Classificação Binária de Imagens de Animais Domésticos Usando Redes Neurais Convolucionais (CNN)

Rian W. Costa[1], Lucas F. dos Santos[2], Daniel L. Marques[3], Vinicius A. Lima[4] and Davi M. Gonçalves[5]
Universidade Federal de São João del-Rei (UFSJ)
rianwagner80@aluno.ufsj.edu.br[1], lucasferst.13@aluno.ufsj.edu.br[2], daniellmarquesfacul@aluno.ufsj.edu.br[3],
viniciusalima1007@aluno.ufsj.edu.br[4], medeirosdavi07@aluno.ufsj.edu.br[5]

Abstract

O desafio da classificação de imagens consiste em analisar e categorizar objetos a partir dos pixels que compõem uma imagem. Neste estudo, apresentamos uma solução para esse problema por meio de uma Rede Neural Convolucional (CNN). A rede foi treinada com um conjunto de dados de imagens de gatos e cachorros. Depois do treinamento, a CNN visa fazer predições de novas imagens, distinguindo com precisão entre gatos e cachorros.

1 Introdução

A classificação automática de imagens é um desafio central em várias áreas da inteligência artificial, particularmente no reconhecimento de padrões e visão computacional. Um dos problemas clássicos nesse campo é a distinção entre diferentes objetos ou categorias a partir de imagens, uma tarefa que, apesar de simples para humanos, pode ser complexa para máquinas.

As Redes Neurais Convolucionais (CNNs) emergiram como uma poderosa ferramenta para lidar com a classificação de imagens, graças à sua capacidade de capturar padrões espaciais e hierárquicos. Essas redes são especialmente eficazes em tarefas onde é necessário identificar características intrínsecas dos objetos em uma imagem, como a textura e a forma.

1.1 Problema

A classificação de imagens é uma tarefa fundamental em diversas aplicações de visão computacional, desde a identificação de objetos em ambientes autônomos até a organização automática de álbuns de fotos. O problema específico abordado neste trabalho consiste na classificação binária de imagens de animais domésticos, mais especificamente, cães e gatos. Essa tarefa, embora simples à primeira vista, apresenta desafios consideráveis devido à alta variabilidade nas imagens, como diferentes raças, poses, iluminação, e cenários de fundo.

Identificar corretamente se uma imagem contém um cão ou um gato requer que o modelo aprenda a reconhecer características sutis que distinguem essas duas categorias. Erros na classificação podem ocorrer quando o modelo falha em capturar essas nuances, resultando em predições incorretas. Além disso, a necessidade de um modelo robusto e generalizável é crucial, dado que ele deve ser capaz de classificar corretamente não apenas as imagens do conjunto de treinamento, mas também novas imagens que ele nunca viu antes.

Neste contexto, o problema a ser resolvido é o desenvolvimento de uma Rede Neural Convolucional capaz de distinguir entre cães e gatos com alta precisão, superando as limitações impostas pela variabilidade intrínseca do conjunto de dados utilizado.

1.2 Aplicações de Redes Neurais Convolucionais em Diversas Áreas

A capacidade das CNNs de capturar e aprender padrões complexos a partir de dados visuais as torna especialmente adequadas para resolver problemas em campos como a medicina, segurança e veículos autônomos. A seguir, discutimos algumas das principais aplicações das CNNs em cada uma dessas áreas.

- **Medicina:** Na saúde, CNNs são eficazes em diagnósticos médicos a partir de imagens. Litjens [7] revisou o uso de deep learning em tomografias e ressonâncias magnéticas, destacando sua aplicação em diagnósticos assistidos por imagem.
- **Segurança:** As CNNs são cruciais na segurança, especialmente em vigilância. Redmon [8] introduziu o modelo YOLO, que revolucionou a detecção de objetos em tempo real, tornando-se amplamente utilizado em sistemas de monitoramento para identificar atividades suspeitas.
- **Veículos autônomos:** No campo dos veículos autônomos, as CNNs são fundamentais para a percepção visual. O KITTI dataset, de Geiger [9], tornou-se referência para testar algoritmos de visão computacional, impulsionando avanços em detecção de objetos, rastreamento e segmentação semântica.

