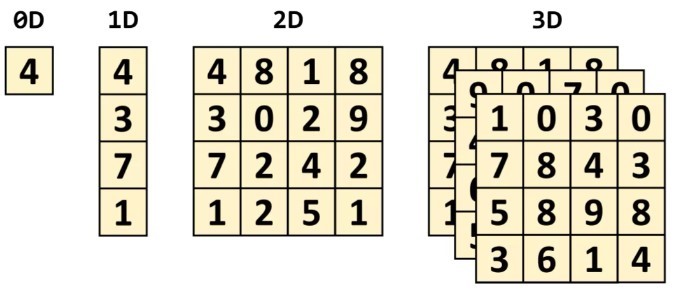


Figura 10: Conceito de tensor

Fonte: Evanson (2020)

Dada esta breve explicação de tensores, definamos, então, as camadas “Flatten”. Essas camadas transformam tensores de várias dimensões em tensores de primeira ordem. As camadas “Flatten” são geralmente colocadas antes das camadas “Dense”, pois estas requerem que seus dados de entrada estejam em formato unidimensional.

De forma geral, a VGG16 é uma CNN bastante curta comparada com as outras utilizadas neste trabalho. Ela tem a peculiaridade de apenas utilizar filtros 3x3 (Géron, 2021).

## “Xception”

Por ser bem maior que a VGG16, apresentou-se na Tabela 2 apenas algumas camadas da “Xception”. Tabela 2: Camadas da Xception

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| No | Nome | No | Nome | No | Nome | No | Nome |
| 0 | InputLayer | 40 | Sep.Conv2D | 80 | Sep.Conv2D | 120 | Sep.Conv2D |
| 1 | Conv2D | 41 | BatchNorm. | 81 | BatchNorm. | 121 | BatchNorm. |
| 2 | BatchNorm. | 42 | Activation | 82 | Activation | 122 | Conv2D |
| 3 | Activation | 43 | Sep.Conv2D | 83 | Sep.Conv2D | 123 | MaxPooling2D |
| 4 | Conv2D | 44 | BatchNorm. | 84 | BatchNorm. | 124 | BatchNorm. |
| 5 | BatchNorm. | 45 | Add | 85 | Add | 125 | Add |
| 6 | Activation | 46 | Activation | 86 | Activation | 126 | Sep.Conv2D |
| 7 | Sep.Conv2D | 47 | Sep.Conv2D | 87 | Sep.Conv2D | 127 | BatchNorm. |
| 8 | BatchNorm. | 48 | BatchNorm. | 88 | BatchNorm. | 128 | Activation |
| 9 | Activation | 49 | Activation | 89 | Activation | 129 | Sep.Conv2D |
| 10 | Sep.Conv2D | 50 | Sep.Conv2D | 90 | Sep.Conv2D | 130 | BatchNorm. |

Fonte: Dados originais da pesquisa

A “Xception” tem como suas principais características a realização de convoluções separáveis e a criação de atalhos entre camadas anteriores e posteriores. Para entender melhor este conceito, definamos algumas camadas desta CNN que ainda não citadas neste trabalho.

A “SEP.CONV2D” é a camada responsável por realizar as convoluções separáveis. Essas convoluções são divididas em convoluções profundas e convoluções pontuais. As convoluções profundas aplicam filtros separadamente em cada canal de entrada (RGB). Após isso, a convolução pontual é aplicada combinando todas as saídas da convolução profunda. Esta técnica reduz o número de parâmetros necessários nas convoluções (Géron, 2021).

A camada “Add” serve para criar atalhos que enviam informações de camadas iniciais para camadas mais profundas. Isto é necessário pois verificou-se que a medida que as CNNs ficavam mais profundas, seus desempenhos diminuíam. A solução encontrada para este problema foi adicionar às camadas profundas dados de entrada vindos das camadas iniciais, as quais possuíam dados mais próximos dos valores de entrada.

