Relatório de PDI

Luan Lavandoski Guarnieri e Maria Eduarda Krutzsch

Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

{lguarnieri, mekrutzsch}@furb.br

1 Introdução

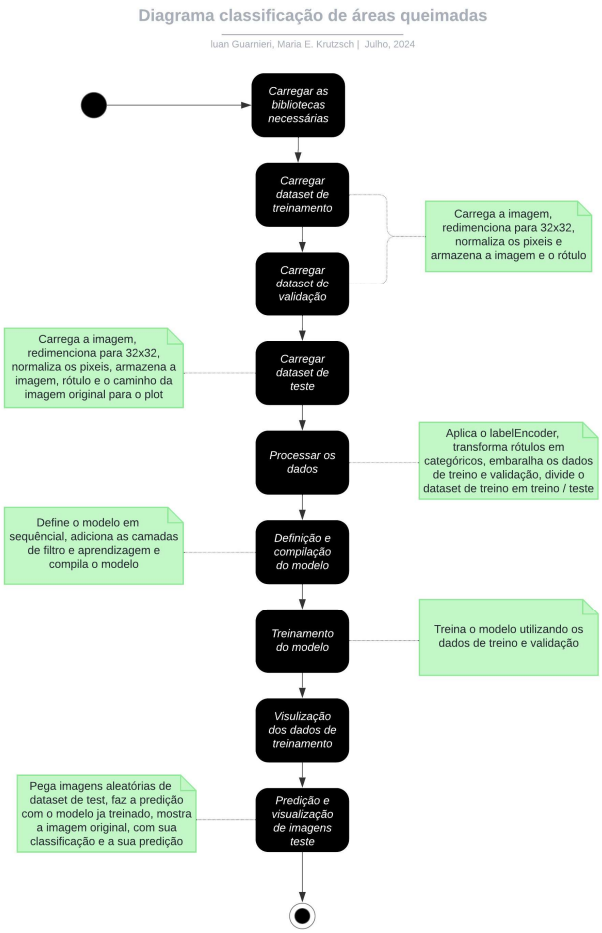
Este trabalho tem como objetivo descrever o desenvolvimento do trabalho final da disciplina de Processamento de Imagens, no qual optou-se por implementar um classificador CNN (Convolutional Neural Networks) para classificação binária de imagens de incêndios florestais. O classificador tem por objetivo identificar se uma imagem de satélite contém ou não um incêndio.

2 Desenvolvimento

2.1 Conjunto de Dados

A base de dados, é composta por mais de 40 mil imagens do Canadá de satélite, divididas em duas classes: “wildfire”, que representa incêndio, e “nowildfire”, que representa uma imagem sem incêndio. O dataset se encontra dividido em diretórios separados para treinamento (70%), teste (15%) e validação (15%). As imagens foram carregadas, redimensionadas para 32x32 pixels e normalizadas dividindo os valores dos pixels por 255, para que ficassem no intervalo [0,1]. A normalização ajuda a estabilizar o treinamento, incentivando o modelo a aprender representações mais gerais, ao invés de memorizar valores específicos. A figura 1 apresenta o diagrama com as atividades que o algoritmo realiza.

Figura 1 – Diagrama de atividades



Fonte: elaborado pelo autor.

2.2 Divisão dos dados

Após o carregamento das imagens com o redimensionamento e normalização, é aplicado **LabelEncoder** para codificar os rótulos das imagens, da biblioteca **scikit-learn**. Para problemas de classificação, a função de perda usada durante o treinamento espera que os rótulos de entrada sejam numéricos, e, preferencialmente, no formato categórico. Sendo assim, os valores foram convertidos para o formato categórico usando **to_categorical**. Em seguida, as imagens de treinamento e validação passam pela função **shuffle**, para evitar um modelo enviesado, pois é possível que os dados coletados contenham padrões ou sequências específicas. Por fim, os dados são divididos em treino e teste, utilizando uma divisão de 80% para treino e 20% para teste.