2 Fundamentação Teórica

As redes neurais convolucionais (Convolutional Neural Networks, CNNs) representam uma classe especial de redes neurais profundas, amplamente reconhecidas por sua eficácia em tarefas de processamento de imagens. Inspiradas na arquitetura do sistema visual humano, segundo Krizhevsky, Sutskever e Hinton [1], as CNNs são projetadas para processar dados que têm uma estrutura de grade, como imagens, extraindo características hierárquicas em diferentes níveis de abstração, que vão desde bordas e texturas até objetos completos.

A estrutura básica de uma CNN é composta por três tipos principais de camadas: camadas de convolução, camadas de pooling e camadas totalmente conectadas. A camada de convolução é responsável por aplicar filtros sobre a imagem de entrada para extrair características específicas, como bordas e texturas, através de operações que capturam a correlação local entre pixels, conforme descrito por Goodfellow, Bengio e Courville [2]. Essas operações são repetidas ao longo de toda a imagem, gerando mapas de características que destacam as propriedades mais salientes em diferentes regiões da imagem.

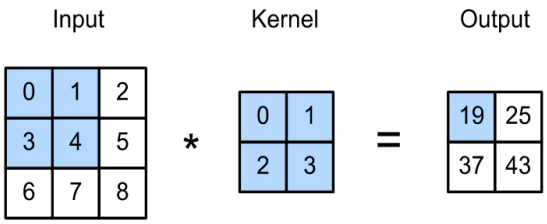
As camadas de pooling, por outro lado, reduzem a dimensionalidade dos mapas de características gerados pelas camadas de convolução, preservando as informações mais relevantes e reduzindo a sensibilidade da rede a pequenas translações da imagem, conforme discutido por Scherer, Müller e Behnke [3]. Isso é alcançado através de operações de agregação, como a max-pooling, que seleciona o valor máximo dentro de uma região específica do mapa de características.

Finalmente, as camadas totalmente conectadas (fully connected layers) combinam as características extraídas pelas camadas anteriores para produzir a classificação final ou outra tarefa de saída desejada. Essas camadas funcionam como uma rede neural tradicional, onde cada neurônio está conectado a todos os neurônios da camada anterior, segundo He et al. [4].

Durante o treinamento das CNNs, são utilizados algoritmos de otimização como o Adam [5] e funções de ativação como o ReLU (Rectified Linear Unit), que introduz não-linearidades essenciais para que a rede possa aprender padrões complexos a partir dos dados, como mostrado por Nair e Hinton [6].

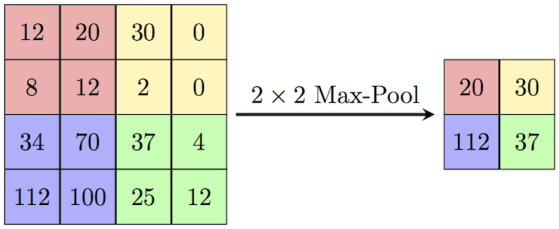
2.1 Camadas de Convolução

As camadas de convolução são o componente central das CNNs. Elas aplicam filtros (ou kernels) que percorrem a entrada (por exemplo, uma imagem) para extrair características específicas, como bordas, texturas, e padrões mais complexos. Cada filtro gera um mapa de características (feature map) que representa a presença dessas características na imagem. A convolução é definida por parâmetros como o tamanho do filtro, a profundidade do filtro e o stride (passo), que determinam como o filtro é aplicado na imagem. A capacidade das camadas de convolução de aprender e extrair características automaticamente torna as CNNs altamente eficazes em tarefas de reconhecimento de padrões.