Por fim, ressalta-se a camada “BatchNormalization”. Esta camada normalmente aparece antes das camadas de ativação. Ela serve para normalizar os valores de entrada na função de ativação a partir da média e desvio padrão destes valores. *input−média*

# inputnormalizado=desvio padrão (2)

## “ResNet101”

A “ResNet” conta com uma peculiaridade: as “skip connections” ou “shortcuts connections”. Estas conexões são como os atalhos apresentados anteriormente. Inicialmente, uma CNN tem o trabalho de prever o valor de uma função “h(x)”. Uma vez, porém, que o valor de entrada é inserido na saída, o objetivo do preditor será prever o valor “f(x) = h(x) – x”. Isto se chama aprendizado residual, por isso o nome “ResNet” (Géron, 2021).

A Figura 11 apresenta uma comparação entre o aprendizado residual e o aprendizado convencional.

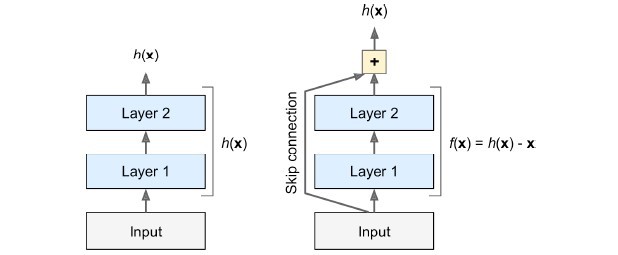


Figura 11: Aprendizado convencional (esquerda) e residual (direita)

Fonte: Géron (2021)

Este conjunto de “skip connections” são chamadas de unidades residuais. Estas unidades são o que existe nas camadas profundas da “ResNet”. A Figura 12 apresenta a arquitetura de uma ¨ResNet”.

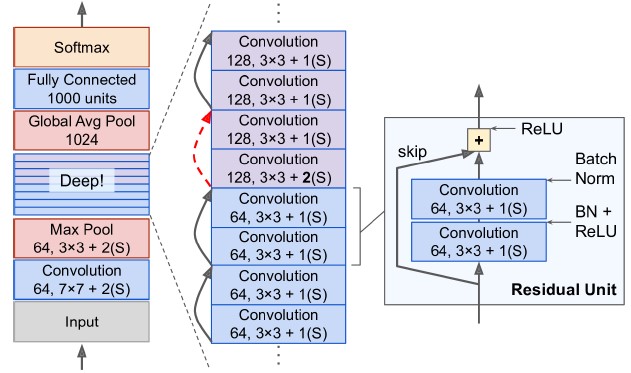


Figura 12: Arquitetura da “ResNet”

Fonte: Géron (2021)

Uma peculiaridade da arquitetura da “ResNet” é a existência de camadas de “Dropout”. Estas camadas desativam conexões entre neurônios da rede neural de forma aleatória. O objetivo desta desativação é a redução do “overfitting”.

## Resultados e Discussão

Nesta seção serão apresentados os resultados obtidos da aplicação das CNN. Conforme foi dito inicialmente, a aplicação da YOLO foi apenas com o objetivo de se conhecer o caminho a ser percorrido. Desta forma, não serão apresentadas métricas da aplicação da YOLO, apenas algumas imagens resultados.

Como resultados da aplicação da VGG16, Xception e ResNet serão apresentadas a matriz de confusão (precisão, sensibilidade, “f1-score” e acurácia) e os gráficos de perdas e de acurácias ao decorrer dos treinamentos. Antes de analisarmos os significados das métricas vejamos os valores de saídas de uma matriz de confusão.

* TP: “true positives”, imagem real era o objeto a detectar e a CNN identificou a existência do objeto;
* FP: “false positives”, imagem real não era o objeto a detectar e a CNN identificou a existência do objeto;
* TN: “true negatives”, imagem real não era o objeto a detectar e a CNN não identificou a existência do objeto;e
* FN: “false negatives”, imagem real era o objeto a detectar e a CNN não identificou a existência do objeto.