2.3 Construção do modelo CNN

A imagem 1 representa como foi implementado o modelo. O modelo foi implementado usando a biblioteca Keras com TensorFlow e incluiu algumas camadas como a **Conv2D,** que é responsável por aplicar filtros às imagens de entrada para extrair características. Neste modelo, foram usadas duas camadas convolucionais. **MaxPooling2D** também foi aplicado, e reduz a dimensionalidade e o número de parâmetros do modelo, enquanto preserva características mais importantes. Foi utilizado um pool de tamanho 2x2. **BatchNormalization** normaliza as ativações da camada anterior, acelerando o treinamento. **Dropout** é utilizada para evitar o overfitting. Ela é responsável por aleatoriamente definir unidades de entrada como 0. Por fim, camadas densas (**dense),** realizam a classificação final, conectando todas as características extraídas pelas camadas anteriores.

Imagem 1 – Código responsável por criar o modelo com parâmetros

```
183  # ============================== Construir o modelo ==============================
184
185  model = Sequential()
186
187  model.add(Conv2D(32, (3, 3), padding='same', input_shape=(32, 32, 3), activation='relu'))
188  model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
189  model.add(MaxPooling2D(pool_size=(2, 2)))
190  model.add(BatchNormalization())
191  model.add(Dropout(0.4))
192
193  model.add(Flatten())
194  model.add(Dropout(0.4))
195  model.add(Dense(64, activation='relu'))
196  model.add(Dense(2, activation='softmax'))
197
198  model.summary()
```

Fonte: elaborado pelo autor.

2.4 Treinamento do modelo

O processo de treinamento do modelo incluiu o uso de call-backs para gerenciar a taxa de aprendizado e evitar o overfitting. Callbacks são funções auxiliares que podem ser aplicadas para ajustar o desempenho. Neste modelo, foram utilizados dois: **ReduceLROnPlateau**, que ajusta dinamicamente a taxa de aprendizado quando uma métrica para de melhorar, monitorando a perda de validação (**val_loss**); e **EarlyStopping**, que interrompe o treinamento quando uma métrica para de melhorar, prevenindo o overfitting. A imagem 2, a seguir, representa como foi implementado o

treinamento do modelo.

Imagem 2 – Código responsável por treinar o modelo

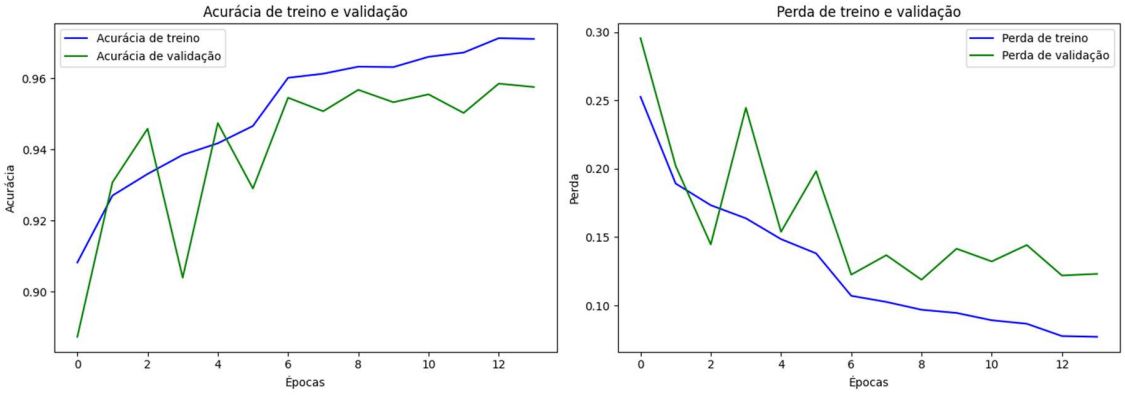
```
# ============================ Treinamento do modelo ============================

# callbacks para redução de learning rate e parada antecipada
# serve para controlar o overfitting
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=0.0001)
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

history = model.fit(X_train, Y_train,
                    validation_data=(validation_dataset, validation_labels_cat),
                    batch_size=32,
                    epochs=15,
                    callbacks=[reduce_lr, early_stopping])

```

Fonte: elaborado pelo autor.