2.2 Camadas de Pooling

As camadas de pooling seguem as camadas de convolução e têm como função principal a redução da dimensionalidade dos mapas de características, preservando as informações mais importantes. Existem diferentes tipos de pooling, sendo o max pooling e o average pooling os mais comuns. No max pooling, o valor máximo dentro de uma região do mapa de características é selecionado, enquanto no average pooling, é calculada a média dos valores. Essa redução de dimensionalidade ajuda a minimizar o overfitting e a reduzir o custo computacional, tornando o treinamento da rede mais eficiente.



2.3 Camadas Totalmente Conectadas

As camadas totalmente conectadas (fully connected layers) são tipicamente encontradas nas etapas finais de uma CNN e são responsáveis pela combinação das características extraídas nas camadas anteriores para realizar a classificação ou outra tarefa específica. Nestas camadas, cada neurônio está conectado a todos os neurônios da camada anterior, permitindo a combinação de todas as características em uma representação final. Essas camadas funcionam de forma semelhante às camadas em uma rede neural tradicional, utilizando funções de ativação como a ReLU para gerar a saída final, que pode ser uma probabilidade para cada classe no caso de classificação.

2.4 Operadores de função

- **ReLU:**

A função de ativação ReLU (Rectified Linear Unit) é amplamente utilizada em CNNs devido à sua simplicidade e eficácia. A ReLU é definida pela fórmula $f(x) = max(0, x)$, onde x é o valor de entrada para a função. Isso significa que a ReLU retorna zero para entradas negativas e o próprio valor para entradas positivas.

Uma das principais vantagens da ReLU é que ela introduz não-linearidade no modelo, permitindo que a rede neural aprenda representações complexas dos dados. Além

disso, a ReLU é computacionalmente eficiente porque envolve apenas uma operação condicional. No entanto, uma limitação da ReLU é que os neurônios podem "morrer" durante o treinamento se muitas ativações se tornarem zero, um problema conhecido como "dying ReLU". Para mitigar esse problema, variantes como a Leaky ReLU foram desenvolvidas, onde a função permite pequenas saídas negativas para valores negativos de x.

- **Função de ativação Sigmoid:**

A função de ativação Sigmoid é uma função não linear que transforma a entrada em um valor entre 0 e 1. É definida pela fórmula:

$$\sigma(\chi)\ \frac{1}{1+e^{-x}}$$

onde 'e' é a base dos logaritmos naturais. A função Sigmoid é frequentemente usada em camadas de saída para problemas de classificação binária, pois sua saída pode ser interpretada como uma probabilidade. A saída está sempre no intervalo (0, 1), o que facilita a interpretação em termos de probabilidade.

- **Função de perda Binary Crossentropy:**

A função de perda Binary Crossentropy é usada para problemas de classificação binária e mede a diferença entre as distribuições de probabilidade previstas pela rede e as distribuições reais dos dados. A fórmula é:

$$L(y, y') = -[y\ log(y')\ + (1 - y)\ log(1 - y')]$$

onde y é o rótulo verdadeiro (0 ou 1) e y' é a probabilidade prevista pela rede. É a escolha padrão para problemas de classificação binária e proporciona uma medida clara da discrepância entre a previsão e o valor verdadeiro.

- **Otimizador ADAM**

O otimizador Adam (Adaptive Moment Estimation) é um algoritmo de otimização amplamente utilizado em redes neurais devido à sua eficiência e robustez. Ele combina as vantagens dos algoritmos de gradiente descendente com momentum e o algoritmo RMSprop. A fórmula para atualizar os parâmetros é:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\Delta L(\Theta)\ v_t = \beta_2 v_{t-1} + (1 - \beta_2)$$

$$((\Delta L(\Theta))^2\ m_t' = \frac{m_t}{1-\beta_1^{\ t}}\ v_t' = \frac{v_t}{1-\beta_2^{\ t}}\ \theta = \theta - \frac{am_t}{\sqrt{v_t'+\epsilon}}$$

m_t e v_t são as estimativas dos momentos de primeiro e segundo ordem do gradiente.
β_1 e β_2 são os parâmetros de decaimento exponencial.

