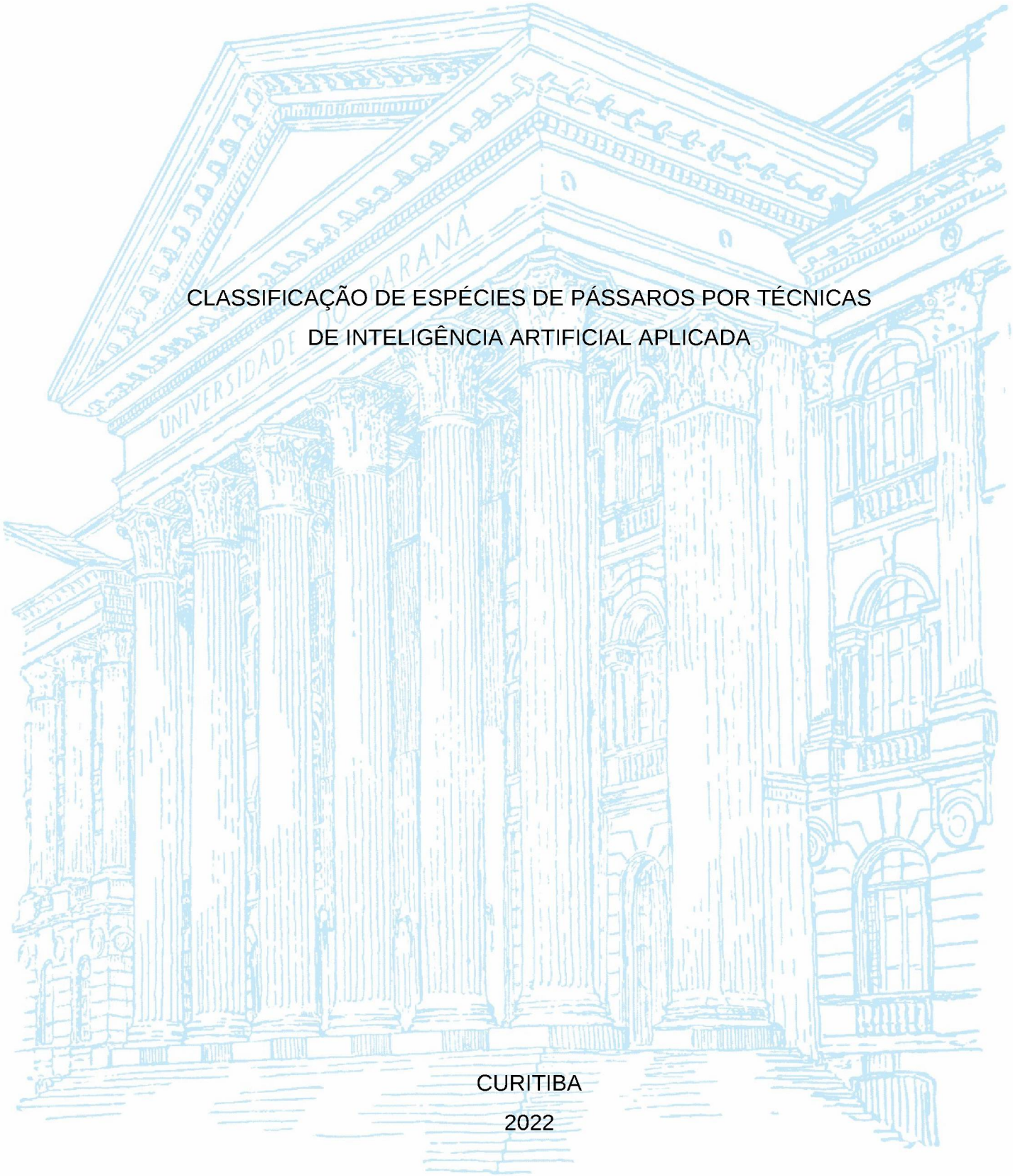
UNIVERSIDADE FEDERAL DO PARANÁ

ANTONIO CARLOS PERON JÚNIOR



CLASSIFICAÇÃO DE ESPÉCIES DE PÁSSAROS POR TÉCNICAS DE INTELIGÊNCIA ARTIFICIAL APLICADA

CURITIBA

2022

ANTONIO CARLOS PERON JÚNIOR

CLASSIFICAÇÃO DE ESPÉCIES DE PÁSSAROS POR TÉCNICAS DE INTELIGÊNCIA ARTIFICIAL APLICADA

Trabalho de Conclusão de Curso apresentado ao curso de Pós-Graduação em Inteligência Artificial Aplicada, Setor de Educação Profissional e Tecnológica, Universidade Federal do Paraná, como requisito parcial à obtenção do título de Especialista em Inteligência Artificial Aplicada.

Orientador: Prof. Dr. Jaime Wojciechowski

CIDADE

2022

MINISTÉRIO DA EDUCAÇÃO
SETOR DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
CURSO DE PÓS-GRADUAÇÃO INTELIGÊNCIA ARTIFICIAL APLICADA - 40001016348E1

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação INTELIGÊNCIA ARTIFICIAL APLICADA da Universidade Federal do Paraná foram convocados para realizar a arguição da Monografia de Especialização de **ANTONIO CARLOS PERON JUNIOR** intitulada: **Classificação De Espécies De Pássaros Por Técnicas De Inteligência Artificial Aplicada**, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua Aprovação no rito de defesa.

A outorga do título de especialista está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 16 de Novembro de 2022.

JAIME WOJCIECHOWSKI
Presidente da Banca Examinadora

RAFAELA MANTOVANI FONTANA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Classificação de espécies de pássaros por técnicas de Inteligência Artificial Aplicada

Antonio Carlos Peron Júnior
Setor de Educação Profissional e Tecnológica
Universidade Federal do Paraná
Curitiba, Brasil
antonio.peron@ufpr.br

Jaime Wojciechowski
Setor de Educação Profissional e Tecnológica
Universidade Federal do Paraná
Curitiba, Brasil
jaimewo@ufpr.br

***Resumo*— A observação de aves no seu ambiente natural, é o hobby de parte da população. Ela pode ser feita como forma de atividade recreacional ou com uma finalidade de estudos. A principal difuldade desta prática é a identificação de qual ave foi visualizada. Neste documento fez-se um estudo de classificação de imagens de aves e foi criado um modelo de IA, que pode ser usado como base de um software. São apresentados diversos gráficos comparando três implementações de IA. O framework utilizado foi o Tensorflow, da empresa Google Brain. Nele há diversas implementações, sendo que foi escolhida uma, a MobileNet. No momento da elaboração deste artigo, foram encontradas três versões, sendo a MobileNetV1, MobileNetV2 e a MobileNetV3. As técnicas comparadas se baseiam no conceito conhecido como Convolutional Neural Network (CNN). As três versões do Mobilenet foram comparadas, utilizando as principais métricas de desempenho mais uma comparação de tempo de execução do script. Entre as medidas de precisão, houve uma diferença de até dois pontos percentuais. Entretanto, no tempo de execução, houve uma diferença de 56% entre o MobileNetV1 e MobileNetV3.**

Palavras-Chave—birdwatching, CNN, tensorflow, métricas comparação, mobilenet

Abstract - Bird watching in their natural environment is a hobby for part of the population. It can be done as a form of recreational activity or for a study purpose. The main difficulty of this practice is the identification of which bird was seen. In this document, a study of bird image classification was carried out and an AI model was created, which can be used as a basis for a software. Several graphs are presented comparing three AI implementations. The framework used was Tensorflow, from Google Brain. In it there are several implementations, and one was chosen, MobileNet. At the time of writing this article, three versions were found, being MobileNetV1, MobileNetV2 and MobileNetV3. The compared techniques are based on the concept known as Convolutional Neural Network (CNN). The three versions of Mobilenet were compared, using key performance metrics plus a script runtime comparison. Among precision measures, there was a difference of up to two percentage points. However, at runtime, there was a 56% difference between MobileNetV1 and MobileNetV3.

Keywords—birdwatching, CNN, tensorflow, metrics comparison, mobilenet

I. DESENVOLVIMENTO

O vínculo e as interações do ser humano com a vida selvagem sempre existiu. Após a aquisição da linguagem, da evolução da comunicação e organização social, aliado ao desenvolvimento tecnológico, a interação da maioria da população humana com a vida selvagem reduziu, mostrando-se através de fins recreativos como o turismo, ou de forma educativa. Notando este distanciamento, uma parcela da população demonstra interesse em fazer observações da fauna silvestre, com um destaque a observação de aves [1], principalmente pois estas tem menos restrições de deslocamento e de espaço físico. Uma parcela considerável de aves reside na parte urbana das aglomerações humanas.

Com esta ideia em mente, a promoção da Educação Ambiental ajusta-se perfeitamente. Por isto estão sendo feitos incentivos ao turismo de observação de aves (*Birdwatching*) e a Educação Ambiental, no âmbito escolar [2].

Como a prática de *Birdwatching* pode ser feita inclusive em ambientes urbanos, nota-se uma dificuldade entre os não técnicos em identificar de qual espécie (ou raça) de ave pertence o espécime vislumbrado. Um dos motivos desta dificuldade, além do parco conhecimento de biologia e ciências afins, é que a biodiversidade da avifauna mundial é de aproximadamente 12.000 espécies de aves. Outra dificuldade de se identificar corretamente qual raça de ave, inclusive entre os profissionais da área, pode ser descrita ao avistar o pássaro *Crested auklet* e confundi-lo com o *Parakeet auklet [3]*. Enquanto o primeiro tem o peito mais escuro e uma penugem na cabeça, o segundo não possui penugem na cabeça e o seu peito é mais claro [3]. Dependendo da cena vista, estas características podem não ter sido evidenciadas, dificultando a correta classificação da raça.

Figura 1 - *Crested auklet*

Figura 2 - *Parakeet auklet*

Para o desenvolvimento deste projeto, foi elaborada uma linha de desenvolvimento que inicia com os objetivos onde é abrangida a ideia central do trabalho. A descrição deste item está contiga na seção A. Objetivos. Para compreender os conceitos e teorias, que são a base para o desenvolvimento deste projeto, elaborou-se a seção B. Embasamento Teórico. Esta está dividida em três sub-itens, sendo: 1) Aprendizado de Máquina e Machine Learning, 2) Redes Neurais e Redes Neurais Convolucionais e 3) Tecnologia Utilizada. Em seguida, a seção C. Base de Dados, descreve a procedência da base de dados e o seu conteúdo. Imediatamente após, há o item D. Métodos, onde há uma descrição da metodologia utilizada para o experimento. Logo depois há o item E. Tecnologias, onde há detalhes dos equipamentos e softwares utilizados. Por último, há o tópico II. Resultados e Dicussões, sendo subdividido em 1) MobilenetV1, 2) MobilenetV2, 3) MobilenetV3. Neste campo, estão os resultados obtidos com a análise dos experimentos e as comparações entre os itens deste tópico.