Dada as definições acima, a precisão é definida como:

*TP*

# precisão= (3)

*TP*+*FP*

A precisão é o percentual de acerto da CNN dentro de suas previsões. Vejamos a definição de sensibilidade.

*TP*

# sensibilidade= (4)

*TP*+*FN*

A sensibilidade é o percentual de acerto da CNN dentro dos dados analisados. Para entender o significado destas métricas vejamos um exemplo dado por Géron (2021). Suponhamos que queiramos treinar uma CNN para identificar vídeos seguros para crianças. Provavelmente desejaríamos que esta CNN rejeite muitos vídeos bons (baixa sensibilidade) e mantenha apenas vídeos seguros (alta precisão).

Ainda segundo Géron (2021), é importante combinar as métricas de precisão e sensibilidade em uma única métrica. Esta combinação é realizada por meio da métrica “f1score”. Esta métrica é definida pela média harmônica entre precisão e sensibilidade, desta forma, o “f1-score” será alto apenas quando precisão e sensibilidade forem altos.

2

*“f* 1−*score”*=

(5) *precisão sensibilidade*

1

+

1

Por fim, tem-se a acurácia. Esta métrica é definida como:

# acurácia= TP+TN (6)

*TN*+*FN*+*FP*+*TP*

A acurácia é a medida de acerto total de um classificador, é a quantidade de acertos pela quantidade total de previsões.

## YOLO

Apenas a critério de informação, apresenta-se abaixo a Figura 13 e a Figura 14.

Estas imagens foram dados de saída da YOLO após seu treinamento através do “framework” “Darknet”.



Figura 13: Imagem oriunda da YOLO (previsão correta)

Fonte: Resultados originais da pesquisa



Figura 14: Imagem oriunda da YOLO (previsão errada)

Fonte: Resultados originais da pesquisa

## VGG16

Abaixo apresenta-se a Tabela 3. Nesta tabela pode-se ver as matrizes de confusão de cada uma das classes de imagens. Abaixo de cada matriz de confusão há as métricas daquela classe.

Tabela 3: Matriz de confusão e métricas da VGG16

|  |  |  |  |
| --- | --- | --- | --- |
| Aviões | | Carros | |
| **(TP)** 503 | **(FP)**  3 | **(TP)** 517 | **(FP)**  3 |
| **(FN)** 10 | **(TN)** 136 | **(FN)** 0 | **(TN)** 132 |
| Acurácia | 98% | Acurácia | 99% |
| Precisão | 98% | Precisão | 98% |
| Sensibilidade | 93% | Sensibilidade | 100% |
| “f1-score” | 95% | “f1-score” | 99% |
| Drones | | Navios | |
| **(TP)** 436 | **(FP)** 10 | **(TP)** 482 | **(FP)**  2 |
| **(FN)** 7 | **(TN)** 199 | **(FN)** 1 | **(TN)** 167 |
| Acurácia | 97% | Acurácia | 99% |
| Precisão | 95% | Precisão | 99% |
| Sensibilidade | 97% | Sensibilidade | 99% |
| “f1-score” | 96% | “f1-score” | 99% |

Fonte: Resultados originais da pesquisa

Os resultados foram considerados satisfatórios, sendo todos eles bem altos. Ressalta-se que as menores métricas obtidas pela VGG16 foram nas classes avião e drone. Este resultado abaixo da média nestas duas classes faz sentido, pois ambas podem possuir algumas semelhanças (planos de fundo parecidos, por exemplo) que geram certa confusão na CNN. Ressalta-se que embora estes resultados tenham sido abaixo da média das outras classes, ainda assim eles estão bem altos de forma absoluta.

As Figuras 15 e 16 apresentam o gráfico de perdas e de acurácias durante o treinamento (dados de treinamento e validação foram testados nesta fase). As perdas representam a diferença entre os valores preditos e os valores reais durante a fase de treinamento.