α é a taxa de aprendizado.

ϵ é um termo de suavização para evitar divisão por zero. Ajusta a taxa de aprendizado para cada parâmetro individualmente e combina as vantagens de gradiente descendente com momentum e RMSprop, tornando o treinamento mais eficiente.

3 Implementação

A implementação da CNN foi realizada com o objetivo de classificar a base de dados de imagens selecionada, garantindo que o modelo fosse capaz de diferenciar entre as classes presentes nos dados. Este processo envolveu várias etapas, incluindo o pré-processamento dos dados, a escolha de ferramentas apropriadas e a definição da arquitetura da rede.

3.1 Base de Dados

Para este experimento, foi utilizada a base de dados 'Dogs vs. Cats' da plataforma Kaggle. Essa base contém mais de 25 mil imagens de cães e gatos. A base de dados passou por um pré-processamento com objetivo de aumentar a acurácia no treino da rede.

3.2 Pré-processamento da Base de Dados

O pré-processamento é uma etapa crucial para garantir a qualidade dos dados de entrada e, consequentemente, o desempenho da CNN. As etapas de pré-processamento realizadas foram:

- Redimensionamento das Imagens: As imagens foram redimensionadas para 180 x 180 pixels, garantindo consistência no tamanho de entrada da rede.
- Normalização: Os valores dos pixels foram normalizados para o intervalo [0, 1], facilitando a convergência durante o treinamento.

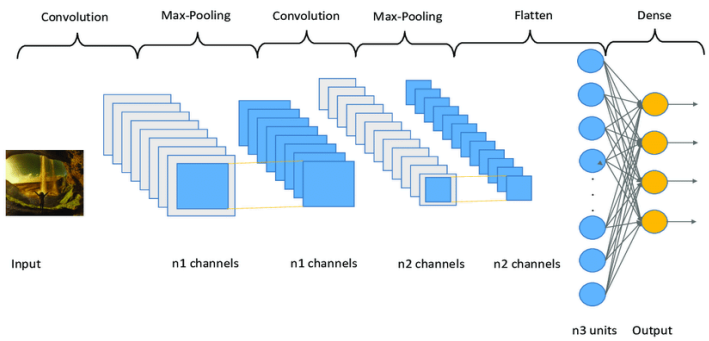
3.3 Ferramentas utilizadas

Para a implementação da CNN, foram utilizadas as seguintes ferramentas:

- **Linguagem de Programação**: Python foi a linguagem escolhida devido à sua ampla biblioteca de aprendizado de máquina e processamento de imagens.
- **Bibliotecas**: Foram utilizadas bibliotecas como TensorFlow e Keras para a construção e treinamento da rede, OpenCV para manipulação de imagens, e CUDA para processamento em GPU .
- **Ambiente de Desenvolvimento**: O desenvolvimento foi realizado em Jupyter Notebook, que oferece suporte a GPUs para acelerar o treinamento.

3.4 Implementação da CNN

A CNN foi implementada seguindo uma arquitetura típica, com 3 camadas convolucionais, 2 camadas de pooling, e 1 camada totalmente conectada. Abaixo, cada componente da rede é detalhado:



3.4.1 Datasets

Para a criação dos datasets de treinamento e validação, utilizou-se a função *image_dataset_from_directory* da biblioteca TensorFlow. Essa função carrega imagens de um diretório, organiza-as em batches e redimensiona-as automaticamente.

Parâmetros Utilizados:

- **Diretório**: Especifica o caminho onde as imagens estão armazenadas, organizadas em subpastas por classe (cachorro e gato).
- **Tamanho da Imagem (***image_size***)**: As imagens foram redimensionadas para 180x180 pixels, garantindo um tamanho uniforme para o processamento na rede.
- **Tamanho do Batch (***batch_size***)**: Definido como 32, o que otimiza o uso de memória e acelera o treinamento.