A. Objetivos:

No delineamento deste trabalho, aborda-se a identificação de pássaros, por meio de imagens. Tendo esta abordagem em mente, temos um *range* de 400 (quatrocentas) espécies de pássaros e uma fotografia de uma ave. O interesse é associar, de forma acertiva, a imagem com a espécie de ave. A ideia é aplicar técnicas de Inteligência Artificial e verificar qual espécie a ave fotografada mais se aproxima. Há vários desafios para se efetuar este tipo de classificação. Dentre eles é a variabilidade de fundo, a iluminação que a imagem foi capturada e a pose da ave. Isto se deve pois a maioria das imagens foram coletadas no habitat natural dos pássaros. Posto isto, não é possível controlar a rotação, ângulo e escala de visão no momento da aquisição. Neste tipo de situação, o impasse de se identificar corretamente impõe dificuldades até entre profissionais e especialistas humanos. Neste contexto, é comum ocorrer que as características distintas entre espécies não fiquem evidentes. Nesta situação, é necessário aglutinar diversas características diferentes e buscar um acordo para apontar uma ou outra espécie.

Este documento implementa três técnicas de Inteligência Artificial – todas embasadas em Redes Neurais Artificais Convolucionais, CNN – para prever qual ave aparece em determinada imagem. A API utilizada é o framework Tensorflow – Keras. As técnicas descritas foram aplicadas utilizando a linguagem de programação Python em uma base de dados pública, conforme detalhado a seguir. Os procedimentos e códigos foram executados em um *notebook*, descrito no item de Tecnologias.

Neste trabalho, foi utilizado a arquitetura MobileNet que pode ser usada em vários casos de uso. Esta tecnologia apresenta modelos eficientes para ser utilizada em dispositivos com recursos limitados, como sistemas móveis ou embarcados. Dependendo do caso de uso, ele pode usar vários tamanhos de camada de entrada e diferentes fatores de largura. Isso permite que distintos modelos de largura reduzam o número de multiplicações e, assim, reduzam o custo de inferência em dispositivos móveis. MobileNets suportam qualquer tamanho de entrada maior que 32 x 32, com tamanhos de imagem maiores oferecendo melhor desempenho. O número de parâmetros e o número de multiplicações podem ser modificados usando o alphaparâmetro, que aumenta/diminui o número de filtros em cada camada. Alterando o tamanho e o alphaparâmetro da imagem, todos os 16 modelos do papel podem ser construídos, com pesos ImageNet fornecidos.

B. Embasamento Teórico:

1) Aprendizado de Máquina e Machine Learning

Conforme [4], a Inteligência Artificial (IA), como parte da Ciência da Computação, vela pela automatização dos comportamentos inteligentes que possuem uma base teórica e prática. Esta base se traduz em uma estrutura de dados, algoritmos e técnicas de programação para chegar a um resultado. A IA tem duas inquietações: representação do conhecimento e a sua busca. Ou seja, capturar em linguagem formal a situação para a posterior manipulação.

Segundo [5], aprendizado de máquina abrange várias áreas da ciência, como neurobiologia, teoria de controle, estatística, psicologia dentre outras áreas do conhecimento.

O aprendizado de máquina é um sub-conjunto de IA. Resumindo o que [4] conta sobre aprendizado de máquina, o aprendizado remete a modelos mentais da inteligência humana. A robótica ajudou a transportar parcialmente esta teoria para o mundo computacional, através de sensores e heurísticas. De acordo com [6], é a simplificação do espaço de busca da solução de um problema, ocasionado pelo conhecimento prévio de determinado problema.

Há duas classes de aprendizado, para [4]: o aprendizado supervisionado e o aprendizado não supervisionado. Como [9] explica, o aprendizado supervisionado faz parte de aprendizado indutivo e há exemplos categorizados, que respeitam um modelo de conhecimento que deseja-se construir. Continuando o raciocínio, [6] comenta que o aprendizado não supervisionado, os algoritmos construídos agrupam os dados disponibilizados de acordo com a similaridade encontrada.

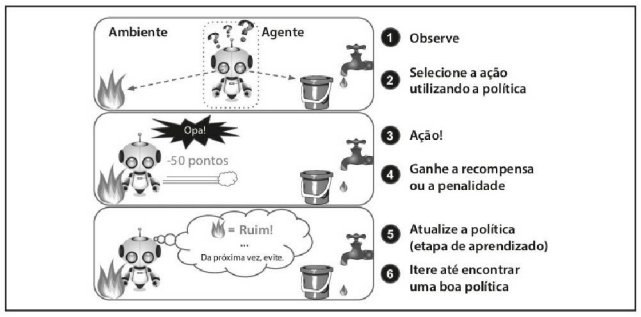


Figura 3 – Aprendizado por reforço [7]

Em [7], além das classes de aprendizado indicados, afirma que há o Aprendizado por Reforço e o Aprendizado Online e em Lote. O aprendizado por Reforço, o agente (sistema de aprendizado) observa o modelo, seleciona ações para a obtenção de recompensas (positivas) ou penalidades (negativas). Com este comportamento, o sistema aprende qual a melhor estratégia (ou política) que recebe o maior número de recompensas. A Figura 1, exemplifica este comportamento.

Já o aprendizado em Lote, é um sistema treinado com um lote de dados e não aprende de forma incremental. Caso seja necessário acrescentar novos dados, o sistema deve ser treinado novamente e colocado em produção. Para facilitar o processo de aumento da base de conhecimento, automatizou-se o processo e ele ficou conhecido como Aprendizado Online, segundo [6].

Segundo [4], há cinco formas de aprendizado de máquina, os quais são: simbólico, conexionista, genético e o emergente e, por último, o probabilístico. Szegedy at. al. [9] chama o algoritmo genético e emergente de evolucionário e o probabilístico de estatístico.

Comentado por [4], no aprendizado simbólico há um conjunto de dados disponíveis, onde os algoritmos conseguem fazer uma generalização aceitável. Neste tipo de aprendizado são feitas diversas atividades, onde é feito uma espécie de aprendizado indutivo. Neste tipo de aprendizado, os espaços de busca tendem a ser de grandes proporções, pois as atividades normalmente fracionam os dados fornecidos em porções menores. Já para [6] o aprendizado simbólico faz uma representação de conceitos baseado na análise de exemplos, efetuando a representação por meio de expressões lógicas, árvores, redes semânticas, dentre outros meios.

No aprendizado conexionista, também conhecidos como modelos neurais ou processamento paralelo distribuído (PDP, do inglês *Parallel Distributed Processing*) as atividades são distribuídas por meio de um conjunto de camadas de neurônios. Desta maneira, há um ajuste fornecido pelas camadas mencionadas. Como o processamento é distribuído por meio de nós e camadas, estes sistemas tendem a degradar suavemente. Neste modelo, a representação dos parâmetros de entrada quanto dos de saída é muito forte. Ou seja, o projetista deve criar um esboço de mundo no formato numérico. Para a rede ter um sucesso no seu objetivo, a devida codificação faz diferença para o aprendizado da rede de neurônios. O processamento conexionista ocorre pois os dados são vetores/matrizes e a conexão entre neurônio e outro são operações numéricas, por exemplo multiplicação entre matrizes. Estas escolhas são conhecidas como viés indutivo do sistema. Caso uma rede de neurônios seja devidamente programada, ela acaba sendo treinada ou condicionada e não programada explicitamente pelo programador do sistema. Em diversas vezes, as redes capturam invariâncias do mundo. Continuando com o raciocínio de [4], entre as tarefas que as redes neurais se adaptam bem são a classificação, reconhecimento de padrões, evocação de memória, predição, otimização e filtragem de ruído. Goldschmidt [6] foi mais direto ao conceituar, comentando que este modelo é inspirado no sistema biológico nervoso, utilizando modelos matemáticos simplificados.

O aprendizado genético e o emergente (de acordo com [4]) ou evolucionário (segundo [9]), também são baseados em modelos biológicos. O que diferencia do conexionista é que o primeiro possui uma evolução natural, que se assemelha a reprodução utilizando a genética para provocar a evolução. Luger [4] comenta que esta abordagem é baseada no molde de uma população de indivíduos, que sobrevive através daqueles que são considerados mais ajustados. Esta forma de aprendizado simula a forma mais alinhada e dominante que receberam um ajuste da natureza. Ainda seguindo o raciocínio de [4], este tipo de aprendizado separa o problema em três estágios. No primeiro estágio são colocados potenciais soluções individuais do escopo do problema. Estas representações são tão simples que não se pode desmembrar em soluções menores. No segundo estágio, são configurados algoritmos de mutação ou de acasalamento, onde as características do primeiro estágio são recombinadas. O terceiro estágio faz a separação entre os indivíduos gerados, separando os melhores candidatos dos menos adaptados. Os candidatos considerados mais adaptados são considerados como uma resposta para o problema apresentado.

O aprendizado estatístico (de acordo com [6]) ou probabilístico (conforme [8]), é o paradigma que utiliza métodos estatísticos (em sua maioria, os paramétricos), para localizar aproximações mais próximas do ideal, que o modelo de conhecimento desteja conduzindo, na visão de [9]. Em [4] é acrescentado que neste tipo de aprendizado de máquina os métodos Bayesianos predominam, sendo exemplos as redes Bayesianas dinâmicas (ou RBD's) e diversas extensões. São aplicados os modelos ocultos de Markov (ou MOM's).

Há um tipo de aprendizado de máquina citado apenas em [6], que é o baseado em exemplos. Este modelo na busca de casos similares, dentro da base, semelhantes ao que deve ser analizado. Usando este raciocínio, é possível efetuar uma dedução da saída do sistema. Este modelo é ideal, caso haja casos representativos estudados anteriormente.

2) Redes Neurais e Redes Neurais Convolucionais

Nesta seção são detalhadas as redes neurais e redes neurais convolucionais, pois é a teoria em que se baseiam as teconologias MobileNetV1, MobileNetV2 e MobileNetV3.

Goldschmidt [6] faz uma relação entre as Redes Neurais da ciência da computação com o cérebro biológico encontrado em organismos biológicos. No texto do autor, o cérebro, em especial o humano, é um processador altamente complexo e que consegue processar dades em forma paralela, de maneira eficiente. Atualmente não existe nenhum equipamento artificial que consiga realizar as mesmas atividades de maneira semelhante. Acredita-se que este comportamento está ligado à sua estrutura.

Segundo [5], a rede neural é uma imitação do cérebro que tem como objetivo executar tarefas pré-determinadas e/ou funções de interesse. Consdera-se imitação pelo motivo do cérebro ser mas complexo que um computador, com processamento não linear e paralelo.

Outro detalhe que chama a atenção é que muitas, senão todas, as decisões cerebrais são baseadas na aprendizagem, e este conhecimento fica mantido até outro o substituir. Esta característica deve ser possível em sistemas baseados na experiência e que podem ser direcionados para uma aplicação de interesse. Em [6], há a comparação entre uma rede neural artificial com a natural.

