UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Matheus Lázaro Honório da Silva - 201801523
Guilherme Araújo França - 202003569
Joviro Rocha Neto - 201702764

Trabalho prático - CNN para Classificação - Imagens de Raio-x - (Classes: Pneumonia ou Normal)

Inteligência Artificial

Goiânia
2024

Introdução

O avanço das tecnologias de aprendizado de máquina, especialmente no campo da visão computacional, tem aberto novas fronteiras no diagnóstico médico. Um domínio particularmente promissor é o uso de imagens de raio-X para a detecção de condições pulmonares, como a pneumonia. Este projeto concentra-se no uso de um dataset composto por 5.863 imagens de raio-X do tórax, provenientes de pacientes pediátricos com idades entre um e cinco anos, tratados no Centro Médico para Mulheres e Crianças de Guangzhou (Província da China). O dataset está publicamente disponibilizado por PAUL MOONEY, através da plataforma Kaggle, e denomina-se “Chest X-Ray Images (Pneumonia)” [1]. As imagens são categorizadas em duas classes principais: 'Pneumonia' e 'Normal'. Estruturado em três diretórios principais - 'treino', 'teste' e 'validação'.

Conforme estabelecido no escopo deste trabalho, foi utilizada uma baseline que utiliza CNNs (Redes Neurais Convolucionais), para a classificação das imagens de raio-x disponibilizadas pelo dataset “Chest X-Ray Images (Pneumonia)”, em Normal ou Pneumonia. A baseline denomina-se “Pneumonia/Normal ? - CNN Model”, sob autoria de RAFET CAN KANDAR [2], e também está publicamente disponível. Desta forma, este conjunto de dados fornece uma base sólida para a aplicação de técnicas de aprendizado profundo, mais notavelmente Redes Neurais Convolucionais (CNNs), na identificação automática de padrões indicativos de pneumonia em imagens de raio-X do tórax. Este trabalho possui como objetivo a tentativa de superar o modelo de CNN usado nesta baseline escolhida, utilizando a biblioteca Python Keras, criando-se um novo modelo que produza uma maior precisão.

Formulação do Problema

A detecção e diagnóstico precoces da pneumonia, especialmente em pacientes pediátricos, são cruciais para um tratamento eficaz e a redução de complicações. Tradicionalmente, esta tarefa depende da interpretação visual de imagens de raio-X por radiologistas experientes. No entanto, este processo pode ser desafiador, sujeito a variações e, ocasionalmente, a erros. Além disso, em áreas com falta de especialistas, pode haver um atraso significativo no diagnóstico. Diante disso, a automação do processo de diagnóstico utilizando técnicas de aprendizado de máquina apresenta uma oportunidade valiosa. O problema central deste projeto é desenvolver um modelo de CNN que possa analisar imagens de raio-X do tórax e identificar com precisão a presença de pneumonia, diferenciando-as de imagens normais. O desafio reside em garantir que o modelo seja altamente preciso, confiável e capaz de operar efetivamente em diferentes condições e variações de imagens.

Objetivo

O objetivo principal deste projeto é desenvolver um modelo de Rede Neural Convolucional (CNN) que seja capaz de classificar imagens de raio-X do tórax de pacientes pediátricos em duas categorias: 'Pneumonia' e 'Normal'. Este modelo visa auxiliar no diagnóstico rápido e preciso da pneumonia, reduzindo a dependência de análise manual por especialistas e acelerando o processo de tomada de decisão clínica.

Além disso, o projeto busca avaliar a eficácia das CNNs em interpretar nuances sutis em imagens médicas, um passo crucial para a implementação mais ampla de tecnologias de inteligência artificial no campo da saúde. Ao alcançar esses objetivos, o projeto fornecerá insights valiosos sobre a aplicação de técnicas de Redes Neurais Convolucionais em desafios médicos complexos.

Metodologia

Base de Referência (Baseline):

https://www.kaggle.com/code/rafetcan/pneumonia-normal-cnn-model

Para este projeto, a referência inicial ou "baseline" foi estabelecida com base no modelo de CNN para a classificação de imagens de raio-X do tórax como 'Pneumonia' ou 'Normal', desenvolvido por Rafet Can Kandar. Este modelo, disponível no Kaggle, serve como um ponto de partida para a avaliação e aprimoramento do desempenho. O objetivo é superar a eficácia deste modelo baseline em termos de precisão, sensibilidade e especificidade na classificação das imagens. O código do baseline fornece uma estrutura útil sobre como as imagens são processadas, a arquitetura da rede neural utilizada e as métricas de desempenho.

Conjunto de Dados:

https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

O conjunto de dados utilizado é o "Chest X-Ray Images (Pneumonia)", compilado por Paul Mooney, também disponível no Kaggle. Consiste em 5.863 imagens de raio-X do tórax em formato JPEG, categorizadas em 'Pneumonia' e 'Normal'. As imagens foram coletadas de pacientes pediátricos no Centro Médico para Mulheres e Crianças de Guangzhou. Este conjunto de dados está dividido em três grupos: treinamento, teste e validação, facilitando a implementação de técnicas de aprendizado supervisionado.