3.4.2 Criação do Modelo

Para a criação do modelo de rede neural convolucional, utilizou-se a API Sequential do Keras:

```
model = keras.Sequential()
```

Esse tipo de modelo é adequado para arquiteturas em que as camadas são empilhadas sequencialmente, ou seja, cada camada tem exatamente uma camada de entrada e uma de saída, onde a saída da última camada é a entrada da próxima.

A primeira camada adicionada ao modelo foi a Rescaling, que normaliza os valores dos pixels das imagens:

```
model.add(Rescaling(scale = 1.0/255))
```

Essa camada é essencial para garantir que os pixels, originalmente representados por valores entre 0 e 255, sejam ajustados para o intervalo [0, 1]. Essa normalização melhora o desempenho do modelo ao facilitar o processo de aprendizado, já que valores menores ajudam a estabilizar e acelerar a convergência durante o treinamento.

3.4.3 Colocando Camadas Convolucionais

As camadas convolucionais são o núcleo das Redes Neurais Convolucionais (CNNs), responsáveis por extrair características importantes das imagens, como bordas, texturas e padrões complexos. No modelo implementado, foram adicionadas três camadas convolucionais utilizando a função *Conv2D* do Keras.

- **Primeira Camada Convolucional**:

```
Conv2D(32, kernel_size = (3, 3),
       activation = 'relu')
```

Esta camada aplica 32 filtros de tamanho 3x3 às imagens. A função de ativação ReLU é utilizada para introduzir não-linearidade, permitindo que a rede capture padrões complexos.

- **Segunda Camada Convolucional**:

```
Conv2D(64, kernel_size = (3, 3),
       activation = 'relu')
```

A segunda camada convolucional amplia a profundidade da rede, utilizando 64 filtros, o que permite a detecção de características mais detalhadas e específicas.

- **Terceira Camada Convolucional**:

```
Conv2D(128, kernel_size = (3, 3),
       activation = 'relu')
```

A terceira camada adiciona ainda mais complexidade ao modelo, com 128 filtros, capacitando a rede a aprender representações mais abstratas das imagens.

3.4.4 Aplicando o Max-Pooling

As camadas de Max Pooling são utilizadas para reduzir a dimensionalidade das representações obtidas pelas camadas convolucionais, mantendo as características mais relevantes e descartando informações menos significativas. No modelo implementado, foram inseridas duas camadas de Max Pooling, cada uma delas após uma camada convolucional.

```
MaxPooling2D(pool_size = (2, 2))
```

3.4.5 Camada de Achatamento (Flatten)

A camada de achatamento, representada pela função *Flatten()*, é uma etapa fundamental na transição entre as camadas convolucionais e as camadas totalmente conectadas de uma rede neural convolucional.

Após as operações de convolução e pooling, as características extraídas ainda estão organizadas em uma estrutura multidimensional. A função da camada *Flatten()* é transformar essa estrutura multidimensional em um único vetor unidimensional. Esse vetor linearizado pode então ser alimentado nas camadas densas subsequentes, permitindo que a rede aprenda as combinações dessas características para tomar decisões finais na classificação.

No modelo implementado, a camada *Flatten()* foi adicionada após a última camada de convolução, preparando a saída para a etapa final de classificação. Essa camada não altera o número total de unidades, apenas reorganiza os dados, preservando todas as informações necessárias para que a rede possa realizar a tarefa de classificação com eficiência.

```
model.add(Flatten())
```

3.4.6 Camada Densa e função Sigmoid

A camada densa, também conhecida como camada totalmente conectada (fully connected), é a etapa final na arquitetura de uma rede neural convolucional, onde cada neurônio está conectado a todos os neurônios da camada anterior. No modelo implementado, a camada densa é definida como *Dense(1, activation = 'sigmoid')*, o que significa que ela contém um único neurônio de saída e utiliza a função de ativação Sigmoid.

A função da camada densa é consolidar as características extraídas e transformadas pelas camadas anteriores em uma predição final. No caso específico deste modelo, a saída da camada densa é usada para realizar a tarefa de classificação binária (gato ou cachorro).