Modelo Natural	Modelo Artificial
Cérebro	RNA
Neurônio Biológico	Neurônio artificial/elementos processadores
Rede de Neurônios	Estrutura de camadas
10 Bilhões de Neurônios	Centenas/milhares de neurônios
Aprendizado	Aprendizado
Generalização	Generalização
Associação	Associação
Reconhecimento de Padrões	Reconhecimento de Padrões

Tabela I – Comparação Modelo Natural e Modelo Artificial, baseado na tabela 5.1, em [6]

A Figura 4, mostra uma analogia do neurônio natural como deve ser um neurônio artificial:

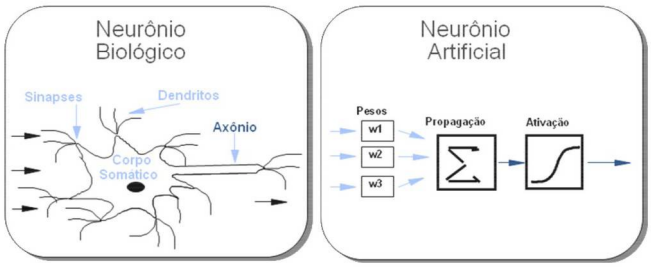


Figura 4 – Analogia neurônio Biológico/neurônio Artificial, retirado de [6], página 74

No momento que as redes neurais artificiais conseguem se aproximar do conceito proposto, [6] sugere que deve-se conseguir ter algumas características de processamento de dados, sendo:

- Generalização: Quando se observa os dados, para retirar informações, nota-se que há a existência de ruídos misturados. O algoritmo de aprendizado deve ser capaz de saber quando ignorar este ruído para a generalização ser possível;
- Abstração: A abstração está ligada à essência dos conceitos básicos, contidos na base de dados. Ou seja, saber localizar quais características são mais relevantes (ou com um peso maior) do que outras;
- Associação: As relações estabelecidas pelas RNA's, muitas vezes fogem das explicações racionais. Como por exemplo, reconhecer um indivíduo pelo som do caminhar do mesmo;
- Não Programáveis: As RNA's não são programadas diretamente, ou seja, elas recebem (na entrada) uma quantidade de dados e (a saída) deve brandir saídas corretas sem nenhuma intervenção do operador;
- Aprendizado por Experiência: Os dados são a grande fonte do conhecimento e para isso estes devem ser apresentados diversas vezes à rede;
- Robustez e Degradação Gradual: A robustez de uma RNA pode ser considerada o seu tamanho. Entretanto pode ocorrer a degradação gradual, pois alguns neurônios podem ser considerados inoperantes, quando deveriam ser mantidos;
- Soluções Aproximadas: As soluções propostas, em sua maioria, não são exatas. Sempre há um sobreajuste, podendo ser dado soluções erradas;
- Busca paralela e endereçamento pelo conteúdo: Ao contrário da configuração da arquitetura de Von Neumann, a experiência fica distribuída pela rede. Consequentemente, não há como fazer uma procura sequencial e sim necessita-se de uma procura paralela.

A figura 5 do ator [7] mostra a arquitetura, simplificada, de uma RNA:

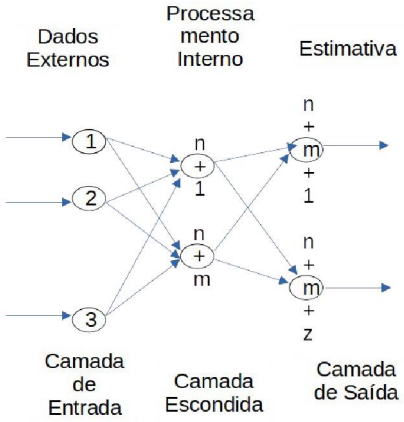


Figura 5 – Arquitetura simplificada da RNA. Fonte: adaptado de [6]

De acordo com a Figura 5, pode-se notar que cada neurônio recebe um nome (no caso, uma numeração). Normalmente inicia-se no canto superior esquerdo, em direção a parte inferior esquerda. Este sentido de numeração é mantido até esgotarem-se as camadas.

Cada entrada de neurônio é a saída de um anterior. Para cada entrada, é dado um peso (W_{ij}), gerando as ponderações. O somatório das entradas ponderadas, dá-se o nome de valor NET. Este valor pode chegar ao valor de ativação do neurônio (F). Caso isto ocorra, o neurônio será tido como ativo. Em caso contrário, é um neurônio inativo.

A partir deste ponto é interessante iniciar o trato dos elementos básicos dos neurônios artificiais, também conhecido como modelagem.

- Conexão entre os Processadores: Também conhecido como peso sináptico, trata-se do efeito que um neurônio faz sobre outro. A sua notação é $W_{ij,}$ onde i é a identificação do neurônio de saída e j o de chegada.

- Regra de Propagação: Corresponde a forma que os estímulos vindos de outros neurônios combinam-se aos pesos sinápticos correspondentes para compor o potencial de ativação de um neurônio.

  Pode-se resumir na seguinte equação:

$$net|_k = \sum w|_{ik} \times O|_i$$

Figura 5 - Exemplo de Regra de Propagação

- Função de Ativação: As redes neurais, por maior que seja a sua estrutura, são capazes de capturar relações lineares, entre as entradas e saídas. Para trazer o poder de modelagem nas relações não-lineares, os resultados de cada saída são processados pelas funções de ativação. Para tanto, existem várias funções de ativação, entre elas: Linear, Sigmoide, Softmax, ReLU, Leaky ReLU.

Em relação a arquitetura de uma rede neural, também conhecida como topologia, há duas vertentes:

- Não Recorrentes: Conhecidas como redes "sem-memória". As redes de uma camada, são conhecidas como Perceptrons.
- Recorrentes: Ao contrário das redes não recorrentes, esta tem um histórico do processamento anterior e elas tem algumas aplicações onde há a mensuração de tempo, possibilitando uma melhor previsão de futuro.

Braga et al [10], comenta sobre as conexões podem ter dois tipos:

1) *feedforward*, ou acíclica: é o tipo de de rede neural artificial com o design mais simples. Nela há uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída;

2) *feedback*, ou cíclica: é o tipo de rede neural que tem o ciclo de processamento semelhante ao da feedforward, porém há uma realimentação dos neurônios de entrada, com os dados gerados pelos neurônios de saída;

O processamento das redes neurais tem duas fases: Aprendizado ou treinamento, e Aplicação ou Recuperação da Informação.

A fase de treinamento da rede é o processo que gera os pesos sinápticos, que serão utilizados para a aquisição do conhecimento. Estes pesos são produzidos de acordo com os estímulos do meio externo a qual são espostos. A sequência de passos utilizada neste ponto, é a seguinte:

a) Estimulação da rede pela base de dados fornecida, também conhecida como *inputs;*
b) Alteração dos pesos sinápticos, previamente defnidos no algoritmo;
c) Resposta alternativa da rede neural, em resposta ao "novo" ambiente detectado.

Como estes passos podem ser repetidos indefinidamente, deve-se estabelecer um ou mais critérios de parada, como:

- Limitar o número de interações que devem ser feitas;
- Com base no erro produzido nas iterações, estabelece-se um limiar ideal para cada caso.

A fase de recuperação de informação depende da fase de treinamento. A fase de treinamento cria um modelo e a fase de recuperação cria uma saída com dados que não foram processados previamente. Portanto, nesta fase, os pesos e as redes não serão atualizados.

Segundo [6], as redes neurais são as que mais conseguem refletir a forma de pensamento humano. Para os casos onde as regras de funcionamento não estão claramente delineadas e estas devem ser retiradas de um conjunto de dados previamente conhecidos. Ou seja, a principal função de um conjunto de treinamento é fornecer uma saída consistente. Os conjuntos, de entrada ou saída, podem ser chamados de vetor. Avançando nesta forma de raciocínio, temos um vetor de entrada que serve de entrada para um algoritmo. Este algoritmo fará o trabalho de ajustar os pesos sinápticos. Com este trabalho, espera-se que os pesos convirjam para um valor, de acordo com a proposta situacional.

Braga et al [10] introduz os conceitos de classes de treinamento: Supervisionado, Não-supervisionado e por reforço.

No aprendizado supervisionado, as redes neurais, recebem entradas com um padrão conhecido e, consequentemente, com a saída esperada. Por causa desta situação, os pesos sinápticos são ajustados até que estejam de acordo com os limites de tolerância especificados pelo usuário. Simplificadamente, o funcionamento do algoritmo, nesta situação é o seguinte: a rede neural é aplicada no vetor de entrada e os pesos são gerados. O vetor de saída é calculado e comparado com o vetor alvo. Neste ponto é encontrado um erro, o qual realimenta o algoritmo até chegar nos parâmetros definidos.

No aprendizado não supervisionado, há a necessidade de se encontrar as propriedades que diferenciam os conjuntos de dados. Neste processo, os padrões são encontrados e re-apresentados à rede. As características que tem regularidade e redundância descobertas serão utilizadas neste tipo de aprendizado. Este tipo de aprendizado é muito utilizado nas tarefas de agrupamento de dados, já que envolve a descoberta de características relevantes nos dados de entrada.

O aprendizado por reforço é utilizado para tarefas de controle. Neste aprendizado, não há uma resposta desejada, mas ações boas ou ruins geradas na rede. Se um estado é considerado satisfatório, a rede é reforçada. Caso contrário, a ação é enfraquecida.

Há alguns modelos de redes neurais que são interessantes, pois foram as primeiras a serem projetadas e é possível aferir a evolução tecnológica, além de ser de fácil aferição das suas limitações. São elas: Perceptron de Camada Única, Adaline, Madaline, Perceptron de Múltiplas Camadas.

Segundo [6], a rede perceptron de camada única é o modelo mais antigo de rede neural. Ele serve para classificar (ou separar) o conjunto de dados em uma, de duas, classes. Estas redes são baseadas no modelo perceptron e todas tem estas características:

- Necessariamente usam o aprendizado supervisionado;
- Têm somente uma camada de pesos ajustáveis;
- Cada camada se conecta á próxima camada e não há caminho reverso (*feedforward*);
- A regra de propagação é necessariamente um produto escalar;
- As funções degrau são o padrão de ativação.

Este modelo de rede neural tem como aplicação problemas lineares, tendo como exemplo as portas lógicas “AND” e “OR¨, segundo [7]. Estas redes não conseguem resolver a porta lógica “XOR¨.

O modelo Adaline foi criado na mesma época que o Perceptron de Camada Única. Como o seu antecessor, Adaline é uma função de ativação linear e a função de erro é quadrática, pelo método de gradiente descedente.

As redes do modelo Adaline, possuem as limitações das Perceptron de Camada Única, além do seu treinamento ser mais demorado, entretanto o treinamento destas é mais suave.

Outro modelo, denominado Madaline, teve como diferencial, elementos adaptativos organizados de especificamente nas camadas treináveis. O Madaline é um modelo de diversos modelos Adaline, implementando critérios de decisão como E, OU e MAORIA. Segundo [6], no modelo Madaline consegue resolver problemas que necessitem dos recursos da porta lógica “XOR”.