Implementação

Para fins de desenvolvimento, utilizamos a plataforma Colab. A implementação está disponível em: https://colab.research.google.com/drive/19beouNoji0zo2HIMGWW2l3hZmZUBknGX?usp=sharing. Devido à premissa de superar o modelo de CNN implementado por Rafet Can Kandar, utilizamos esta mesma baseline. Entretanto, optamos por disponibilizar os dados do dataset publicamente através do repositório no github: https://github.com/matheusLazaroCC-UFG/TrabalhoPratico-IA-CNN-Pneumonia-Normal.

- Na etapa de seleção de “Caminhos dos arquivos”, definimos variáveis para os diretórios de treinamento, teste e validação (variáveis train_dir, test_dir e val_dir).
- Na etapa de “Leitura de Imagens e Divisão Treino-Teste”, temos a implementação de uma função denominada picture_separation, que é responsável por separar as imagens em arrays de dados e rótulos. Esta função é utilizada em seguida e recebe como parâmetro o diretório de

treinamento, e retorna os arrays x_train, y_train, e uma lista de nomes das imagens de treinamento.

- x_train: array para armazenar os rótulos da imagens de treinamento.
- y_train: array para armazenar os dados da imagens de treinamento.
- image_list: lista para armazenar os nomes da imagens de treinamento.

Ainda nesta etapa, criamos um dataframe pandas, com os nomes e os rótulos das imagens de treinamento. Em seguida, exibimos as primeiras linhas do dataframe. Exemplo:

	images	target
0	PNEUMONIA/person1111_virus_1836.jpeg	1
1	PNEUMONIA/person1270_virus_2163.jpeg	1
2	PNEUMONIA/person1403_virus_2406.jpeg	1
3	PNEUMONIA/person613_bacteria_2479.jpeg	1
4	PNEUMONIA/person23_bacteria_83.jpeg	1

Após isso, exibimos as informações do dataframe de treinamento. Exemplo:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5216 entries, 0 to 5215
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   images  5216 non-null   object
 1   target  5216 non-null   int64 
dtypes: int64(1), object(1)
memory usage: 81.6+ KB
```

Em seguida, exibimos a contagem dos rótulos das imagens de treinamento:

```
1    3875
0    1341
Name: target, dtype: int64
```

Com isso, criamos e enviamos o caminho de validação, usando a mesma função picture_separation, recebendo como parâmetro o diretório de validação, e retornando:

- x_val: array para armazenar os rótulos da imagens de validação.
- y_val: array para armazenar os dados da imagens de validação.
- img_val: lista para armazenar os nomes da imagens de validação.

Temos então o procedimento de exibir as primeiras linhas do dataframe de validação. Exemplo:

	images	target
0	PNEUMONIA/person1947_bacteria_4876.jpeg	1
1	PNEUMONIA/person1951_bacteria_4882.jpeg	1
2	PNEUMONIA/person1950_bacteria_4881.jpeg	1
3	PNEUMONIA/person1946_bacteria_4875.jpeg	1
4	PNEUMONIA/person1946_bacteria_4874.jpeg	1

Em seguida, exibimos a contagem do rótulos das imagens de validação:

```
1    8
0    8
Name: target, dtype: int64
```

Criamos e enviamos o caminho de teste, usando a mesma função picture_separation, recebendo como parâmetro o diretório de testes, e retornando:

- x_test: array para armazenar os rótulos da imagens de teste.
- y_test: array para armazenar os dados da imagens de teste.
- img_test: lista para armazenar os nomes da imagens de teste.

Temos então o procedimento de exibir as primeiras linhas do dataframe de validação. Exemplo:

	images	target
619	NORMAL/NORMAL2-IM-0274-0001.jpeg	0
620	NORMAL/IM-0085-0001.jpeg	0
621	NORMAL/NORMAL2-IM-0059-0001.jpeg	0
622	NORMAL/IM-0029-0001.jpeg	0
623	NORMAL/IM-0107-0001.jpeg	0

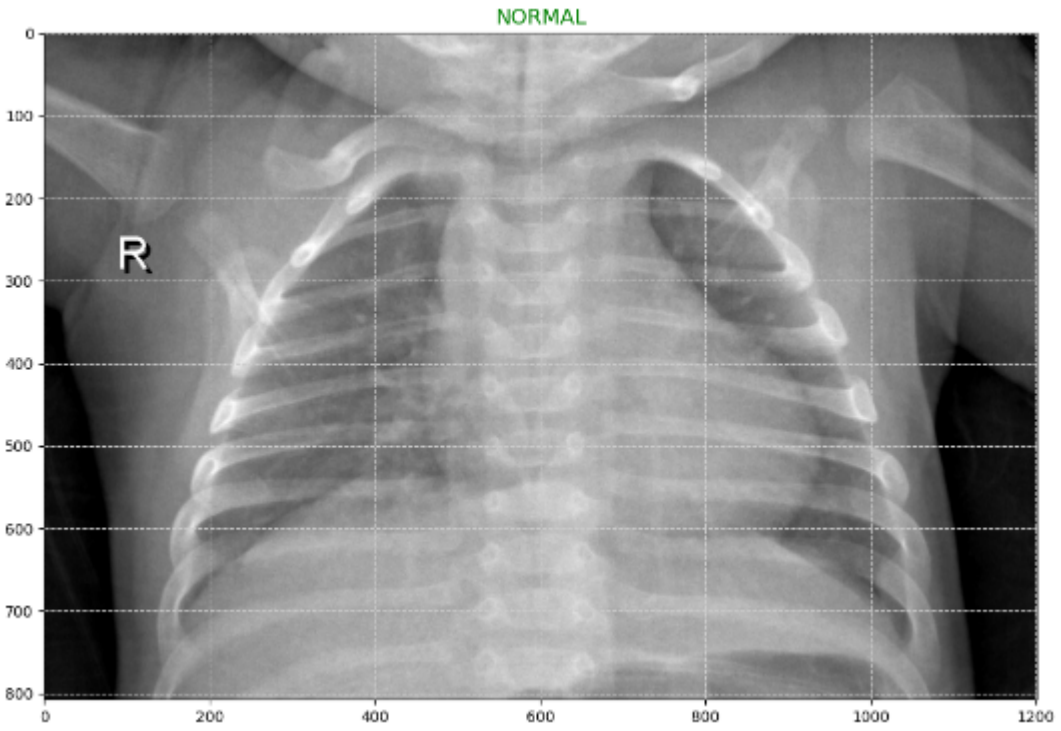
Em seguida, exibimos a contagem do rótulos das imagens de validação:

```
1    390
0    234
Name: target, dtype: int64
```

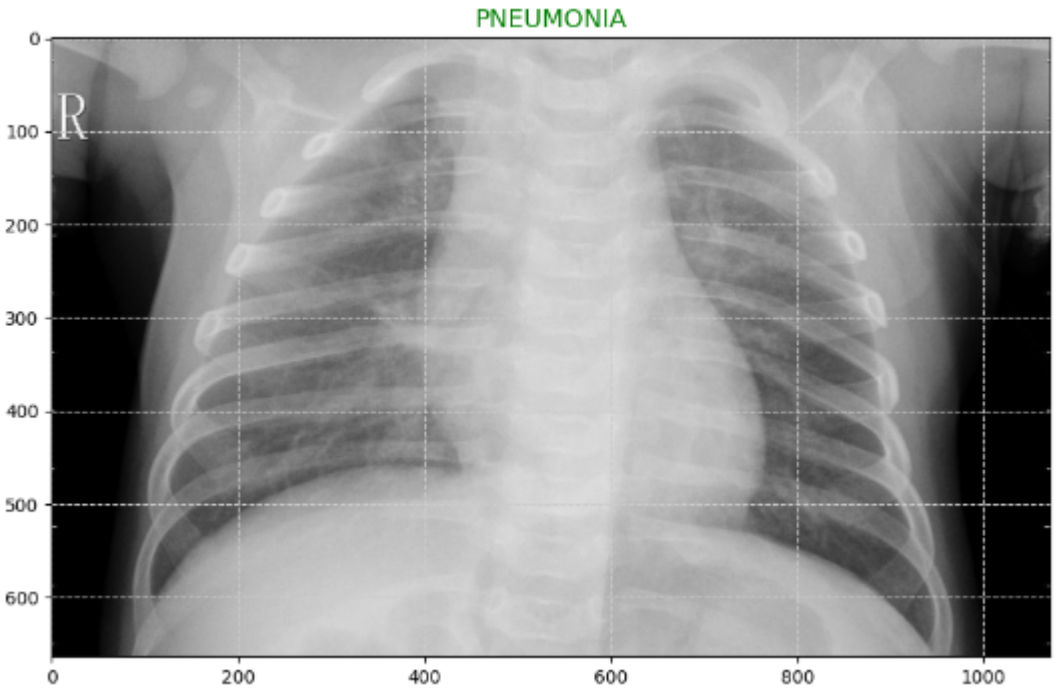
- A próxima etapa caracteriza-se por concatenar os dataframes de treinamento, teste e validação em um único dataframe chamado full_data. Após isso, exibimos as primeiras e as últimas linhas deste dataframe.

	images	target
0	PNEUMONIA/person1111_virus_1836.jpeg	1
1	PNEUMONIA/person1270_virus_2163.jpeg	1
2	PNEUMONIA/person1403_virus_2406.jpeg	1
3	PNEUMONIA/person613_bacteria_2479.jpeg	1
4	PNEUMONIA/person23_bacteria_83.jpeg	1
	images	target
5851	NORMAL/NORMAL2-IM-1431-0001.jpeg	0
5852	NORMAL/NORMAL2-IM-1437-0001.jpeg	0
5853	NORMAL/NORMAL2-IM-1440-0001.jpeg	0
5854	NORMAL/NORMAL2-IM-1436-0001.jpeg	0
5855	NORMAL/NORMAL2-IM-1442-0001.jpeg	0

- A próxima etapa caracteriza-se por, partimos para o uso do dataframe full_data. Exemplo de de imagem de treinamento do tipo NORMAL.:



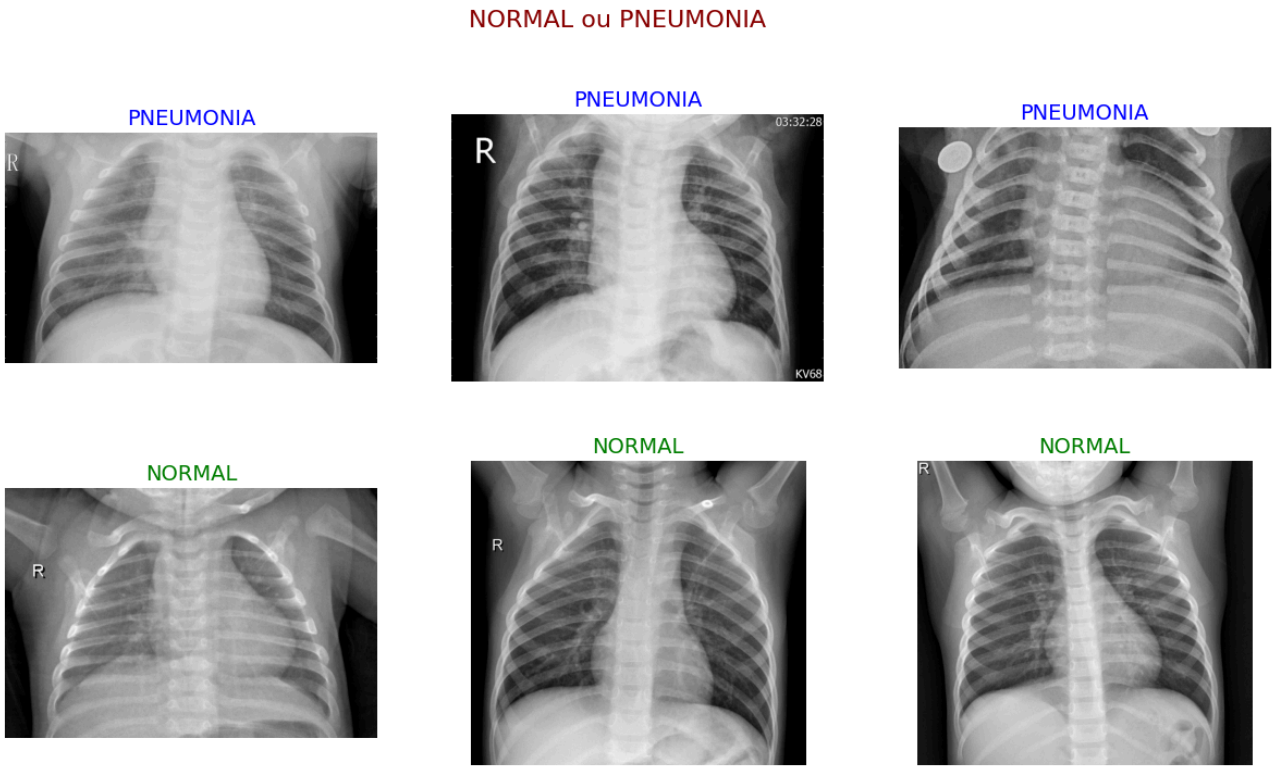
Exemplo de imagem de treinamento do tipo PNEUMONIA:



Exibimos a contagem dos rótulos das imagens do dataframe full_data

```
1    4273
0    1583
Name: target, dtype: int64
```

- Temos então o procedimento de exibir seis imagens do dataframe full_data, três normais e três com pneumonia, em uma grade de 2x3. Exemplo:



Em seguida, obtemos quantas classes existem:

```
NumberOfClass:  2
```

- Após isso, temos o procedimento de Aumento de dados, em que definimos o tamanho do loto de treinamento como 32, criamos um gerador de imagens de treinamento, que aplica algumas transformações nas imagens para aumentar a variedade dos dados. As transformações são:
    - Normalizar os valores dos pixels entre 0 e 1
    - Aplicar um ângulo de cisalhamento nas imagens
    - Espelhar as imagens horizontalmente
    - Aplicar um fator de zoom nas imagens

  Em seguida, criamos um gerador de imagens de teste, que apenas normaliza os valores dos pixels entre 0 e 1. Criamos um gerador de imagens de validação, que apenas normaliza os valores dos pixels entre 0 e 1. Criamos um iterador de imagens de treinamento, que lê as imagens do diretório de treinamento e as converte em arrays de dados e rótulos. Fazemos o mesmo para as imagens de teste e de validação.

Resultados

Desenvolvimento do Modelo CNN com Keras:

Implementamos um exemplo de como construir, treinar e avaliar um modelo de classificação binária de imagens usando o framework Keras, que é uma biblioteca de alto nível para desenvolver redes neurais em Python. O modelo usa uma arquitetura de rede neural convolucional (CNN), que é um tipo de rede neural que se destaca no processamento e análise de imagens digitais. Uma CNN usa uma variação de perceptrons multicamada que aplicam a operação de convolução para extrair características relevantes das imagens.

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense, Ba
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import EarlyStopping, LearningRateScheduler
import math
```

Essas linhas importam as classes e funções necessárias do Keras e do módulo math.