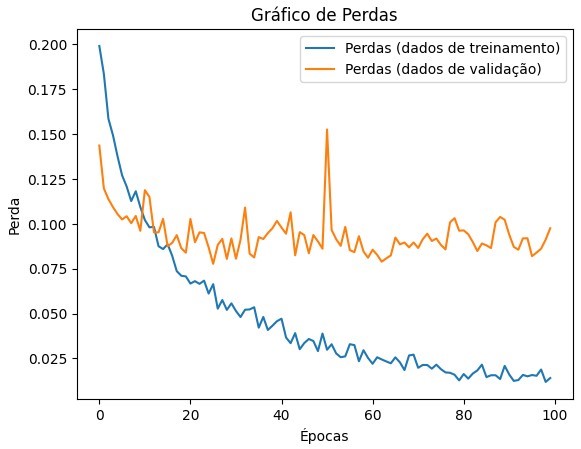


Figura 15: Gráfico de perdas da VGG16

Fonte: Resultados originais da pesquisa

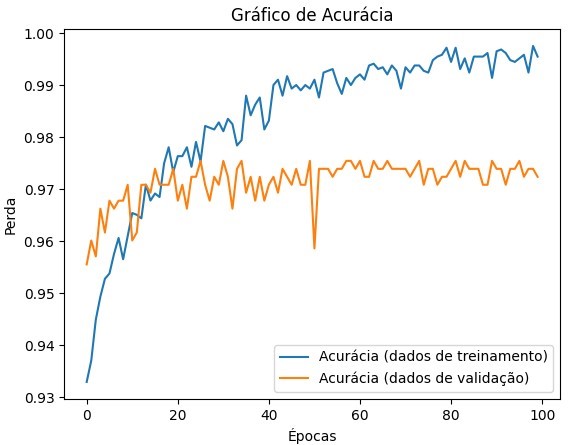


Figura 16: Gráfico de acurácias da VGG16

Fonte: Resultados originais da pesquisa

Nas figuras apresentadas acima, percebe-se o fenômeno de “overfitting”, pois a partir aproximadamente da época 20, embora as perdas e acurácias nos dados de treinamento apresentem resultados melhores, nos dados de validação estas métricas se mantém praticamente constantes.

## Xception

Abaixo apresenta-se a Tabela 4. Nesta tabela pode-se ver as matrizes de confusão de cada uma das classes de imagens. Abaixo de cada matriz de confusão há as métricas daquela classe.

Tabela 4: Matriz de confusão e métricas da Xception

|  |  |  |  |
| --- | --- | --- | --- |
| Aviões | | Carros | |
| **(TP)** 519 | **(FP)** 3 | **(TP)** 490 | **(FP)** 1 |
| **(FN)** 2 | **(TN)** 128 | **(FN)** 0 | **(TN)** 161 |
| Acurácia | 99% | Acurácia | 99% |
| Precisão | 98% | Precisão | 99% |
| Sensibilidade | 98% | Sensibilidade | 100% |
| “f1-score” | 98% | “f1-score” | 100% |
| Drones | | Navios | |
| **(TP)** 445 | **(FP)** 2 | **(TP)** 495 | **(FP)** 1 |
| **(FN)** 5 | **(TN)** 200 | **(FN)** 0 | **(TN)** 156 |
| Acurácia | 99% | Acurácia | 99% |
| Precisão | 99% | Precisão | 99% |
| Sensibilidade | 100% | Sensibilidade | 100% |
| “f1-score” | 100% | “f1-score” | 100% |

Fonte: Resultados originais da pesquisa

As Figuras 17 e 18 apresentam os gráficos de perda e acurácia durante o treinamento da Xception para os dados de treinamento e validação.