A escolha da função Sigmoid também é benéfica porque ela permite que a rede apresente uma saída suave e contínua, o

que pode ser útil na tarefa de ajustar o modelo durante o processo de treinamento.

3.4.7 Compilação do Modelo

Após a definição da arquitetura da rede neural, o próximo passo é compilar o modelo, o que envolve a configuração dos parâmetros necessários para o treinamento. No modelo implementado, a compilação é realizada com a função *model.compile()*, onde são especificados três principais componentes: a função de perda, o otimizador e as métricas de avaliação.

- **Função de perda:**

```
loss = 'binary_crossentropy'
```

Ela mede a diferença entre as probabilidades previstas pela rede e os rótulos reais, penalizando as previsões erradas de maneira que o modelo ajusta seus parâmetros para minimizar esse erro durante o treinamento.

- **Otimizador:**

```
optimizer = 'adam'
```

O Adam combina as vantagens dos algoritmos *AdaGrad* e *RMSProp*, ajustando dinamicamente a taxa de aprendizado para cada parâmetro da rede, o que ajuda a melhorar a eficiência e a convergência do treinamento.

- **Métricas de avaliação:**

```
metrics = ['accuracy']
```

Para monitorar o desempenho do modelo durante o treinamento e avaliação, a métrica utilizada é a *accuracy* (precisão). Esta métrica calcula a proporção de previsões corretas feitas pelo modelo em relação ao total de previsões realizadas, proporcionando uma visão clara da capacidade do modelo em classificar corretamente as imagens.

```
model.compile(loss =
'binary_crossentropy', optimizer =
 'adam', metrics = ['accuracy'])
```

3.4.8 Salvamento do Melhor Modelo

Para garantir que o modelo treinado possa ser recuperado e reutilizado posteriormente, é importante implementar um mecanismo de salvamento eficiente. No modelo implementado, utilizou-se o callback *ModelCheckpoint* do TensorFlow para salvar o modelo durante o treinamento. O callback foi configurado da seguinte forma:

```
filepath = "trained_model.keras"
```

Especifica o caminho do arquivo onde o modelo treinado será salvo. O nome do arquivo é “trained_model.keras”, e o modelo será armazenado no formato Keras, o que facilita o carregamento e a reutilização futura.

```
save_best_only = True
```

Define que apenas o modelo com o melhor desempenho, de acordo com a métrica monitorada, será salvo. Isso significa que somente o modelo com a menor perda de validação (*val_loss*) será armazenado, garantindo que você tenha sempre a melhor versão do modelo durante o treinamento.

```
monitor = "val_loss"
```

Especifica a métrica a ser monitorada para determinar o "melhor" modelo. No caso, a métrica escolhida é a perda de validação (*val_loss*), que avalia a qualidade das previsões do modelo em dados de validação que não foram vistos durante o treinamento.

```
ModelCheckpoint(filepath =
"trained_model.keras", save_best_only =
True, monitor = "val_loss")
```

3.4.9 Treinamento da Rede

O treinamento do modelo é realizado através da função *model.fit()*, que ajusta os parâmetros da rede neural com base nos dados de treinamento. No processo de treinamento do modelo, as seguintes configurações foram utilizadas:

```
‘train_dataset’
```

O conjunto de dados de treinamento, composto pelas imagens e rótulos que o modelo usa para aprender.

```
‘epochs = 60’
```

Define o número de épocas, ou seja, o número de vezes que o modelo passa por todo o conjunto de dados de treinamento. No caso, o modelo foi treinado por 60 épocas, permitindo que o algoritmo de aprendizado ajuste os parâmetros da rede de forma mais refinada ao longo de várias iterações.

```
‘validation_data = validation_dataset’
```

O conjunto de dados de validação é utilizado para avaliar o desempenho do modelo em dados que não foram vistos durante o treinamento. Isso ajuda a monitorar a capacidade de generalização do modelo e a identificar se ele está sobre ajustado aos dados de treinamento.