3 Resultados e discussões

Em relação a acurácia e perda ao longo das épocas, a acurácia do treino aumenta de forma constante, enquanto a acurácia de validação flutua um pouco, mas consegue se manter alta. A perda do treino diminui continuamente, o que é um sinal positivo. A perda de validação também diminui, embora exista algumas flutuações. O gráfico 1 a seguir demonstra os resultados obtidos em uma execução.

Gráfico 1 – Demonstração da acurácia e da perda



Fonte: elaborado pelo autor.

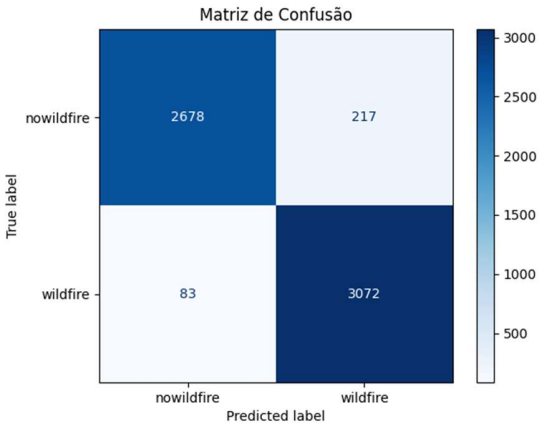
Os gráficos de acurácia e perda mostram um pequeno sinal de overfitting, apesar dos parâmetros inseridos e das funções de call-back. Em algumas execuções, a acurácia de validação apresentou variações e não melhora após certas épocas, o que faz com que as funções de call-back acabem interrompendo o treinamento. A matriz de confusão que o algoritmo gera mostra que o modelo tem um bom desempenho para o conjunto de teste, apesar de que há um número significativo de falsos positivos e falsos negativos.

4 Conclusões

Em geral, o modelo apresentou resultados bons. Em média, nas execuções do algoritmo, apresentou na maior parte dos casos acurácia superior a 95%, e cerca de 92% de acurácia para o conjunto de teste.

Como problemas identificados, podemos citar a presença ainda do overfitting, pelo fato de a acurácia de validação não melhorar após certas épocas em algumas execuções. Outro problema seria a presença de falsos positivos e negativos nas matrizes de confusão. A imagem 3 a seguir, ilustra a matriz de confusão gerada do treinamento do modelo.

Imagem 3 – Matriz de confusão gerada pelo algoritmo



Fonte: elaborado pelo autor.

Por fim, como melhorias, poderíamos explorar outras técnicas para a regularização, como dropout maior ou uma regularização L2/L1 para diminuir o overfitting. Outra abordagem de melhoria também seria o ajuste de hiperparâmetros, realizando testes para melhorar os resultados.

5 Referências

[1] IBM. Convolutional Neural Networks. Disponível em: https://www.ibm.com/br-pt/topics/convolutional-neural-networks. Acesso em: 30 jun. 2024.

[2] Keras Documentation. Regularization Layers. Disponível em: https://keras.io/api/layers/regularization_layers. Acesso em: 30 jun. 2024.

[3] TensorFlow Documentation. Keras Layers. Disponível em: https://www.tensorflow.org/api_docs/python/tf/keras/layers. Acesso em: 30 jun. 2024.

[4] Z. H. Jarrallah and M. A. A. Khodher, "Satellite Images Classification Using CNN :A Survey," *2022 International Conference on Data Science and Intelligent Computing (ICDSIC)*, Karbala, Iraq, 2022, pp. 111-116, doi: 10.1109/ICDSIC56987.2022.10075828. Acesso em: 30/06/2024. https://ieeexplore.ieee.org/document/10075828

[5] Wildfire Prediction Dataset (Satellite images). Disponível em: https://www.kaggle.com/datasets/abdelghaniaaba/wildfire-prediction-dataset/data. Acesso em: 28 jun 2024.

[6] Wildfire prediction CNN. Disponível em: https://www.kaggle.com/code/vaishnavipatil4848/wildfire-prediction-cnn/notebook. Acesso em: 29 jun 2024.