Já o Perceptron de Múltiplas Camadas (MLP – *Multi-Layer Perceptron*) consta em [7] como uma evolução do Madaline. Este modelo já aumentou significativamente a complexidade de construção, pois possui diversas unidades computacionais, normalmente interconectadas e com a forma de processamento *feedforward* (todas as conexões de uma camada para outra têm apenas um sentido, o da camada de entrada para o de saída). E para completar a maior complexidade, cada neurônio de uma camada, se conecta com todos da camada seguinte. Frequentemente a função de ativação utilizada nas implementações do MLP é a sigmóide.

No modelo MLP, Szegedy et al [9] explica que a qualidade de generalização, e consequentemente no desempelho da rede, depende do número de neurônios. Apesar de não haver um número exato, ou determinado, Szegedy et al [9] cta Hecht-Nielsen (1988), sendo este que sugere a fórmula (2n+1) para calcular a quantidade de neurônios na camada intermediária, sendo n a quantidade de neurônos da camada de entrada. Entretanto, o mais aconselhável é começar o MLP com uma quantidade mínima de neurônios e, empiricamente, acrescentando neurônios gradualmente até que o objetivo seja atendido.

O MLP, algoritmo de aprendizado supervisionado mais popular é o *back-propagation*, com propagação de erro. Este ajusta os pesos da rede, por meio de correção de erros nos pares de entrada e saída. Como no modelo Adaline, o treinamento utilizado no MLP utilizado uma generalização delta, que usa as fases *forward* e *backward*.

Como diz [5], as redes neurais convolucionais (CNN) tratam-se de uma rede neural, que processa dados em uma topologia no formato de grid, com base em um dos princípios neurocientífcos, que influenciam o aprendizado profundo. As transformações lineares efetuadas, ao invés de atuar sobre toda a entrada, ocorrem dentro da grade de entrada.

Géron [7], em uma nota de rodapé, explica o conceito matemático de convolução: uma convolução é uma operação matemática que desliza uma função sobre outra e mede a integral da sua multiplicação pontual. O mesmo autor relaciona este procedimento com a transformada de Fourier e a transformada de Laplace.

O kernel, ou matriz de peso convolucional, perpassa toda a entrada, emitindo uma saída do tipo grade, executando desta forma, a função de ativação elemento a elemento. A figura 6 esquematiza este funcionamento:

Segundo Salvi et al [5], no aprendizado profundo, a ideia é usar o *Backpropagation*, para aprender automaticamente os valores de cada *kernel*, em vez de se usar um kernel projetado manualmente. As CNN’s extraem características de dados brutos, de forma automática, em vez de métodos clássicos onde os algoritmos treinam para realizar alguma tarefa com base em recursos artesanais. A etapa de pré-processamento também é mínima porque a ideia é deixar o modelo aprender quais tipos de variabilidade ele deve aprender para se tornar invariante. O kernel trabalha como *template-matching*, ou seja, após cada operação convolucional, cada célula de saída será a entrada de uma função de entrada. Se a ativação for alta, significa que o neurônio encontrou as características do kernel dentro da respectiva região de entrada. Fazendo o empilhamento de diversos *kernel* para a construção pelo uma camada, temos uma CNN.

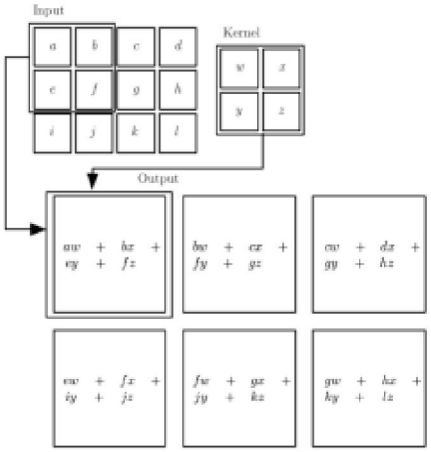


Figura 6 – Analogia de kernel, [5]

As CNNs aprendem de forma supervisionada como aproximar funções não lineares com as seguintes características, segundo [5]:

- Extração de características: as entradas de um neurônio vêm das saídas da camada de neurônios anteriores, forçando este último a extrair características locais. Por exemplo, a extração da localização exata de uma feição é menos importante desde que preserve a posição relativa com as outras feições;
- Mapeamento de recursos: cada mapa de recursos vem de neurônios restritos a compartilhar o mesmo conjunto de pesos simpáticos. Isso provoca a redução do número de parâmetros livres em comparação a um MLP e permite a criação de mapas de recursos que são invariantes ao deslocamento;
- Sub-Amostragem: após cada camada convolucional, pode-se aplicar uma camada de agrupamento, que diminui o mapa de características, reduzindo a sensibilidade da saída do mapa de características a deslocamentos e outras formas de distorção.

Convoluções podem reduzir a resolução de entrada quando não preenchidos com zeros e o *kernel* é maior que 1 × 1. Uma camada de *pooling* é outro mecanismo para redução de resolução: ela compacta um subconjunto de valores de recursos, com base, por exemplo, na média ou no valor máximo em um determinado subconjunto de recursos. Essa abordagem é comumente usada em muitas arquiteturas CNN. Como a camada de pooling não possui parâmetros, isso se torna uma vantagem sobre a convolução quando o objetivo é a redução da resolução. Um exemplo das camadas Average Pooling e Max Pooling é apresentado na Figura 7.

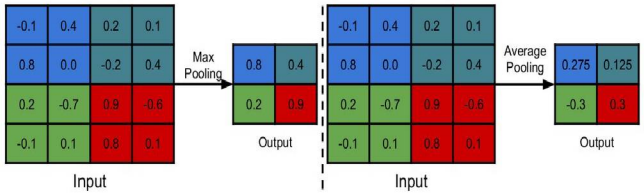


Figura 7 - Exemplo de Average e Max Pooling, de kernel 2x2, [5]

A detecção de objetos é uma tarefa que combina dois problemas: regressão, para criar uma caixa delimitadora limitando cada objeto dentro da cena, e classificação, para classificar cada caixa delimitadora em uma das classes conhecidas. A detecção de veículos em um vídeo de rodovia é um exemplo da tarefa de detecção de objetos.

A detecção de objetos é uma tarefa mais difícil do que a classificação. Enquanto a classificação é limitada em rotular a imagem ao objeto principal, que varia de tarefa para tarefa, na detecção de objetos há dois objetos de classes diferentes, que podem estar localizados em qualquer lugar da imagem, inclusive com regiões sobrepostas.

3) Tecnologia Utilizada

Neste trabalho foram desenvolvidos três softwares, todos baseados em Redes Neurais Convolucionais, sendo estas a MobileNetV1 [8], MobileNetV2 [11] e MobileNetV3 [12]. Neste tópico é apresentado um descritivo de cada uma das tecnologias.

A tecnologia MobileNet possui, no momento da escrita deste artigo, três versões: MobileNetV1, MobileNetV2 e MobileNetV3.

Howard et al. [8], o qual fala sobre a MobileNetV1, afirma que a evolução das redes neurais convolucionais estão se tornando mais complicadas de se compreender, para conseguir uma precisão maior. Além disso, estes avanços não tem melhorado a precisão destas redes, tampouco em relação ao seu tamanho e a sua velocidade. Citando alguns casos, como robótica, carro autônomo e realidade aumentada, torna-se decisões baseadas nas tarefas de reconhecimento, principalmente em plataformas que sejam mais limitadas. A proposta da MobileNetV1 é ter uma arquitetura eficiente com um conjunto de dois hiperparâmetros para construir modelos muito pequenos e de baixa latência que podem ser facilmente compatíveis com os requisitos de design para aplicativos de visão móvel e incorporados.

Seguindo o artigo de Howard et al. [8], há o detalhamento do funcionamento da MobileNet V1. Neste artigo, os autores comentam sobre a convolução separável em profundidade, a estrutura de rede e treinamento, o multiplicador de largura e o multiplicador de resolução.

Howard et al. [8] afirmam que a primeira versão do MobileNet é baseada em convoluções separáveis em profundidade, que é uma forma de convoluções fatoradas. Estas convoluções separam uma convolução padrão em uma convolução 1 x 1, chamada de convolução pontual. Desta maneira, a convolução em profundidade aplica um único filtro para cada canal de entrada. A convolução pontual então aplica uma convolução 1 x 1 para combinar as saídas da convolução em profundidade. Uma convolução padrão, filtra e combina as entradas em um novo conjunto de saídas em uma única etapa. A convolução separável em profundidade divide isso em duas camadas, uma separada para filtragem e uma camada separada para combinação. Essa fatoração tem o efeito de reduzir drasticamente a computação e o tamanho do modelo. A operação de convolução padrão tem o efeito de filtrar recursos com base nos kernels convolucionais e combinar recursos para produzir uma nova representação.

As etapas de filtragem e combinação podem ser divididas em duas etapas através do uso de convoluções fatoradas, chamadas convoluções separáveis em profundidade para redução substancial no custo computacional. As convoluções separáveis em profundidade são compostas de duas camadas: convoluções em profundidade e convoluções em pontos. Na construção desta tecnologia usou-se convoluções de profundidade para aplicar um único filtro por cada canal de entrada (profundidade de entrada). A convolução pontual, uma simples convolução 1 x 1, é então usada para criar uma combinação linear da saída da camada de profundidade. MobileNets usam não linearidades batchnorm e ReLU para ambas as camadas. A convolução em profundidade é extremamente eficiente em relação à convolução padrão. No entanto, ele apenas filtra os canais de entrada, não os combina para criar novos recursos. Portanto, uma camada adicional que calcula uma combinação linear da saída da convolução em profundidade por meio da convolução 1 x 1 é necessária para gerar esses novos recursos. A combinação de convolução em profundidade e convolução 1 x 1 (ponto a ponto) é chamada de convolução separável em profundidade. As convoluções separáveis em profundidade 3 x 3, usadas na MobileNet, usam de 8 a 9 vezes menos computação do que as convoluções padrão com apenas uma pequena redução da precisão.

Howard et al. [8], explicam que a estrutura de rede e treinamento é construída em convoluções separáveis em profundidade, exceto pela primeira camada, que é uma convolução completa. A arquitetura MobileNet é definida na Tabela 1, onde todas as camadas são seguidas por uma batchnorm e não linearidade ReLU, com exceção da camada final totalmente conectada que não possui não linearidade e alimenta uma camada softmax para classificação. Contando as convoluções de profundidade e de ponto (como camadas separadas), há um total de 28 camadas nesta tecnologia.