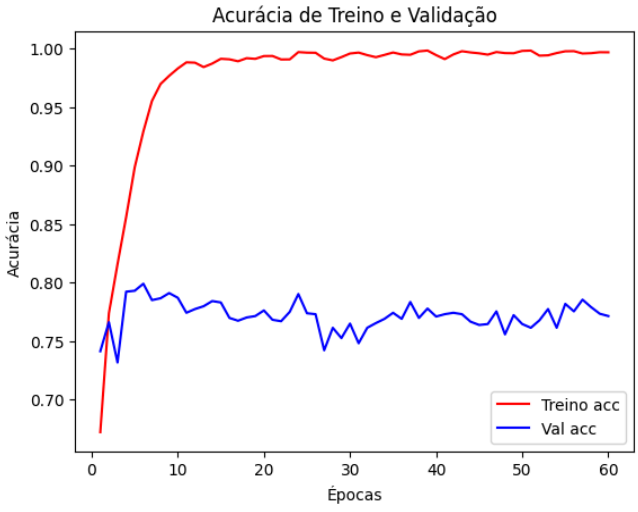
3.4.10 Especificação da Máquina e do Ambiente

O treinamento da rede foi executado em uma máquina com as seguintes especificações:

Sistema Operacional Windows 11 Pro 64-bit
Processador AMD Ryzen 5600H 3.30GHz 6 Núcleos
Placa de vídeo NVIDIA Geforce RTX 3060 Laptop
Memória RAM 16gb 3200MHz

4 Análise dos Resultados

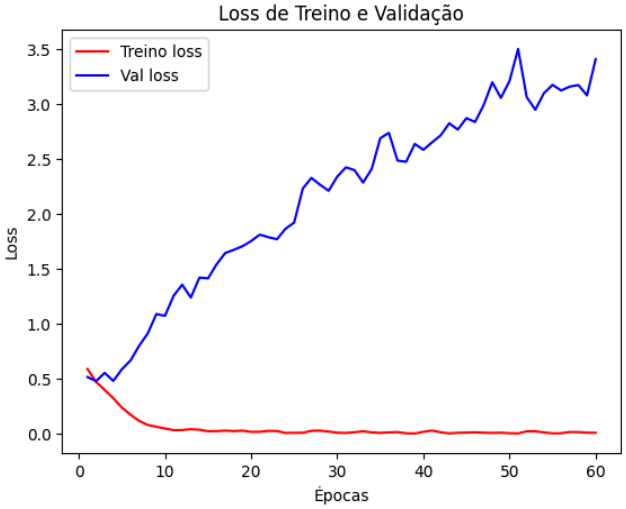
Os resultados obtidos foram analisados com base na acurácia e na matriz de perda gerada após a fase de testes:



O primeiro gráfico apresenta a acurácia de treino e validação ao longo de 60 épocas. Observa-se que a acurácia de treino (em vermelho) aumenta rapidamente nas primeiras épocas, estabilizando-se próximo de 100% após a 10ª época. Esse comportamento sugere que o modelo foi capaz de aprender bem os padrões presentes nos dados de treinamento.

No entanto, a acurácia de validação (em azul) apresenta uma trajetória diferente. Ela aumenta rapidamente nas primeiras épocas, mas logo oscila e se estabiliza em torno de 75% a 80%, sem alcançar o desempenho do conjunto de treino. Essa discrepância entre as acurácias de treino e validação indica que o modelo pode estar sofrendo de **OVERFITTING**. O modelo aprendeu a representar muito bem os dados de treino, mas não generaliza tão bem para dados que não foram vistos durante o treinamento.

Esse comportamento pode ser um sinal de que o modelo está capturando detalhes e ruídos específicos do conjunto de treinamento, mas esses detalhes não são necessariamente relevantes ou presentes no conjunto de validação. Assim, o modelo não consegue manter um alto desempenho quando exposto a novos dados.



