Instituto Tecnológico de Aeronáutica

Divisão de Ciência da Computação

Bolsa de Valores

Redes Neurais com Aprendizagem Clássica e Profunda - 2º Laboratório: Redes Neurais Convolucionais (CNNs)

Integrantes:

Marcus Gabriel de Almeida Nunes
Pedro Anacleto Martins Senna De Oliveira
Guilherme Schwinn Fagundes
Kalil Georges Balech
Julio Cesar Coelho de Amorim

Professor:

Prof. Paulo Marcelo Tasinaffo

São José dos Campos, 27 de setembro de 2023

Sumário

1 Introdução

No contexto atual da ciência da computação e do aprendizado de máquina, as Redes Neurais Convolucionais (CNNs) têm desempenhado um papel significativo na classificação e identificação de objetos em imagens. Este trabalho propõe um laboratório focalizado na utilização da bilioteca FastAPI para implementar e explorar o potencial das CNNs na diferenciação de dígitos manuscritos.

O trabalho se concentra na implementação de um modelo de Redes Neurais Convolucionais (CNN) para distinguir entre dígitos manuscritos 7 e 3. A tarefa de distinguir entre os dígitos 7 e 3 pode parecer simples à primeira vista, mas envolve desafios intrínsecos que exigem a construção cuidadosa e o ajuste fino dos modelos de aprendizado de máquina.

Ao longo deste estudo, os participantes terão a oportunidade de adquirir conhecimentos práticos e teóricos sobre o funcionamento das CNNs, seu treinamento e validação.

2 Código Desenvolvido

Nesta seção serão comentados em detalhe os trechos do código desenvolvido. Assim, para cada bloco de código há uma explicação abaixo. Os resultados serão mostrados na seção 3.

```
from fastai import *
from fastai.vision.all import *
from fastai.imports import *
from fastai.data.external import untar_data, URLs
from PIL import ImageFile
```

As importações realizadas aqui incluem módulos e funções necessárias para o processamento de imagens e para trabalhar com a biblioteca Fastai. Fastai é uma estrutura de aprendizado profundo construída em cima do PyTorch, simplificando tarefas comuns de aprendizado de máquina e visão computacional. PIL (Python Imaging Library) é uma biblioteca usada para manipulação de imagens em Python.

```
path = untar_data(URLs.MNIST_TINY)
items = get_image_files(path)
```

O conjunto de dados MNIST (Modified National Institute of Standards and Technology) é um conjunto popular para reconhecimento de dígitos manuscritos, contendo imagens em preto e branco de dígitos de 0 a 9. O código utiliza a função untar_data para baixar e extrair os dados do conjunto de dados MNIST_TINY (contém apenas as classes de dígitos 3 e 7) usando a URL fornecida pela constante URLs.MNIST_TINY. Em seguida, get_image_files é usado para obter uma lista de caminhos de arquivo para cada imagem presente nesse conjunto de dados.

```
train_path = path/'train'
test_path = path/'test'
```

Depois de ter descompactado os dados, esses caminhos são definidos para os conjuntos de treinamento e teste dentro do conjunto de dados MNIST_TINY. Em problemas de aprendizado de máquina, normalmente dividimos os dados em conjuntos de treinamento e teste. O conjunto de treinamento é usado para treinar o modelo, enquanto o conjunto de teste é usado para avaliar o desempenho do modelo treinado.

```
PILImageBW.create(items[-1])
```

O trecho acima exibe um exemplo de item do conjunto de dados baixado.

```
data_def = DataBlock(
  blocks=(ImageBlock, CategoryBlock),
  get_items=get_image_files,
  get_y=parent_label,
  splitter=RandomSplitter(valid_pct=0.2, seed=42),
  item_tfms=Resize(
    224,
    method=ResizeMethod.Squish,
    pad_mode=PadMode.Zeros
  )
)
```

O trecho de código acima está relacionado à definição de um bloco de dados (DataBlock) usando a biblioteca Fastai para manipulação e preparação de dados:

1. **DataBlock:** é uma ferramenta fundamental no Fastai para definir como os dados devem ser carregados, processados e apresentados ao modelo durante o treinamento. Ele organiza o pipeline de processamento de dados.