Type / Stride	Filter Shape	Input Size
Conv / s2	3 × 3 × 3 × 32	224 × 224 × 3
Conv dw / s1	3 × 3 × 32 dw	112 × 112 × 32
Conv / s1	1 × 1 × 32 × 64	112 × 112 × 32
Conv dw / s2	3 × 3 × 64 dw	112 × 112 × 64
Conv / s1	1 × 1 × 64 × 128	56 × 56 × 64
Conv dw / s1	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 128	56 × 56 × 128
Conv dw / s2	3 × 3 × 128 dw	56 × 56 × 128
Conv / s1	1 × 1 × 128 × 256	28 × 28 × 128
Conv dw / s1	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 256	28 × 28 × 256
Conv dw / s2	3 × 3 × 256 dw	28 × 28 × 256
Conv / s1	1 × 1 × 256 × 512	14 × 14 × 256
5× Conv dw / s1	3 × 3 × 512 dw	14 × 14 × 512
Conv / s1	1 × 1 × 512 × 512	14 × 14 × 512
Conv dw / s2	3 × 3 × 512 dw	14 × 14 × 512
Conv / s1	1 × 1 × 512 × 1024	7 × 7 × 512
Conv dw / s2	3 × 3 × 1024 dw	7 × 7 × 1024
Conv / s1	1 × 1 × 1024 × 1024	7 × 7 × 1024
Avg Pool / s1	Pool 7 × 7	7 × 7 × 1024
FC / s1	1024 × 1000	1 × 1 × 1024
Softmax / s1	Classifier	1 × 1 × 1000

Tabela 1 – Arquitetura MobileNet V1, Fonte: [8]

Howard et al. [8], explica que o multiplicador de largura é um parâmetro, chamado α. O papel do multiplicador de largura é afinar uma rede uniformemente em cada camada. Para uma dada camada e multiplicador de largura α, o número de canais de entrada M torna-se αM e o número de canais de saída N torna-se αN. O multiplicador de largura tem o efeito de reduzir o custo computacional e o número de parâmetros quadraticamente por aproximadamente α^2. O multiplicador de largura pode ser aplicado a qualquer estrutura de modelo para definir um novo modelo menor com precisão razoável, latência e compensação de tamanho. Ele é usado para definir uma nova estrutura reduzida que precisa ser treinada do zero.

Ainda em Howard et al. [8], há o multiplicador de resolução. Este é o segundo hiperparâmetro para redução de custo computacional de uma rede neural, nomeado de ρ. Aplicamos isso à imagem de entrada e a representação interna de cada camada é posteriormente reduzida pelo mesmo multiplicador. Na prática, definimos ρ implicitamente definindo a resolução de entrada.

Sandler et al. [11], o qual fala sobre a MobileNetV2, afirma que esta versão melhora o desempenho da geração anterior, o MobileNetV1, na geração de modelos móveis em várias tarefas e benchmarks. A diferença desta tecnologia, para a sua antecessora, está na estrutura residual invertida onde as conexões de atalho estão entre as finas camadas de gargalo. Na introdução do artigo, é afirmado que as redes neurais revolucionaram muitas das áreas da inteligência de máquina, permitindo uma precisão sobre-humana para tarefas desafiadoras de reconhecimento de imagem. Entretanto, o esforço para a melhoria da precisão normalmente vem acompanhada de um custo: a exigência de altos recursos computacionais.

Sandler et al. [11] detalha que a MobileNetV2 é baseada na MobileNetV1, mantendo a sua simplicidade e não requer nenhum operador especial, melhorando significativamente a sua precisão, alcançando o estado da arte em várias tarefas de classificação e detecção de imagens para aplicativos móveis.

Na segunda versão do MobileNet, a ideia é a substituição do operador convolucional completo por uma versão fatorada que divide a convolução em duas camadas separadas. A primeira camada é chamada de convolução em profundidade, que realiza uma filtragem leve aplicando um único filtro convolucional por canal de entrada. A segunda camada é uma convolução de 1 x 1, chamada de convolução pontual, que é responsável pela construção de novos recursos por meio de cálculo de combinações lineares dos canas de entrada.

Sandler et al. [11], explicando sobre os gargalos preliminares, considera uma rede neural profunda de n camadas L_i, sendo cada uma com um tensor de ativação de dimensões $h_i \times w_i \times d_i$. Seguindo este raciocínio, as propriedades básicas desses tensores de ativação, serão tratados como contêineres de h_i x w_i com d_j dimensões. Informalmente, para um conjunto de entrada de imagens reais, diz-se que o conjunto de ativações de camada (para qualquer camada L_i) forma uma "variedade de interesse". Há muito se assume que as variedades de interesse em redes neurais podem ser incorporadas em subespaços de baixa dimensão. Em outras palavras, quando olhamos para todos os pixels individuais do canal "d" de uma camada convolucional profunda, a informação codificada nesses valores na verdade está em algum coletor, que por sua vez é incorporado em um subespaço de baixa dimensão.

Continuando o raciocínio, Sandler et al. [11] afirma que à primeira vista, tal fato poderia ser capturado e explorado simplesmente reduzindo a dimensionalidade de uma camada, reduzindo assim a dimensionalidade do espaço operacional. Isso foi explorado com sucesso pelo MobileNetV1 [8] para efetivamente equilibrar computação e precisão por meio de um parâmetro multiplicador de largura. Seguindo essa intuição, a abordagem do multiplicador de largura permite reduzir a dimensionalidade do espaço de ativação até que a variedade de interesse abranja todo esse espaço. No entanto, essa intuição falha quando lembramos que as redes neurais convolucionais profundas na verdade têm transformações não lineares por coordenadas, como ReLU. Por exemplo, ReLU aplicado a uma linha no espaço 1D produz um 'raio', onde, como no espaço R^n, geralmente resulta em uma curva linear por partes com n-junções. Se o resultado de uma transformação de camada ReLU(Bx) tem um volume S diferente de zero, os pontos mapeados para o interior S são obtidos através de uma transformação linear B da entrada, indicando assim que a parte de o espaço de entrada correspondente à saída dimensional completa, é limitado a uma transformação linear. Em outras palavras, as redes profundas só têm o poder de um classificador linear na parte de volume diferente de zero do domínio de saída.

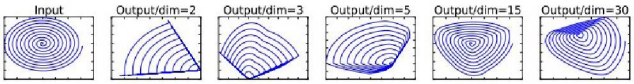


Figura 8 - Exemplos de transformações ReLU de variedades de baixa dimensão incorporadas em espaços de alta dimensão, [11].

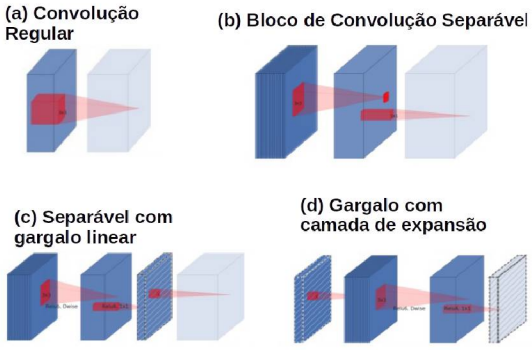


Figura 9 - Evolução de blocos de convolução separáveis, [11].

Por outro lado, comenta Sandler et al. [11], quando o ReLU colapsa o canal, ele inevitavelmente perde informações nesse canal. No entanto, se tivermos muitos canais e houver uma estrutura no coletor de ativação, essa informação ainda poderá ser preservada nos outros canais. Caso a variedade de entrada possa ser incorporada em um subespaço de dimensão significativamente menor do espaço de ativação, a transformação ReLU preserva a informação enquanto introduz a complexidade necessária no conjunto de funções expressáveis.

Resumidamente, Sandler et al. [11] destacam duas propriedades que são indicativas do requisito de que a variedade de interesse deve estar em um subespaço de baixa dimensão do espaço de ativação de dimensão superior:

1. Se a variedade de interesse permanecer com volume diferente de zero após a transformação ReLU, corresponde a uma transformação linear.

2. ReLU é capaz de preservar informações completas sobre a variedade de entrada, mas somente se a variedade de entrada estiver em um subespaço de baixa dimensão do espaço de entrada.

Esses dois insights forneceram a Sandler et al. [11] uma dica empírica para otimizar as arquiteturas neurais existentes: supondo que o coletor de interesse seja de baixa dimensão, podemos capturar isso inserindo camadas de gargalo lineares nos blocos convolucionais. Evidências experimentais sugerem que o uso de camadas lineares é crucial, pois evita que as não linearidades destruam muita informação.

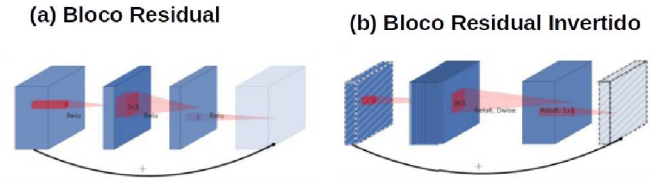


Figura 10 - A diferença entre bloco residual e resíduo invertido, [11].

Sandler et al. [11] mostram a Figura 10, sendo esta uma visualização esquemática da diferença nos projetos. A motivação para inserir atalhos é semelhante à das conexões residuais clássicas: queremos melhorar a capacidade de um gradiente se propagar pelas camadas multiplicadoras. No entanto, o design invertido é consideravelmente mais eficiente em termos de memória (consulte a Seção 5 para obter detalhes), além de funcionar um pouco melhor em nossos experimentos.

Em termos de arquitetura do modelo, Sandler et al. [11] comentam que o bloco de construção básico é uma convolução separável em profundidade de gargalo com resíduos. A estrutura detalhada deste bloco é mostrada na Tabela 2. A arquitetura do MobileNetV2 contém a camada inicial de convolução total com 32 filtros, seguida por 19 camadas de gargalo residuais descritas na Tabela 3. Sandler et al. [11] usaram ReLU6 como a não linearidade devido à sua robustez quando usado com computação de baixa precisão. Sempre usamos o tamanho do kernel 3 × 3 como padrão para redes modernas e utilizamos a normalização de dropout e batch durante o treinamento.

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Tabela 2 - Bloco residual de gargalo transformando de k para k'canais, com passo s e fator de expansão t, [11].

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Tabela 3 - MobileNetV2 : Cada linha descreve uma sequência de 1 ou mais camadas idênticas (módulo passo), repetidas n vezes, [11].

Na Tabela 3, cada linha descreve uma sequência de 1 ou mais camadas idênticas (modulo stride), repetidas n vezes. Todas as camadas na mesma sequência têm o mesmo número c de canais de saída. A primeira camada de cada sequência tem um stride s e todas as outras usam stride 1. Todas as convoluções espaciais usam kernels 3 × 3. O fator de expansão t é sempre aplicado ao tamanho da entrada conforme descrito na Tabela 2.

A. Howard et al. [12] apresentam a tecnologa MobileNetV3. Esta tecnologia foi planejada para ser executada em CPUs de telefones celulares, onde o hardware utilizado é reconhecido através de uma adaptação do algoritmo NetAdapt. Esta adaptação é nomeada como rede com reconhecmento de harware (NAS). A MobileNetV3 explora como algoritmos de busca automatizada e design de rede podem trabalhar juntos para aproveitar abordagens complementares, melhorando o estado da arte geral. Através destas premissas, foram criados dois novos modelos MobileNet para lançamento: MobileNetV3-Large e MobileNetV3-Small, direcionados para casos de uso de recursos altos e baixos. Esses modelos são então adaptados e aplicados às tarefas de detecção de objetos e segmentação semântica. Para a tarefa de segmentação semântica (ou qualquer previsão de pixel denso), propôs-se um novo decodificador de segmentação eficiente Lite Reduced Atrous Spatial Pyramid Pooling (LR-ASPP). Alcançou-se novos resultados de última geração para classificação, detecção e segmentação de dispositivos móveis.