A seguir, o código define uma função de ajuste do learning rate, que é usada pelo callback de agendamento da taxa de aprendizado:

```
# Função de ajuste do learning rate
def lr_schedule(epoch):
    lr = 0.0001
    if epoch > 10:
        lr *= 0.1
    return lr
```

Essa função recebe o número da época atual como argumento e retorna o valor da taxa de aprendizado para essa época. A função define um valor inicial de 0.0001 e reduz esse valor em 90% se a época for maior que 10. Isso significa que o modelo começa com uma taxa de aprendizado relativamente alta e diminui gradualmente à medida que o treinamento avança.

Em seguida, o código constrói o modelo sequencial, adicionando as camadas convolucionais, de agrupamento, de ativação, de dropout, de achatamento, densas e de normalização em lote:

```
# Construção do Modelo (CNN) - ADAPTADO
model = Sequential()

model.add(Conv2D(64, (3, 3), input_shape=X_train.shape[1:]))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D())

model.add(Conv2D(128, (3, 3)))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D())

model.add(Conv2D(256, (3, 3)))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D())

model.add(Flatten())
model.add(Dense(512))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Dense(1))
model.add(Activation("sigmoid"))
```

O modelo consiste em três blocos, cada um com uma camada convolucional, uma camada de ativação ReLU, uma camada de normalização em lote e uma camada de agrupamento máximo. O número de filtros convolucionais aumenta de 64 para 128 e depois para 256, enquanto o tamanho dos filtros permanece constante em (3, 3). O formato da entrada é definido pela forma do conjunto de dados de treinamento, que é uma matriz 4D com as dimensões (número de amostras, altura, largura e canais). O modelo assume que as imagens são em escala de cinza, portanto, o número de canais é 1. As camadas convolucionais extraem características de baixo, médio e alto nível das imagens, enquanto as camadas de agrupamento reduzem o tamanho das imagens pela metade, aplicando a operação de máximo a uma janela de (2, 2).

Após os blocos convolucionais, o modelo adiciona uma camada de achatamento, que transforma as imagens 3D em vetores 1D, seguida por uma camada densa com 512 neurônios e uma camada de ativação ReLU. Essas camadas realizam operações de alto nível, como combinar as características extraídas pelas camadas convolucionais. Em seguida, o modelo adiciona uma camada de normalização em lote e uma camada de dropout com uma taxa de 0.3, que desativa 30% dos neurônios da camada anterior durante o treinamento. Essas camadas ajudam a melhorar o desempenho e a estabilidade do modelo.

Finalmente, o modelo adiciona uma camada densa com um único neurônio e uma camada de ativação sigmoide. Essas camadas realizam a operação de classificação binária, que é o objetivo do modelo. A camada densa produz um valor escalar entre 0 e 1, que representa a probabilidade de a imagem de entrada pertencer à classe positiva. A camada de ativação sigmoide aplica a função sigmoide a esse valor, que é definida como:

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

Essa função tem a propriedade de mapear qualquer valor real para um valor entre 0 e 1, o que é conveniente para representar probabilidades. O modelo pode então usar um limiar, como 0.5, para decidir se a imagem de entrada pertence à classe positiva ou negativa.

Depois de construir o modelo, o código compila o modelo com o otimizador Adam e a função de perda de entropia cruzada binária:

```
# Compilar o modelo com otimizador Adam e learning rate schedule
model.compile(loss="binary_crossentropy",
              optimizer=Adam(learning_rate=0.0001),
              metrics=["accuracy"])
```

A função de perda de entropia cruzada binária é uma medida de quão bem o modelo se ajusta aos dados de treinamento. Ela compara a probabilidade prevista pelo modelo com a classe verdadeira da imagem de entrada, e penaliza o modelo se houver uma grande diferença entre elas. A função de perda é definida como:

L(y, y^) = -ylog(y^) - (1 - y) log(1 - y^)

Onde y é a classe verdadeira da imagem de entrada, que pode ser 0 ou 1, e y^ é a probabilidade prevista pelo modelo, que é um valor entre 0 e 1. A função de perda é mínima quando y e y^ são iguais, e máxima quando y e y^ são opostos.