2. **blocks=(ImageBlock, CategoryBlock):** É especificado que temos dois tipos de dados: imagens (ImageBlock) e categorias (CategoryBlock). No contexto de reconhecimento de dígitos manuscritos, as imagens representam os dígitos e as categorias são os rótulos (no caso, os número 3 e 7).

3. **get_items=get_image_files:** Define a função que será usada para obter os itens (no caso, as imagens). A função get_image_files foi usada anteriormente para coletar os caminhos dos arquivos de imagem.

4. **get_y=parent_label:** Indica como os rótulos serão extraídos dos dados. parent_label é uma função que, dado um caminho de arquivo, extrai a categoria ou rótulo do diretório pai (no caso das imagens MNIST, o diretório pai contém informações sobre o número).

5. **splitter:** Define como os dados serão divididos em conjuntos de treinamento e validação. RandomSplitter divide aleatoriamente os dados, reservando 20% para validação (valid_pct=0.2). O seed=42 é usado para garantir que a divisão seja reproduzível, ou seja, os mesmos resultados serão obtidos sempre que o código for executado.

6. **item_tfms:** Especifica as transformações que serão aplicadas às imagens. Neste caso, as imagens serão redimensionadas para terem tamanho 224x224 pixels. ResizeMethod.Squish é o método usado para redimensionar a imagem (pode distorcer a imagem para ajustá-la ao tamanho desejado), e PadMode.Zeros é o modo de preenchimento que adiciona zeros para manter o aspecto da imagem após o redimensionamento.

Esse bloco de dados definido pelo DataBlock será usado mais tarde para criar um objeto DataLoaders que contém os dados preparados para treinar um modelo de reconhecimento de dígitos manuscritos.

```
dls = data_def.dataloaders(train_path, bs=20, shuffle=True)
dls.show_batch()
```

Este trecho de código usa o bloco de dados (DataBlock) anteriormente definido para criar um objeto DataLoaders e depois exibir um lote de dados para inspeção:

1. data_def é o bloco de dados previamente definido usando DataBlock. Aqui, estamos usando o método dataloaders desse bloco de dados para criar um objeto DataLoaders.

2. train_path é o caminho para o conjunto de dados de treinamento, que será usado para carregar os dados.

3. bs=20 define o tamanho do lote (batch size) como 20. O batch size é o número de amostras de dados que serão usadas em uma iteração durante o treinamento do modelo.

4. shuffle=True indica que os dados serão embaralhados aleatoriamente durante o treinamento. Embaralhar os dados pode ser útil para evitar que o modelo aprenda a depender da ordem dos dados.

Por fim, show_batch() é um método que permite visualizar um lote de dados carregados nos DataLoaders. Ele exibe uma grade de imagens do lote de dados para permitir uma inspeção visual rápida, mostrando como são as imagens e seus rótulos correspondentes.

```
data_def.summary(path)
```

```
print('Available classes: ', dls.vocab)
print('Data example: ', dls.show_batch(max_n=6, nrows=2))
```

Os dois trechos acima, mostra algumas informações sobre as classes escolhidas, como por exemplo quais são elas (classes 3 e 7).

```
learner = vision_learner(dls, xresnet18, pretrained=False, metrics=accuracy)
```

Esse trecho de código cria um objeto learner para treinar um modelo de rede neural utilizando Fastai:

1. vision_learner é uma função do Fastai que cria um objeto Learner específico para problemas de visão computacional.
2. dls é o objeto DataLoaders que contém os dados preparados para treinamento e validação.
3. xresnet18 é uma arquitetura de rede neural específica que será usada como base para o modelo. Neste caso, é uma versão modificada da arquitetura ResNet18 (uma rede neural convolucional), otimizada para treinamento mais rápido e eficiente.
4. pretrained=False indica que não se está usando um modelo pré-treinado. Modelos pré-treinados geralmente são treinados em grandes conjuntos de dados e depois ajustados para tarefas específicas.
5. metrics=accuracy define a métrica que será usada para avaliar o desempenho do modelo durante o treinamento. Neste caso, a métrica accuracy (acurácia) será usada para calcular a precisão do modelo em prever os rótulos corretos.