O segundo gráfico ilustra a evolução da perda (loss) do modelo tanto no conjunto de treino quanto no conjunto de validação ao longo de 60 épocas. A perda de treino (em vermelho) diminui rapidamente nas primeiras épocas e se estabiliza em um valor muito baixo, próximo de zero, o que indica que o modelo se ajustou muito bem aos dados de treino.

Por outro lado, a perda de validação (em azul) apresenta um comportamento distinto. Após uma leve diminuição inicial, a perda de validação começa a aumentar gradualmente, demonstrando uma tendência de crescimento contínuo à medida que as épocas progridem. Esse aumento constante da perda de validação enquanto a perda de treino permanece baixa é um claro indício de **OVERFITTING**.

O overfitting ocorre quando o modelo se ajusta tão bem aos dados de treino que ele começa a capturar ruídos e padrões específicos que não são generalizáveis para novos dados, o que se reflete no aumento da perda de validação. Esse comportamento reforça a análise anterior, onde a acurácia de validação se estabiliza e oscila, enquanto a acurácia de treino continua a melhorar.

Na acurácia dos testes gerais o sistema alcançou o percentual de 78,30%. Uma taxa boa porém bem longe dos 98,91% do primeiro colocado do desafio na plataforma Kaggle, onde pegamos o dataset.

5 Conclusão

Neste estudo, exploramos as redes neurais convolucionais (CNNs) e sua aplicação na tarefa de classificação de imagens, utilizando as bibliotecas TensorFlow e Keras para implementação. A rede neural construída atingiu uma acurácia que se alinha com os padrões encontrados em trabalhos correlatos na literatura.

As CNNs têm se consolidado como uma ferramenta poderosa na visão computacional e no aprendizado de máquina, principalmente por sua capacidade de explorar as estruturas espaciais presentes nas imagens. Essa habilidade tem permitido que as CNNs superem muitos métodos tradicionais, especialmente em aplicações que exigem análise de grandes volumes de dados visuais.

Embora a aplicação discutida neste artigo seja relativamente simples, CNNs já são amplamente utilizadas em problemas mais desafiadores, como a detecção de objetos em tempo real e o reconhecimento facial, demonstrando a flexibilidade e eficácia dessa arquitetura.

Alguns pontos críticos para o sucesso da implementação merecem ser destacados: a preparação dos dados é fundamental, pois a eficácia do modelo depende significativamente de uma subdivisão adequada dos conjuntos de treino e teste, garantindo que o modelo não apenas aprenda, mas também generalize bem, o que pode reduzir o número de épocas necessárias e otimizar o uso de recursos. Além disso, a complexidade da rede e o volume de dados devem ser ajustados em conformidade; o número de camadas precisa ser balanceado com o tamanho do conjunto de dados para evitar tanto o subajuste quanto o sobreajuste. Por fim, o uso de técnicas de regularização, como dropout, batch normalization e data augmentation, é indispensável para alcançar uma acurácia robusta, ajudando a mitigar a variabilidade nos dados e a melhorar a consistência do treinamento.

Em conclusão, as redes neurais convolucionais provaram ser uma escolha eficaz para a resolução de problemas que envolvem dados visuais, sendo uma tecnologia promissora para futuras aplicações na visão computacional e em outros domínios do aprendizado de máquina.

Referências

[1]KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 2012, p. 1097-1105.

[2] GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016.

[3] SCHERER, Dominik; MÜLLER, Andreas; BEHNKE, Sven. Evaluation of pooling operations in convolutional architectures for object recognition. *International Conference on Artificial Neural Networks*, Springer, Berlin, Heidelberg, 2010, p. 92-101.

[4] HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE International Conference on Computer Vision*, 2015, p. 1026-1034.

[5] KINGMA, Diederik P.; BA, Jimmy. Adam: A method for stochastic optimization. *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[6] NAIR, Vinod; HINTON, Geoffrey E. Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, p. 807-814.

[7] LITJENS, Geert et al. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 2017.

[8] REDMON, Joseph et al. You Only Look Once: Unified, Real-Time Object Detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[9] GEIGER, Andreas et al. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research*, 2013.