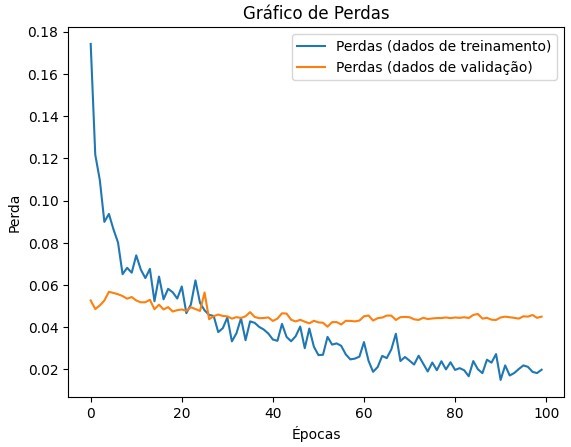


Figura 17: Gráfico de perdas da Xception

Fonte: Resultados originais da pesquisa

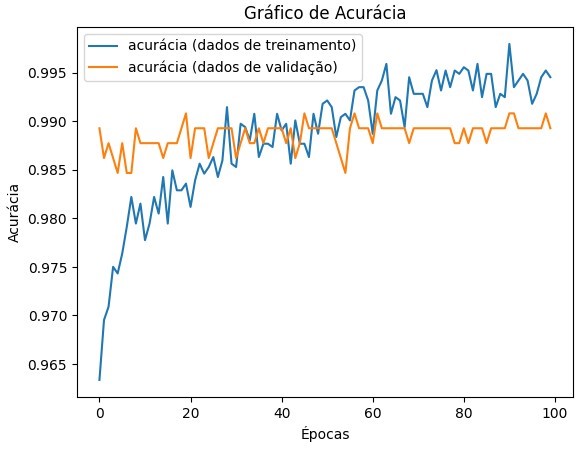


Figura 18: Gráfico de acurácias da Xception

Fonte: Resultados originais da pesquisa

Analisando os gráficos acima, percebe-se que a partir aproximadamente da época 30 do treinamento tanto as acurácias quanto as perdas nos dados de validação se mantém constante, enquanto que nos dados de treinamento elas apresentam melhora. Isso é um sinal de “overfitting”. A Xcepion está capturando algumas “features” bem específicas dos dados de treinamento e não está generalizando.

## ResNet

Abaixo apresenta-se a Tabela 5 com a matriz de confusão da ResNet101.

Tabela 5: Matriz de confusão e métricas da ResNet101

|  |  |  |  |
| --- | --- | --- | --- |
| Aviões | | Carros | |
| **(TP)** 453 | **(FP)** 47 | **(TP)** 474 | **(FP)** 38 |
| **(FN)** 66 | **(TN)** 86 | **(FN)** 16 | **(TN)** 124 |
| Acurácia | 83% | Acurácia | 92% |
| Precisão | 65% | Precisão | 77% |
| Sensibilidade | 57% | Sensibilidade | 89% |
| “f1-score” | 60% | “f1-score” | 82% |
| Drones | | Navios | |
| **(TP)** 390 | **(FP)** 60 | **(TP)** 483 | **(FP)** 11 |
| **(FN)** 15 | **(TN)** 187 | **(FN)** 59 | **(TN)** 99 |
| Acurácia | 88% | Acurácia | 89% |
| Precisão | 76% | Precisão | 90% |
| Sensibilidade | 93% | Sensibilidade | 63% |
| “f1-score” | 83% | “f1-score” | 74% |

Fonte: Resultados originais da pesquisa

As Figuras 19 e 20 apresentam os gráficos de perda e acurácia.