Basicamente, esse trecho de código está configurando um ambiente para treinar um modelo de rede neural baseado na arquitetura XResNet18, do zero (sem pesos pré-treinados), utilizando os dados de treinamento e validação definidos anteriormente nos DataLoaders. O desempenho do modelo será avaliado usando a métrica de acurácia durante o treinamento.

```
lr_min, lr_steep, lr_slide, lr_valley = learner.lr_find(suggest_funcs =
(minimum, steep, slide, valley))

learner.lr_find()
```

Esse trecho de código utiliza o método lr_find() do objeto learner para encontrar a taxa de aprendizado ideal para o treinamento do modelo. Estão sendo usadas diferentes funções (minimum, steep, slide, valley) para sugerir diferentes tipos de taxas de aprendizado ideais. Cada função sugere um tipo específico de taxa de aprendizado com base no comportamento da função de perda durante o treinamento. Nesse caso estamos buscando o vale do gráfico de perda no aprendizado.

O método lr_find() é uma ferramenta útil para encontrar a taxa de aprendizado ideal para o treinamento de redes neurais. Ele testa diferentes taxas de aprendizado automaticamente e exibe um gráfico da variação da função de perda em relação a essas taxas, permitindo identificar a faixa em que a perda diminui rapidamente. As funções de sugestão ajudam a determinar taxas de aprendizado específicas associadas a diferentes comportamentos da função de perda durante o treinamento.

```
learner.fine_tune(4, lr_valley, cbs=[ShowGraphCallback(),
SaveModelCallback(monitor='valid_loss')])
```

Esse trecho de código realiza o ajuste fino (fine-tuning) de um modelo usando o método fine_tune() do objeto learner.

1. fine_tune() é um método do Fastai usado para realizar o ajuste fino de um modelo pré-treinado ou treinado a partir do zero.
2. 4 representa o número de épocas (epochs) para o treinamento adicional. O ajuste fino consiste em treinar o modelo em um conjunto de dados específico (geralmente menor) por algumas épocas para adaptar os pesos do modelo e melhorar o desempenho em uma tarefa específica.
3. lr_valley é a taxa de aprendizado escolhida para o ajuste fino. Presumivelmente, essa taxa de aprendizado foi determinada como ideal usando o método lr_find() ou por alguma outra estratégia de busca.
4. cbs=[ShowGraphCallback(), SaveModelCallback(monitor='valid_loss')] especifica os callbacks (retrochamadas) que serão utilizados durante o treinamento. Callbacks são funções que são chamadas em pontos específicos durante o treinamento para executar ações adicionais.
5. ShowGraphCallback() exibe um gráfico interativo mostrando o progresso do treinamento (métricas, perdas etc.).
6. SaveModelCallback(monitor='valid_loss') é um callback que monitora a perda no conjunto de validação (valid_loss) e salva o modelo sempre que a perda no conjunto de validação for mínima, ajudando a preservar o melhor modelo alcançado durante o treinamento.

Essencialmente, esse código executa o ajuste fino do modelo por 4 épocas, usando uma taxa de aprendizado específica (lr_valley) e registrando informações sobre o treinamento usando os callbacks ShowGraphCallback e SaveModelCallback. O ajuste fino é uma etapa comum para melhorar o desempenho de modelos pré-treinados ou treinados a partir do zero em tarefas específicas.

```
learner.save('learner_1')
```

O trecho acima salva o modelo de aprendizado encontrado para que posteriormente possa ser reutilizado.

```
learner.fine_tune(4, lr_valley, cbs=[ShowGraphCallback(),
SaveModelCallback(monitor='valid_loss')])
```

Esse trecho de código realiza o ajuste fino (fine-tuning) de um modelo usando o método fine_tune() do objeto learner.

1. fine_tune() é um método do Fastai usado para realizar o ajuste fino de um modelo pré-treinado ou treinado a partir do zero.

2. 4 representa o número de épocas (epochs) para o treinamento adicional. O ajuste fino consiste em treinar o modelo em um conjunto de dados específico (geralmente menor) por algumas épocas para adaptar os pesos do modelo e melhorar o desempenho em uma tarefa específica.