No MobileNetV3, Sandler et al. [11] usam uma combinação dessas camadas como blocos de construção para construir os modelos mais eficazes. As camadas também são atualizadas com não linearidades swish modificadas. Tanto o squeeze quanto a excitação, bem como a não linearidade swish, usam o sigmóide, que pode ser ineficiente para calcular, além de difícil manter a precisão na aritmética de ponto fixo, então substituímos isso pelo sigmóide rígido. Em [11], os autores comentam o uso da NAS com reconhecimento de plataforma para pesquisar as estruturas de rede global otimizando cada bloco de rede. Em seguida, usa-se o algoritmo NetAdapt para pesquisar o número de filtros por camada. Essas técnicas são complementares e podem ser combinadas para encontrar efetivamente modelos otimizados para uma determinada plataforma de hardware. A arquitetura MnasNet-A1 foi utilizada como modelo Large mobile inicial e, em seguida, aplicaram o NetAdapt, com uma latência alvo em torno de 80ms. No entanto, foi observado que o design de recompensa original não é otimizado para pequenos modelos móveis. Para o modelo final MobileNetV3-Small, foi utilizado um fator de peso menor, sendo w = -0,15 (vs o original w = −0,07).

Sandler et al. [11], comentam que a segunda técnica empregada na busca de arquitetura é o NetAdapt. Essa abordagem é complementar ao NAS com reconhecimento de plataforma: permite o ajuste fino de camadas individuais de maneira sequencial, em vez de tentar inferir uma arquitetura grosseira, mas global. Em suma, a técnica procede da seguinte forma:

1. Começa com uma arquitetura de rede seed encontrada pelo NAS com reconhecimento de plataforma.

2. Para cada etapa:

• Gerar um conjunto de novas propostas. Cada proposta representa uma modificação de uma arquitetura que gera pelo menos δ redução na latência em relação ao passo anterior.

• Para cada proposta, usa-se o modelo pré-treinado da etapa anterior e preenchemos a nova arquitetura proposta, truncando e inicializando aleatoriamente os pesos ausentes conforme apropriado. Ajuste cada proposta de passos T para obter uma estimativa grosseira da precisão.

• Selecionada a melhor proposta de acordo com alguma métrica.

3. Iterar a etapa anterior até que a latência de destino seja atingida.

Comentando sobre as melhorias de rede, Sandler et al. [11], diz-se que foram redesenhadas as camadas computacionalmente caras no início e no final da rede. Também introduziram uma nova não linearidade, h-swish, uma versão modificada da recente não linearidade swish, que é mais rápida de calcular e mais amigável à quantização. A primeira modificação executada, reconstroi como as últimas camadas da rede interagem. Esta alteração aumenta a precisão do algoritmo. Os modelos atuais baseados na estrutura de gargalo invertido do MobileNetV2 e suas variantes usam a convolução 1x1 como camada final para expandir para um espaço de recursos de maior dimensão. Essa camada é extremamente importante para ter recursos avançados para previsão. No entanto, isso tem um custo de latência extra. Para reduzir a latência e preservar os recursos de alta dimensão, moveu-se essa camada além do pool médio final. Este conjunto final de recursos agora é calculado com resolução espacial de 1x1 em vez de resolução espacial de 7x7. O resultado dessa escolha de design é que o cálculo dos recursos se torna quase livre em termos de computação e latência. Outra camada cara é o conjunto inicial de filtros. Os modelos móveis atuais tendem a usar 32 filtros em uma convolução 3x3 completa para construir bancos de filtros iniciais para detecção de bordas. Muitas vezes, esses filtros são imagens espelhadas um do outro. Os autores de MobileNetV3 decidiram usar a não linearidade hard swish para essa camada, uma vez que ela teve um desempenho tão bom quanto outras não linearidades testadas. Usando este raciocínio, reduziram o número de filtros para 16, mantendo a mesma precisão de 32 filtros usando ReLU ou swish. Isso economiza mais 2 milissegundos e 10 milhões de Mads.

A tabela x apresenta a especificação para MobileNetV3-Large e a Tabela 5, a especificação para MobileNetV3-Small.

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

Tabela 4 - Especificação para MobileNetV3-Large, [11]

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d, 3x3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	-	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1024	-	HS	1
$1^2 \times 1024$	conv2d 1x1, NBN	-	k	-	-	1

Tabela 5 - Especificação para MobileNetV3-Small, [11]

C. Base de dados

A base de dados utilizada é a Birds 400 - Species Image Classification [13], obtida no site da Kaggle. Estão contempladas neste base 400 espécies de pássaros. Esta base

possui, no total, 62.388 imagens, sendo 58.388 imagens de treinamento, 2.000 de teste (5 imagens por espécie) e 2.000 de validação (5 imagens por espécie). O conjunto de treinamento é desbalanceado. O de teste e de validação são balanceados. Há apenas um pássaro por imagem, sendo que este ocupa pelo menos 50% da área enquadrada. São imagens coloridas, no formato JPG, com altura/largura de 224 pixels.

Cada conjunto de dados (treinamento, teste e validação) contém 400 subdiretórios, nomeados com o nome da espécie. Há uma planilha, no formato CSV, chamada Bird_Species.csv. Nesta planilha, há 3 colunas, sendo a primeira coluna de caminhos relativos da estrutura de pastas, a segunda de rótulos e a terceira contém o nome científico.

Na descrição desta base, estas imagens foram publicadas na internet, buscadas pelo nome da espécie. Não há imagens duplicadas, pois as imagens foram submetidas a um software (desenvolvido em Python) que fez a identificação e remoção das imagens duplicadas e com mais de um indivíduo. Depois foi feito um tratamento nas imagens, para garantir que o indivíduo fotografado ocupasse (no mínimo) 50% do frame. Das imagens selecionadas e tratadas, foram numeradas em ordem crescente, com zeros à esquerda.

Um desequilíbrio significativo no conjunto de dados é a proporção de imagens de espécies masculinas para imagens de espécies femininas. Cerca de 85% das imagens são do sexo masculino e 15% do feminino. Os machos típicos são muito mais coloridos, consequentemente, as imagens masculinas e femininas podem parecer totalmente diferentes. Quase todas as imagens de teste e validação são tiradas do macho da espécie.

D. MÉTODOS

Para a realização deste experimento, foi necessário o desenvolvimento de três softwares, cada um baseado em uma das arquiteturas MobileNet. As três aplicações foram desenvolvidas em Python, sendo todas executadas localmente no equipamento descrito. As métricas utilizadas em todos os casos serão os gráficos de Trainning Loss, Validation Loss, Trainning Accuracy, Validation Accuracy. Precision, Recall, F1-Score, Support e a medida de tempo de execução, sendo feita em segundos. A base de dados usada neste estudo é sempre a mesma, sendo a descrita no item C.

A metodologia aplicada segue o raciocínio descrito na figura 11.

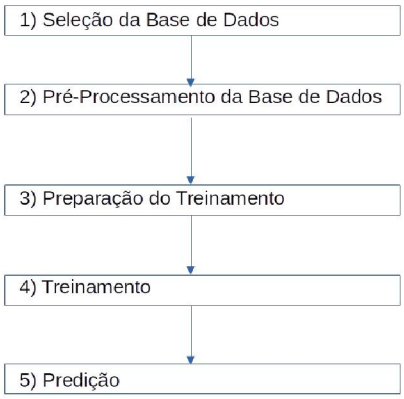


Figura 11 – Sequência aplicada nos métodos

As informações do primeiro passo, 1) Seleção da Base de Dados, foi eleita, por ser uma base de imagens, com uma quantidade considerável de imagens.

O segundo item, 2) Pré-processamento da base de dados, já foi feito pelo autor da base, conforme descrito anteriormente neste trabalho. Neste ponto, as imagens dos pássaros foram separadas em pastas chamadas Train, Test e Valid. Cada pasta contém sub-pastas nomeadas com a espécie da ave. Neste estudo, foram utilizadas as pastas Train e Test. A pasta Valid possui apenas 5 imagens por pasta, não sendo considerado como fator para ter um *underfitting* no treinamento.

O terceiro item, 3) Preparação para o treinamento, teve algumas tarefas, sendo idênticas para as três técnicas, conforme explicado a seguir.

Na primeira parte no algoritmo desenvolvido, foram sorteadas 16 imagens (randomizadas através da função random do numpy), da base de treinamento. Nesta escolha é mostrada a imagem e o nome da sua espécie. Foi escolhida esta pasta, pois para cada espécie temos uma grande gama de imagens, tendo uma probabilidade menor para repetir uma imagem.

O passo seguinte foi gerar um *dataframe*, em que uma das colunas é o caminho completo de cada imagem, a segunda coluna é o nome da espécie da referida imagem. Neste passo há dois dataframes. Um chamado de train_de e outro de test_df. Ambos foram retirados da pasta train. O dataframe train_df corresponde a 80% (oitenta por cento) das imagens, enquanto o test_df corresponde a 20% (vinte por cento).

Como na formação do *dataframe* foi usado o caminho completo da imagem, há a necessidade de se anexar o conteúdo da mesma para o treinamento. Para tanto foi utilizado o *ImageDataGenerator*, do pacote Tensorflow Keras. Na bibliografia referente, esta função gera os lotes de dados de imagem do tensor, com aumento de dados em tempo real. Nesta função foi utilizada a função de pré processamento (*preprocessing_function*), sendo as opções os seguintes itens:

- tf.keras.applications.mobilenet.preprocess_input (versão MobilenetV1);
- tf.keras.applications.mobilenet_v2.preprocess_input (versão MobilenetV2) e
- tf.keras.applications.mobilenet_v3.preprocess_input (versão MobilenetV3);

Após este procedimento, foi utilizado o *ImageDataGenerator*, do pacote TensorFlow Keras. Na bilbliografia referente, esta função gera os lotes de dados de imagem do tensor. Nesta função foram utilizadas as seguintes configurações:

- *dataframe*: podendo ser o train_df e test_df;
- x_col: este campo é uma string que se refere ao nome da coluna do *dataframe* com a referência do

caminho do arquivo. Em todas as opções, os *dataframes* chamaram esta opção de *Filepath*;

- y_col: este campo é uma *string* que se refere ao nome ou *label* da imagem;
- *target_size*: é uma *tupla de inteiros* contendo as dimensões para as quais todas as imagens serão redimensionadas. No caso foi configurado com a string (224, 224);
- *color_mode:* esta configuração é a opção de colorização que será usada, podendo ser uma das seguintes opções: "grayscale", "rgb", "rgba". A escolhida foi a opção padrão “rgb”;
- class_mode: é o modo para gerar os alvos, tendo como opções os seguintes itens: binary, categórico, *input*, *multi_output*, *raw* e *sparse*. A opção seleconada foi a categórico/
- *batch_size*: Tamanho do lote de dados. A opção selecionada foi de 32;
- *shuffle*: Opção de embaralhamento de dados. O item selecionado foi *true*;
- seed: semente aleatória opcional para embaralhamento e transformações. A opção selecionada foi 42;
- *subset*: Subconjunto de dados ("treinamento" ou "validação") se *validation_split* estiver definido em *ImageDataGenerator*.