O otimizador Adam é responsável por atualizar os pesos do modelo com base no gradiente da função de perda, usando uma taxa de aprendizado adaptativa e um momento adaptativo. A taxa de aprendizado é um hiperparâmetro que controla o tamanho do passo que o otimizador dá na direção do mínimo da função de perda. O momento é um termo que ajuda o otimizador a acelerar na direção do mínimo e a evitar oscilações. O otimizador Adam usa uma taxa de aprendizado inicial de 0.0001, que é ajustada pelo callback de agendamento da taxa de aprendizado a cada época.

O código também especifica a métrica de precisão, que é uma medida de quão bem o modelo classifica as imagens de entrada. Ela compara a classe prevista pelo modelo com a classe verdadeira da imagem de entrada, e conta quantas vezes elas são iguais. A precisão é definida como:

$$A = \frac{TP + TN}{TP + TN + FP + FN}$$

Onde TP é o número de verdadeiros positivos, ou seja, imagens que pertencem à classe positiva e são classificadas corretamente pelo modelo. TN é o número de verdadeiros negativos, ou seja, imagens que pertencem à classe negativa e são classificadas corretamente pelo modelo. FP é o número de falsos positivos, ou seja, imagens que pertencem à classe negativa e são classificadas incorretamente pelo modelo. FN é o número de falsos negativos, ou seja, imagens que pertencem à classe positiva e são classificadas incorretamente pelo modelo. A precisão é um valor entre 0 e 1, sendo 1 o melhor e 0 o pior.

Em seguida, o código configura os geradores de dados com aumento de dados:

```python
# Configurar geradores de dados com aumento de dados
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)

val_datagen = ImageDataGenerator(rescale=1./255)

# Criar iteradores de dados com aumento de dados
train_generator_augmented = train_datagen.flow_from_directory(
    train_dir,
    target_size=(64, 64),
    batch_size=batch_size,
    color_mode='grayscale',
    class_mode='binary')

val_generator_augmented = val_datagen.flow_from_directory(
    val_dir,
    target_size=(64, 64),
    batch_size=batch_size,
    color_mode='grayscale',
    class_mode='binary')
```

Os geradores de dados são objetos que geram lotes de dados de imagem com aumento de dados, ou seja, com transformações aleatórias, como rotação, cisalhamento, zoom e inversão horizontal. Isso ajuda a aumentar a variedade e a quantidade de dados de treinamento e validação, melhorando a generalização do modelo. Os geradores de dados também redimensionam as imagens para um tamanho alvo de (64, 64) e as convertem para escala de cinza, se necessário. Além disso, os geradores de dados reescal am os valores dos pixels das imagens para o intervalo [0, 1], dividindo-os por 255. Isso ajuda a normalizar os dados de entrada e a melhorar o treinamento do modelo. Os geradores de dados também definem o modo de classe como binário, o que significa que eles geram rótulos binários para as imagens, de acordo com os nomes das subpastas nas pastas de treinamento e validação.

Em seguida, o código configura os callbacks de parada antecipada e agendamento da taxa de aprendizado:

```python
# Configurar o EarlyStopping com base na precisão de validação
early_stopping = EarlyStopping(monitor='val_accuracy', mode='max', verbose=1, patience=5, restore_best_weights=True)

# Configurar o LearningRateScheduler
lr_scheduler = LearningRateScheduler(lr_schedule)
```

O callback de parada antecipada monitora a métrica de precisão de validação, que é a precisão do modelo nos dados de validação, que são usados para avaliar o modelo a cada época, sem usá-los para o treinamento. O callback interrompe o treinamento se a precisão de validação não melhorar após 5 épocas consecutivas, ou seja, se o modelo atingir um platô de desempenho. O callback também restaura os melhores pesos encontrados durante o treinamento, que são os pesos que correspondem à maior precisão de validação.

O callback de agendamento da taxa de aprendizado usa a função de ajuste do learning rate definida anteriormente para alterar a taxa de aprendizado do otimizador a cada época, de acordo com o número da época.

Finalmente, o código treina o modelo usando os geradores de dados de treinamento e validação com aumento de dados, e os callbacks de parada antecipada e agendamento da taxa de aprendizado:

```
# Treinar o modelo com LearningRateScheduler e EarlyStopping
history = model.fit(
    train_generator_augmented,
    epochs=30,
    validation_data=val_generator_augmented,
    callbacks=[early_stopping, lr_scheduler])
```

O modelo é treinado por no máximo 30 épocas, mas pode ser interrompido antes pelo callback de parada antecipada. A cada época, o modelo processa todos os dados de treinamento em lotes, usando o gerador de dados de treinamento com aumento de dados. A cada lote, o modelo calcula a função de perda e a métrica de precisão, e atualiza os pesos com o otimizador Adam, usando a taxa de aprendizado ajustada pelo callback de agendamento da taxa de aprendizado. A cada época, o modelo também avalia o seu desempenho nos dados de validação, usando o gerador de dados de validação sem aumento de dados. O modelo registra o histórico de perda e precisão de treinamento e validação, que podem ser usados para visualizar o progresso do treinamento.