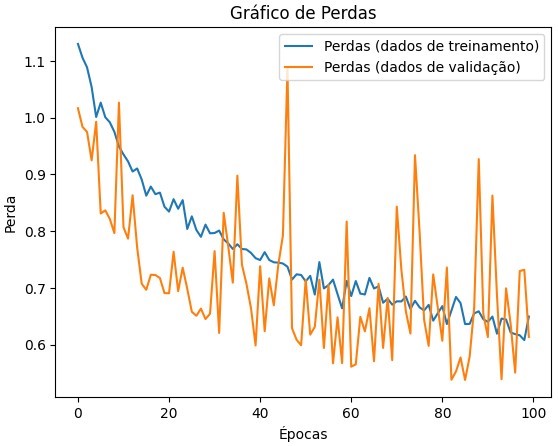


Figura 19: Gráfico de perdas da ResNet101

Fonte: Resultados originais da pesquisa

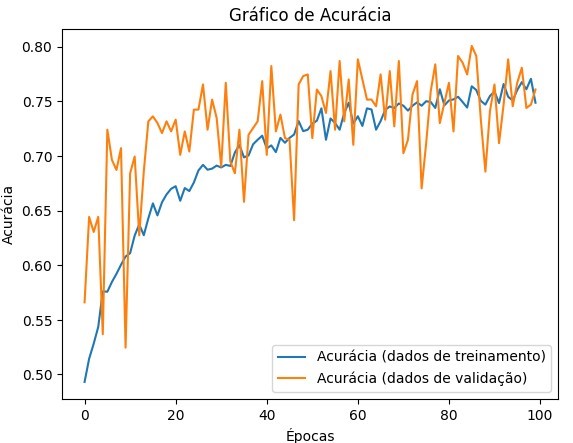


Figura 20: Gráfico de acurácia da ResNet101

Fonte: Resultados originais da pesquisa

Analisando os gráficos acima e os resultados obtidos pelas métricas oriundas da matriz de confusão, supõe-se que a principal causa para estes resultados seja a quantidade de épocas. Os resultados não apresentaram convergência significativa, devendo-se, portanto, aumentar o número de épocas.

## Considerações Finais

Das três redes neurais convolucionais implementadas neste trabalho, a VGG16 e Xception apresentaram bons resultados nos dados de validação, enquanto que a ResNet101 não apresentou um resultado satisfatório.

A VGG16 e a Xception apresentaram o fenômeno de “overfitting”. A aparição deste fenômeno indica que houve um gasto computacional desnecessário, pois a partir de determinada época estas CNNs aprendiam padrões bem específicos nos dados de treinamento, mas não conseguiam generalizar este aprendizado para dados de teste.

Duas soluções poderiam ser utilizadas para este caso. A primeira, levando em conta os elevados níveis de acurácia obtidos pela VGG16 e Xception, seria simplemente reduzir o número de épocas de treinamento da rede para evitar um gasto computacional desnecessario (sem a realização de alterações na arquitetura da rede). A segunda solução seria a alteração de alguns parâmetros da rede tais como:

* Taxa de aprendizado (“learning rate”);
* Mudança do otimizador utilizado tais como: Adam, RMSprop, Adagrap ou Adadelta;
* Mudança nas últimas camadas alteradas no processo de “fine-tuning” (incluindo ou retirando camadas); e
* Alteração da quantidade de camadas pré-treinadas congeladas na seguda fase do treinamento.

Quanto a ResNet101, quando se analisam os gráficos de perdas e de acurácias, observa-se uma grande variação nos dados de validação quando comparados com os dados de teste. Uma possível solução para este caso, como já dito, seria o aumento do número de épocas. Um ponto importante a se analisar, porém, é a viabilidade da utilização da ResNet101 para essa aplicação já que tanto a VGG16 quando a Xception apresentaram bons resultados e um aumento do número de épocas significaria um aumento do custo computacional.

## Agradecimento

Agradeço a Deus pela força concedida para a realização deste trabalho. Que as habilidades aprendidas aqui sejam úteis para Seu Reino. Agradeço também à Fernanda, minha esposa, pelo apoio incondicional a todo instante.