Na sequência, foi feito um redimensionamento das imagens, utilizando o Sequential, do pacote TensorFlow Keras. Neste item tem duas opções que foi utilizado, sendo:

- layers.experimental.preprocessing.Resizing: com uma tupla (224,224). Apesar de se já ter sido utilizada no item anterior, na *tag target_size*, optou-se por re-fazer para manter o tamanho escolhido das imagens;
- layers.experimental.preprocessing.Rescaling: (1./255). Aqui se define a escala das imagens. No caso 1/255.

Neste ponto, os dados estão prontos para o treinamento do modelo. Foram utilizadas três tecnologias, todas relacionadas aos pacotes que estão sendo testados. São elas:

- *tf.keras.applications.MobileNet* (versão MobilenetV1);
- *tf.keras.applications.MobileNetV2* (versão MobilenetV2) e
- *tf.keras.applications.MobileNetV3Small* (versão MobilenetV3);

Cada um destes modelos de treinamento, suporta algumas configurações. As utilizadas foram:

- *input_shape*: tupla contendo ao tamanho das imagens e a quantidade de canais. Esta tupla é opcional. No caso foram utilzada a seguinte tupla: (224, 224, 3);
- *include_top*: Esta opção define se uma camada é totalmente conectada na camada superior. Neste estudo optou-se por *False*;
- *weights*: é uma string, onde é definido um conjunto de pesos. Há a opção *None (inicialização aleatória)*, caminho para um arquivo de pesos a ser carregado ou imagenet (pré-treinamento no ImageNet. A opção *imagenet* foi a escolhida para este trabalho;
- *pooling*: extração de recursos, para quando *include_top* é marcado *false*. A *string* pode ser em branco, quando a saída do modelo será a saída do tensor 4D do último bloco convolucional. *Max* para quando o *pool* máximo global é aplicado. *Avg* significa que o pool de média global será aplicado à saída do último bloco convolucional e, portanto, a saída do modelo será um tensor 2D. A opção AVG foi usada para este estudo.

O treinamento foi feito através da implementação Dense, do Tensorflow Keras Layers. As configurações usadas na primeira camada foram:

- *Units*: Este opção define a dimensionalidade do espaço de saída. Foi utilizado 256.
- Activation: Função de ativação para ser usada. A ativação *relu* foi a opção desta aplicação.

A segunda camada utilizada foi feita através do Dropout, do Tensorflow Keras Layers. A configuração usada na primeira camada foi:

- *rate*: É um inteiro que deve estar entre 0 e 1. Segundo a explicação da documentação, esta é a fração das unidades de entrada a serem descartadas. O valor estipulado para este trabalho foi 0.2.

As duas camadas mencionadas (*Dense* e *Dropout*) foram utilizadas duas vezes seguidas, sendo as configurações mencionadas nas duas chamadas.

Já a geração de camadas de *outputs*, foi utilizada a camada Dense, ativando duas configurações diferentes das gastas nas camadas anteriores. São as novas configurações:

- Units: 400;
- Activation: Softmax;

Neste momento, agrupamos os inputs e outputs através do Método Model, do Tensorflow Keras. Na sequência, é necessário compilar o modelo, através do método Tensorflow Keras compile. As configurações ajustadas neste método, foram as seguintes:

- Optimizer: Usa uma instância de otimização. A escolhida foi a Adam, com o *learning_rate* 0.0001;
- *Loss*: Conhecida como função de perda. A função de perda escolhida foi o padrão *categorical_crossentropy*.
- *Metrics*: Lista de métricas a serem avaliadas pelo modelo durante o treinamento e teste. A métrica escolhida foi *accuracy*.

Neste momento, após configurar o modelo com perdas e métricas com o model.compile(), treinou-se o modelo com o model.fit(). Esta função possibilita as seguinte configurações:

- *Input data*: aceita dois tipos de entrada, podendo ser um tensor do Tensorflow ou uma matriz *numpy*. No caso deste estudo é uma matriz *numpy*;

- *steps_per_epoch*: Número total de etapas (lotes de amostras) antes de declarar uma época concluída e iniciar a próxima época. Neste estudo foi automatizado a contagem com o tamanho da base de treinamento, totalizando 1168.
- *validation_data*: Dados sobre os quais avaliar a perda e quaisquer métricas do modelo no final de cada época. O modelo não será treinado com esses dados. Assim, o fato de que a perda de validação de dados fornecida usando *validação_split* não é afetada por camadas de regularização como ruído e abandono. No sistema desenvolvido, foi usada o *dataframe* da pasta data, diponibilizado na base.
- *validation_steps*: Relevante apenas se *validation_data* for fornecido. Número total de etapas (lotes de amostras) a serem extraídas antes de parar ao realizar a validação no final de cada época. Neste estudo foi com o tamanho da base de testes, totalizando 292.
- *epochs*: Número de épocas para treinar o modelo. Neste estudo foi limitado a 10 épocas.
- *Callbacks*: Lista de retornos de chamada a serem aplicados durante o treinamento. Para este algoritmo, foram dois monitores. O primeiro acompanhamento é relacionado ao val_loss. Caso o valor de perda comece a subir, o treinamento é suspenso. A outra supervisão é a criação de checkpoints, para posterior entendimento de como se comportou o treinamento.

Após todos estes passos, o sistema faz as predições, utilizado o modelo recém criado. O algoritmo registra todos estes passos em um arquivo de log, para cada modelo testado.

E. TECNOLOGIAS

O hardware utilizado neste estudo foi um notebook Asus ROG GL502VS, com (Intel Core i7-7700HQ), com 16GB de memória RAM, SSD 1TB, com sistema operacional Ubuntu 22.04. As versões dos softwares/ferramentas: Python 3.10.4, Visual Studio Code 1.70.1, TensorFlow 2.0.

II. RESULTADOS E DISCUSSÕES

Nesta seção, serão apresentados os resultados obtidos nos softwares desenvolvidos. Estes resultados serão divididos em três tópicos, relacionados ao nome da tecnologia relacionada.

As métricas utilizadas em todos os casos serão os gráficos de Trainning Loss, Validation Loss, Trainning Accuracy, Validation Accuracy. Precision, Recall, F1-Score, Support e a medida de tempo de execução, sendo feita em segundos.

Os gráficos apresentados serão os de Perda (L*oss*) e os de Precisão (*Accuracy*).

Para fazer as interpretações dos valores e gráficos que foram gerados, há a necessidade de se relatar os conceitos das métricas coletadas.

Anteriormente neste trabalho, foram abordados os conceitos de aprendizado profundo e redes neurais artificiais. Estas técnicas, quando aplicadas a um conjunto de dados específico (verificar o item A. Base de Dados), é gerado um modelo que recebe algumas entradas e emite uma saída. Com o modelo pronto, é importante avaliar o desempenho deste, aí entra uma medida popular de Perda (L*oss*). Esta medida quantifica o erro produzido pelo modelo. Os valores altos de perda expressa que muitas saídas errôneas estão sendo geradas. Enquanto um valor de perda baixo, significa que as saídas produzidas estão mais próximas do valor real, segundo [14].

Seguindo o raciocínio de [14], a Perda de Treinamento (*Trainning Loss*) é uma métrica de avaliação que diz como um modelo de aprendizado profundo se ajusta aos dados de treinamento. Resumindo, é o erro do modelo no conjunto de treinamento, sendo que este é uma parte de um conjunto de dados usado para treinar inicialmente o modelo. A perda de treinamento é calculada capturando a soma dos erros para cada exemplo no conjunto de treinamento.

Ainda em [14], a perda de validação (*Validation Loss*) é uma métrica usada para avaliar o desempenho de um modelo de aprendizado profundo no conjunto de validação. Este conjunto de validação é uma parte do conjunto de dados reservado para validar o desempenho do modelo. Tanto a perda de validação quanto a perda de treinamento são calculadas a partir da soma dos erros de cada exemplo no conjunto de validação. A perda de validação tem mais uma nuance: é medida após cada época. Isso nos informa se o modelo precisa de mais ajustes ou não. Para se conseguir chegar nesta conclusão, geralmente é traçada uma curva de aprendizado para a perda de validação.

Com as considerações acima, pode-se fazer as interpretações em cima dos gráficos de Perda. Há três tipos de interpretações possíveis:

- Subajuste: A perda de validação é mais alta que a de treinamento. O underfitting (ou subajuste) ocorre quando o modelo não consegue modelar com precisão os dados de treinamento o que ocasiona grandes erros. Para este caso há dois caminhos para reverter esta situação: treinamento adicional e/ou aumento dos dados de treinamento.
- Sobreajuste: A perda de validação é maior que a de treinamento. A interpretação desta situação é que o modelo está superajustado e não consegue generalizar quando se usa novos dados. É o caso em que o modelo funciona bem com os dados de treinamento, mas funciona mal nos novos dados como os do conjunto de validação. Neste caso nota-se que a curva de perda de validação vai caindo e, em determinado ponto, começa a aumentar novamente. Verifica-se que há duas explicações para esta ocorrência: fez-se o treinamento por um longo período ou os dados não conseguem expressar corretamente a complexidade situacional. Caso o modelo tenha sido treinado por um longo período, um monitoramento que perceba a estabilização da perda e já provoque o seu encerramento. Este

comportamento é conhecido como perda antecipada. Esta situação também é conhecida como *overfitting*.

- Bom encaixe: Ocorre quando a perda de treinamento e a perda de validação diminuem e se estabilizam em um ponto específico. Este padrão indica um ajuste ótimo.