Depois de treinar o modelo, o código avalia o modelo usando o gerador de dados de teste e exibe a precisão e a perda do modelo:

```
# Avaliar o modelo usando o gerador de dados de teste e exibir a precisão e a perda do modelo
evaluation = model.evaluate(test_generator)
```

O gerador de dados de teste é um objeto que gera lotes de dados de imagem sem aumento de dados, a partir da pasta de teste, que contém imagens que não foram usadas para o treinamento ou validação do modelo. O modelo calcula a função de perda e a métrica de precisão nos dados de teste, que são usados para medir o quão bem o modelo generaliza para novos dados. O modelo armazena os valores de perda e precisão de teste na variável evaluation, que podem ser impressos ou retornados.

Comparação com este novo modelo CNN

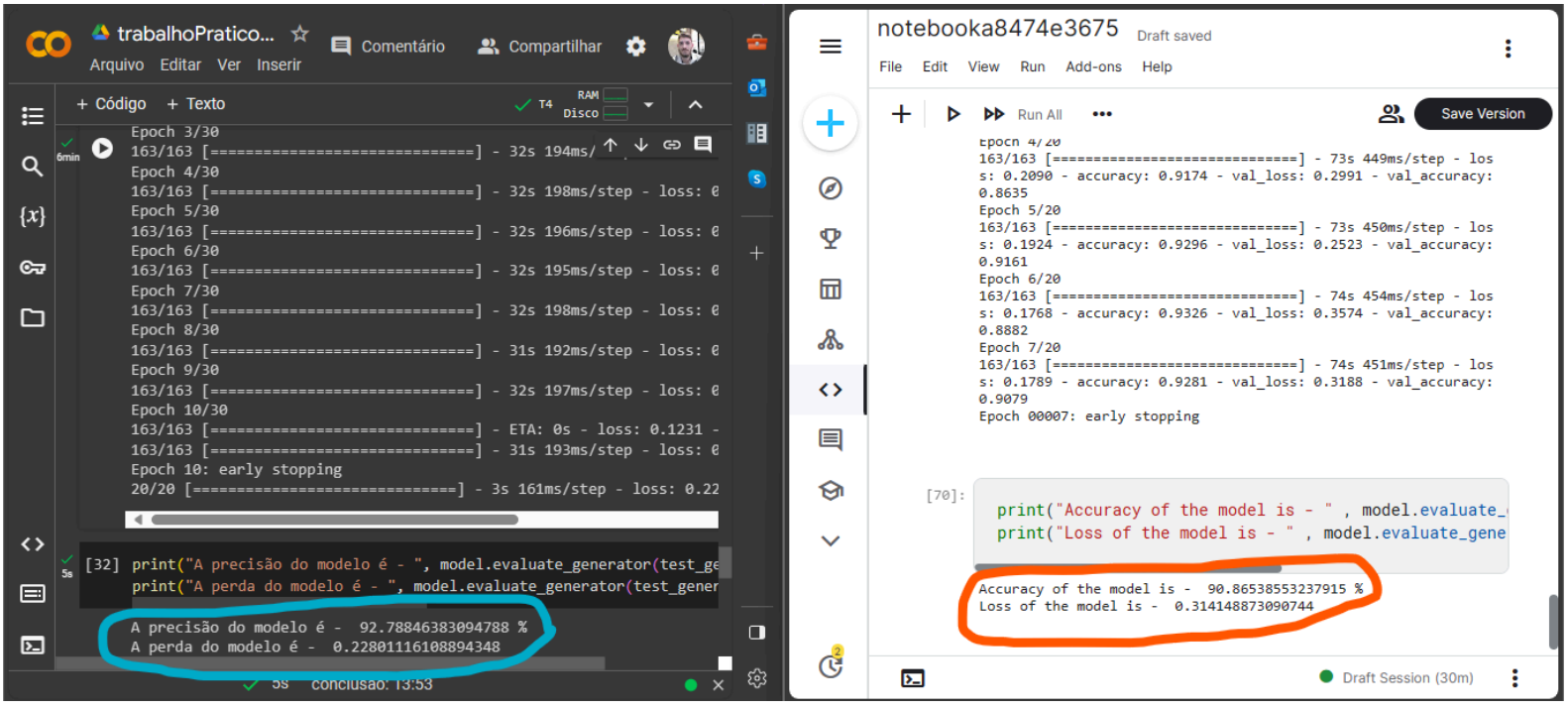
Os modelos usam diferentes números de filtros convolucionais, que são os parâmetros que definem o número de características que os modelos podem extrair das imagens de entrada. Este novo modelo CNN usa 64, 128 e 256 filtros nas três camadas convolucionais, respectivamente, enquanto o modelo CNN original usa 32, 32 e 64 filtros nas três camadas convolucionais, respectivamente. Isso significa que este novo modelo CNN tem mais capacidade de aprender características mais complexas e variadas das imagens de entrada, mas também tem mais risco de sobreajuste e mais custo computacional.