## Referências

Evanson, NIck. 2020. Explainer: What Are Tensor Cores. TechSpot, Estados Unidos da América. Disponível em: https://www.techspot.com/article/2049-what-are-tensor-cores/. Acesso em: 18, abril 2023.

Géron, Aurelien. 2021. Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow. 2ed. O’Reilly, Sebastopol, CA, USA.

Joshi, Prateek. 2015. Opencv with python by examples. 1ed. Packt Publishing. Birmingham, Reino Unido.

Khan, Rizwan. 2019. Ilustração da Arquitetura da VGG16. Disponível em: https://www.researchgate.net/figure/An-illustration-of-VGG16-architecture38\_fig12\_329413866. Acesso em: 18, abril 2023.

Lamb, Thomas. 2003. Ship Design and Construction. 1ed. Andrew MacBride. Estados Unidos da América

Milano, D. Honorato, L.B. 2010. Visão Computacional. Universidade Estadual de Campinas, Faculdade de Tecnologia. Campinas, São Paulo, Brasil. Disponível em: https://d1wqtxts1xzle7.cloudfront.net/35825905/2010\_IA\_FT\_UNICAMP\_visaoComputaciona l-libre.pdf?1417700841=&response-content-disposition=inline%3B+filename

%3DVISAO\_COMPUTACIONAL\_Palavras\_Chaves.pdf&Expires=1692832812&Signature=M 22S1zhfgmgGxVmecB~1XP1ERaYij3HpYTGSKhYcbz4SAPz7dm4nBu~ObF4dujMq2X1woS d~ofRi~asohiPM927JOTfCsj~qqiyzaki5~gTgrT8lrdgKTDmciKv2kl4ZOMfBDhCmor9jOoaMS7 LHlHSIJYTRaaqnVCIYdiFQ74ZchXLPyPG79i1FIUZANRQS459oi1leavDWIkAeHFeS98padK0YMYbjKEvUjDcDE

QVRWNmnk6a2dCEG7ZxW0XRLTKxdsxv1fd6MP445yIop0RtsTAmzAzbvK3T2C8n2gE2WJ G6N5cesOY-yIiyjVqCoZmSN9x0ysH5NPM0Qjj3Bg\_\_&Key-PairId=APKAJLOHF5GGSLRBV4ZA. Acesso em 23, agosto 2023.

Ozel, Mehdi. 2019. Drone Dataset (UAV). Disponível em: https://www.kaggle.com/datasets/dasmehdixtr/drone-dataset-uav. Acesso em: 18, abril 2023.

Rampurawala, Abduttayyeb. 2021. multiclass-image-dataset-airplane-car-ship. Disponível em: https://www.kaggle.com/datasets/abtabm/multiclassimagedatasetairplanecar. Acesso em: 18, abril 2023.

Redmon, Joseph; Divvala, Santosh; Girshick, Ross; Farhadi, Ali. 2015. You Only Look Once: Unified, Real-Time Object Detection. Estados Unidos da América. Disponível em https://arxiv.org/pdf/1506.02640.pdf. Acesso em: 18, abril 2023.

Rosebrock, Adrian. 2021. Convolutional Neural Networks (CNNs) and Layer Types. PyImageSearch, Estados Unidos da América. Disponível em https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/. Acesso em: 18, abril 2023.

Rosebrock, Adrian. 2017. Deep Learning for Computer Vision with Python. 1ed. PyImageSearch. Estados Unidos da América.

Raj, Bharath. 2018. Data Augmentation: How to Use Deep Learning When You Have Limited Data. KDNuggets, Estados Unidos da América. Disponível em https://www.kdnuggets.com/2018/05/data-augmentation-deep-learning-limited-data.html. Acesso em: 18, abril 2023.

## Apêndice

Os códigos implementados nas três CNNs utilizadas foram os mesmos, com apenas algumas pequenas especificidades cada um. Desta forma, apenas um será apresentado para eventual análise: <https://github.com/GMSantos4/CV/blob/main/main_vgg16.ipynb>.