Um outro conceito mostrado por [14], a precisão (*Accuracy*) mede quão bem o modelo gerado faz as previsões, fazendo um comparativo entre as previsões do modelo com os valores reais. Usa-se a porcentagem como referência de acertos. Um exemplo para esta situação é em um conjunto de imagens deseja-se saber se há um cão presente ou não. Tendo um conjunto com 10 imagens, sendo que o modelo previu corretamente a presença do cão em 6, o resultado gera uma precisão de 60% (6/10). Analisando este conceito, ter 100% de precisão indica excesso de aprendizado ou ajuste excessivo. A ideia é extrair um padrão dos dados de treinamento para que ele possa ter um desempenho preditivo decente em entradas não vistas. Cem por cento de precisão pode significar apenas uma memorização do conjunto de treinamento e resultar em uma má generalização. O cálculo da precisão é dado por:

$$Precison = \frac{TP}{(TP + FP)}$$

onde,

TP: *True Positive* (Verdadeiro Positivo),

FP: *False Positive* (Falso Positivo)

A definição de *Recall*, (Lembrar, lembrança) *mostrada em [14]*, equivale a taxa de verdadeiro positivo, das imagens que foram usadas para predição. A fórmula do Recall é:

$$RECALL = \frac{TP}{(TP + FN)}$$

onde,

- FN: *False Negative* (Falso Negativo)

O F1-Score, conceito contido em [14], é a média harmônica de precisão e recall. Leva em consideração tanto os falsos positivos quanto os falsos negativos. Portanto, ele funciona bem em um conjunto de dados desequilibrado. A pontuação F1 dá o mesmo peso para o RECALL e precisão. Existe uma pontuação F1 ponderada na qual podemos dar pesos diferentes para recordação e precisão. A fórmula do F1-Score é dada por:

$$F1 - Score = \frac{(2 \times Precison \times Recall)}{(Precision + Recall)}$$

Suporte (*support*) é o número de ocorrências reais da classe no conjunto de dados especificado. O suporte desequilibrado nos dados de treinamento pode indicar fraquezas estruturais nas pontuações relatadas do classificador e pode indicar a necessidade de amostragem estratificada ou rebalanceamento. O suporte não muda entre os modelos, mas diagnostica o processo de avaliação.

1) MobilenetV1:

A primeira métrica que chama a atenção nesta tecnologia é o tempo de execução: 5895 segundos (aproximadamente 98 minutos). O *Teste Loss* registrado foi de 0.70408 e o *Teste de Accuracy* foi de 82.05%.

A Figura 12, gerada pelo sistema, mostra como se comportaram os gráficos de perda:

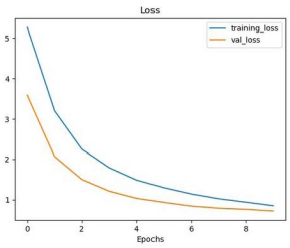


Figura 12 - Gráfico de Loss, MobilenetV1

A Figura 13, gerada pelo sistema, mostra como se comportaram os gráficos de precisão:

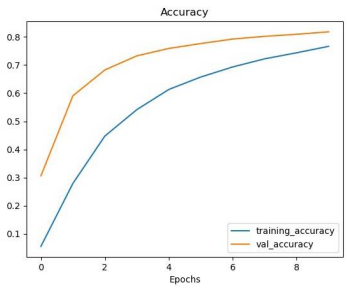


Figura 13 - Gráfico de Accuracy, MobilenetV1

Além das métricas apresentadas, outras métricas também foram geradas, são elas:

- Precision: 0,83;
- Recall: 0,82;
- F1-Score: 0,82;
- Support: 11678;

Estes resultados são a média geral das 400 classes (espécies).

2) MobilenetV2:

Nesta tecnologia, o tempo de execução foi de 6546 segundos ou seja, aproximadamente 107 minutos. O *Teste Loss* registrado foi de 0.68196 e o *Teste de Accuracy* foi de 82.42%.

A Figura 14, gerada pelo sistema, mostra como se comportaram os gráficos de perda:

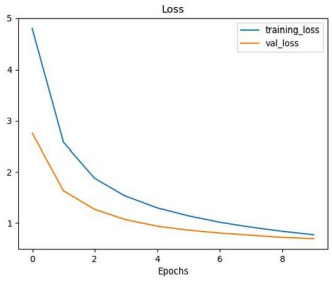


Figura 14 - Gráfico de Loss, MobilenetV2

A Figura 15, gerada pelo sistema, mostra como se comportaram os gráficos de precisão:

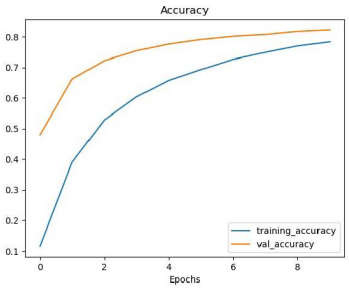


Figura 15 - Gráfico de Accuracy, MobilenetV2

Além das métricas apresentadas, outras métricas também foram geradas, são elas:

- Precision: 0,83;
- Recall: 0,83;
- F1-Score: 0,82;
- Support: 11678;

Estes resultados são a média geral das 400 classes (espécies).

3) MobilenetV3:

Na última tecnologia abordada, o tempo de execução foi de 3661 segundos, totalizando aproximadamente 61 minutos. O *Teste Loss* registrado foi de 0.681960.66440 e o *Teste de Accuracy* foi de 83.69%.

A Figura 16, gerada pelo sistema, mostra como se comportaram os gráficos de perda:

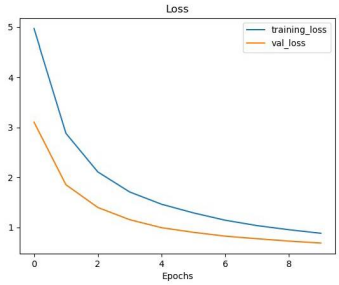


Figura 16 - Gráfico de Loss, MobilenetV2

A Figura 17, gerada pelo sistema, mostra como se comportaram os gráficos de precisão:

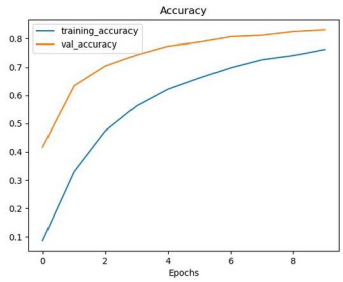


Figura 17 - Gráfico de Accuracy, MobilenetV2

Além das métricas apresentadas, outras métricas também foram geradas, são elas:

- Precision: 0,84;
- Recall: 0,83;
- F1-Score: 0,82;
- Support: 11678;

Estes resultados são a média geral das 400 classes (espécies).

Analisando-se os dados gerados e apresentados anteriormente neste tópico, não se notou diferenças significativas entre as três técnicas usando a base se dados disponibilizada. Os artigos de referência das tecnologias MobileNet V1 (referência [8]), MobileNet V2 (referência [11]) e MobileNet V3 (referência [12]), comentam que houve um aumento de desempenho geral nos sistemas, levando-se em conta todos os parâmetros de desempenho. Neste artigo notou-se que da primeira versão, MobilenetV1[8], para a segunda, MobilenetV2[11], houve uma perda de desempenho na execução do script.

Para efeito de comparação, a Tabela 1 tem a compilação de dados das três tecnologias.

	MobileNetV1	MobileNetV2	MobileNetV3
Tempo (s)	5895	6456	3361
Loss	0,70408	0,68196	0,66440
Accuracy	82,05%	82,42%	83,69%
Precision	0,83	0,83	0,84
Recall	0,82	0,83	0,84
F1-Score	0,82	0,82	0,83
Support	11678	11678	11678

Tabela 1 – Aglutinação de resultados

III. CONCLUSÕES

Analisando-se os gráficos de perda (*loss*), as três tecnologias (MobilenetV1, MobilenetV2 e MobilenetV3) obtiveram um Bom Encaixe, segundo os gráficos *Training Loss*, *Validation Loss*, *Training Accuracy* e *Validation Accuracy*.

Ao analisar os dados, tendo como base os indicadores Precision, Recall, F1-Score, Support, o Teste Loss e o Teste de Accuracy, não se notou uma variação substancial nos valores. Nos parâmetros mencionados, a maior variação registrada foram de 2 pontos percentuais.

Entretanto na variação de tempo de execução: A tecnologia MobilenetV3 demorou 56,6% do tempo do que a MobilenetV2. Esta redução de tempo de execução mostrou que a abordagem utilizada na MobilenetV3 foi acertada.

Para trabalhos futuros, acredita-se ser interessante comparar estes dados com outras tecnologias presentes no portfólio do Tensorflow.

Referências

[1] Bárbara Priscila Moreira de Mélo e G. F. de Lima, "Proposta de observação de aves como atividade estratégica à conservação ambiental no Jardim Botânico Benjamim Maranhão em João Pessoa - PB". Universidade Federal da Paraíba, 29 de abril de 2015. Acessado: 29 de agosto de 2022. [Online]. Disponível em: https://repositorio.ufpb.br/jspui/handle/tede/7933

[2] A. O. dos Santos e A. K. A. Montenegro, "BIRDWATCHING: Educação Ambiental como ferramenta de combate ao tráfico da avifauna". Anima Educação, 1o de janeiro de 2021. Acessado: 5 de agosto de 2022. [Online]. Disponível em: https://repositorio.animaeducacao.com.br/handle/ANIMA/14880

[3] A. Marini e A. L. Koerich, "Classificação automática de espécies de pássaros usando estratégias superficiais e profundas". PUC-PR, 2014. Acessado: 1o de setembro de 2022. [Online]. Disponível em: https://archivum.grupomarista.org.br/pergamumweb/vinculos/tede/andreiamarini.pdf

[4] G. F. Luger, inteligência Artificial, vol. 1. Pearson, 2014.

[5] A. de A. Salvi e R. C. Barros, "Convolutional neural networks compression for object detection", Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2021. Acessado: 24 de setembro de 2022. [Online]. Disponível em: https://hdl.handle.net/10923/19344

[6] R. R. Goldschmidt, Uma Introdução à Inteligência Computacional: fundamentos, ferramentas e aplicações, Primeira Edição., 1 vols. 2010.

[7] A. Géron, Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems, First edition. Beijing ; Boston: O'Reilly Media, 2017.

[8] A. G. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", 2017, doi: 10.48550/ARXIV.1704.04861.

[9] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, e Z. Wojna, "Rethinking the Inception Architecture for Computer Vision". arXiv, 11 de dezembro de 2015. Acessado: 26 de setembro de 2022. [Online]. Disponível em: http://arxiv.org/abs/1512.00567

[10] A. de P. Braga, A. P. de L. F. de Carvalho, e T. B. Ludermir, Redes Neurais Artificiais: Teoria e Aplicações, 1o ed. Rio de Janeiro - RJ: LTC - LIVROS TÉCNICOS E CIENTÍFICOS EDITORA S.A., 2000.

[11] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, e L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks". arXiv, 21 de março de 2019. Acessado: 26 de setembro de 2022. [Online]. Disponível em: http://arxiv.org/abs/1801.04381

[12] A. Howard et al., "Searching for MobileNetV3". arXiv, 20 de novembro de 2019. Acessado: 28 de setembro de 2022. [Online]. Disponível em: http://arxiv.org/abs/1905.02244

[13] Gerry, "BIRDS 400 SPECIES- IMAGE CLASSIFICATION". BIRDS 400 SPECIES- IMAGE CLASSIFICATION, 28 de junho de 2022. Acessado: 28 de junho de 2022. [Online]. Disponível em: https://www.kaggle.com/datasets/gpiosenka/100-bird-species/code

[14] Google IA, "Curso intensivo de machine learning com as APIs do TensorFlow", 28 de agosto de 2022. https://developers.google.com/machine-learning/crash-course