Os modelos usam diferentes números de neurônios nas camadas densas, que são os parâmetros que definem o número de operações de alto nível que os modelos podem realizar nas características extraídas pelas camadas convolucionais. Este novo modelo CNN usa 512 neurônios na camada densa, enquanto o modelo CNN original usa 1024 neurônios na camada densa. Isso significa que este novo modelo CNN tem menos capacidade de combinar as características extraídas pelas camadas convolucionais, mas também tem menos risco de sobreajuste e menos custo computacional.

Os modelos usam diferentes taxas de dropout, que são os hiperparâmetros que definem a fração de neurônios que são desativados aleatoriamente durante o treinamento. Este novo modelo CNN usa uma taxa de dropout de 0.3, enquanto o modelo CNN original usa uma taxa de dropout de 0.4. Isso significa que este novo modelo CNN desativa 30% dos neurônios da camada anterior durante o treinamento, enquanto o modelo CNN original desativa 40% dos neurônios da camada

anterior durante o treinamento. Isso afeta o nível de regularização que os modelos aplicam aos seus pesos, ou seja, o quanto eles evitam o sobreajuste aos dados de treinamento. Uma taxa de dropout mais alta significa uma regularização mais forte, que pode ajudar a prevenir o sobreajuste, mas também pode prejudicar o aprendizado do modelo se for muito alta. Uma taxa de dropout mais baixa significa uma regularização mais fraca, que pode ajudar o modelo a aprender mais, mas também pode aumentar o risco de sobreajuste se for muito baixa.

Os modelos usam diferentes otimizadores, que são os algoritmos que atualizam os pesos dos modelos com base no gradiente da função de perda. Este novo modelo CNN usa o **otimizador Adam**, que é um tipo de algoritmo que usa uma taxa de aprendizado adaptativa e um momento adaptativo. o modelo CNN original usa o otimizador **RMSprop**, que é um tipo de algoritmo que usa uma taxa de aprendizado fixa e um decaimento do gradiente quadrático. Isso significa que este novo modelo CNN ajusta a taxa de aprendizado do otimizador a cada época, de acordo com o callback de agendamento da taxa de aprendizado, que reduz a taxa de aprendizado em 90% se a época for maior que 10, enquanto o modelo CNN original usa uma taxa de aprendizado constante de 0.001. Isto afeta a velocidade e a precisão com que os modelos encontram o mínimo da função de perda. Uma taxa de aprendizado adaptativa significa que o modelo pode se adaptar às mudanças na superfície da função de perda, acelerando quando o gradiente é grande e desacelerando quando o gradiente é pequeno. Uma taxa de aprendizado fixa significa que o modelo usa o mesmo tamanho de passo na direção do mínimo da função de perda, independentemente do gradiente. Uma taxa de aprendizado adaptativa pode ajudar o modelo a encontrar o melhor valor para a taxa de aprendizado, que é um dos hiperparâmetros mais importantes do modelo. Uma taxa de aprendizado fixa pode dificultar o ajuste da taxa de aprendizado, que pode ser muito alta ou muito baixa para o modelo.



Á esquerda, está o resultado do novo modelo CNN implementado com Keras

À direita, está o resultado do modelo original, executado no kaggle

---

Conclusão

Em conclusão, ao analisar os resultados dos dois modelos de CNNs para classificação de imagens de pneumonia versus imagens normais, fica evidente que o modelo CNN original adaptado superou o baseline, alcançando uma acurácia mais elevada. Esse desempenho superior é crucial em um contexto médico, como a classificação de pneumonia, onde a precisão na identificação de casos é fundamental para o diagnóstico e tratamento adequado dos pacientes.

O modelo adaptado apresentou várias melhorias em relação ao baseline, como o aumento do número de filtros nas camadas convolucionais, o uso de normalização em lote, o aumento de dados e uma taxa de aprendizado adaptativa. Essas melhorias permitiram ao modelo adaptado extrair características mais discriminativas das imagens, reduzir o overfitting e generalizar melhor para casos não vistos. A utilização de técnicas como aumento de dados também é particularmente relevante em problemas médicos, onde a disponibilidade de dados limitados é comum.

Portanto, para aplicações de classificação de pneumonia em imagens médicas, é aconselhável adotar abordagens mais avançadas e adaptadas, como o novo modelo apresentado. Essas técnicas têm o potencial de melhorar significativamente a precisão do diagnóstico, proporcionando resultados mais confiáveis e auxiliando os profissionais de saúde no tratamento adequado dos pacientes.

Referências

[1] MOONEY, Paul. Chest X-Ray Images (Pneumonia). 2018. Disponível em: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia. Acesso em: 25 jan. 2024.

[2] KANDAR, Rafet Can. Pneumonia/Normal ? - CNN Model. 2024. Disponível em: https://www.kaggle.com/code/rafetcan/pneumonia-normal-cnn-model. Acesso em: 25 jan. 2024.

[3] TENSORFLOW. Keras. 2024. Disponível em: https://www.tensorflow.org/guide/keras?hl=pt-br. Acesso em: 25 jan. 2024.