3. lr_valley é a taxa de aprendizado escolhida para o ajuste fino. Presumivelmente, essa taxa de aprendizado foi determinada como ideal usando o método lr_find() ou por alguma outra estratégia de busca.

4. cbs=[ShowGraphCallback(), SaveModelCallback(monitor='valid_loss')] especifica os callbacks (retrochamadas) que serão utilizados durante o treinamento. Callbacks são funções que são chamadas em pontos específicos durante o treinamento para executar ações adicionais.

5. ShowGraphCallback() exibe um gráfico interativo mostrando o progresso do treinamento (métricas, perdas etc.).

6. SaveModelCallback(monitor='valid_loss') é um callback que monitora a perda no conjunto de validação (valid_loss) e salva o modelo sempre que a perda no conjunto de validação for mínima, ajudando a preservar o melhor modelo alcançado durante o treinamento.

Essencialmente, esse código executa o ajuste fino do modelo por 4 épocas, usando uma taxa de aprendizado específica (lr_valley) e registrando informações sobre o treinamento usando os callbacks ShowGraphCallback e SaveModelCallback. O ajuste fino é uma etapa comum para melhorar o desempenho de modelos pré-treinados ou treinados a partir do zero em tarefas específicas.

```
learner.show_results()
```

O trecho acima mostra alguns resultados do treinamento.

```
interp = ClassificationInterpretation.from_learner(learner)
interp.plot_confusion_matrix()
```

O trecho acima monta e exxibe a matriz de confusão do treinamento da rede neural.

```
interp.plot_top_losses(6, nrows=2)
```

O código usa o objeto interp (interpretation object) para plotar as amostras que resultaram nas maiores perdas durante a validação ou teste do modelo.

```
warnings.filterwarnings("ignore")
from PIL import Image
import os

images = [file for file in os.listdir(test_path) if file.endswith('.png')]

for file in images:
    img_path = test_path/file
    img = Image.open(img_path)

    prediction = learner.predict(img_path)[0]

    print(f'File: {file} - Prediction: {prediction}')
    display(img)
```

O trecho acima realiza a automatização das predições de todas as imagens contidas no diretório de teste da biblioteca. A biblioteca os é responsável por listar todos os arquivos com a extensão .png presentes no diretório de teste, armazenando-os na lista images, para que posteriormente sejam realizadas as predições com o modelo.

Assim, o loop for percorre cada arquivo de imagem na lista images. Para cada imagem, o código abre o arquivo utilizando a biblioteca PIL, faz a previsão com o modelo learner, exibe o nome do arquivo e a previsão feita pelo modelo, e mostra a imagem correspondente usando a função display().

3 Resultados

3.1 Classes de Treino

Figura 1: Exemplos de figuras das duas classes de treino

3.2 Taxa de Aprendizado Ideal para o Modelo

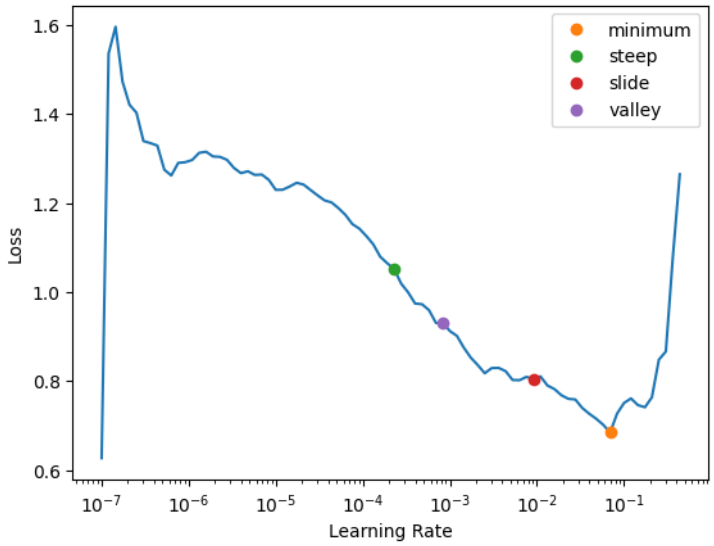

Figura 2: Taxa de aprendizado ideal encontrada pelo learner para a taxa de aprendizado ideal do modelo

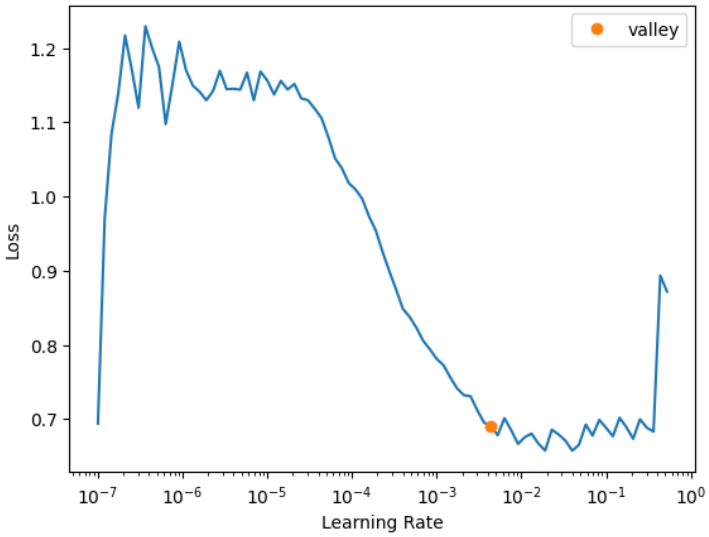

Figura 3: Vale encontrado pelo learner para a taxa de aprendizado ideal do modelo

3.3 Ajuste Fino do Modelo

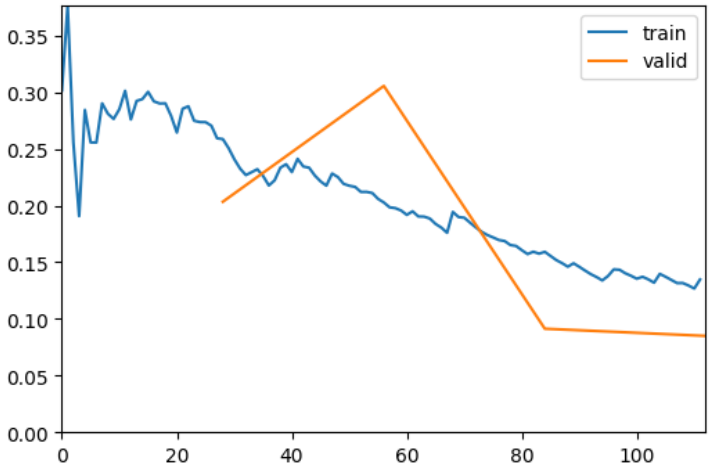

Figura 4: Resultado do ajuste fino - monitoramento das perdas na validação e no treinamento de ajuste fino

3.4 Treinamento da Rede Neural

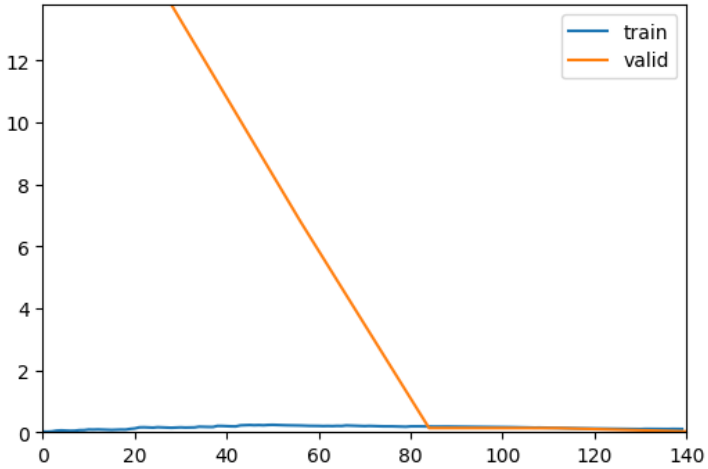

Figura 5: Função de perda durante o treinamento da rede neural com 5 épocas

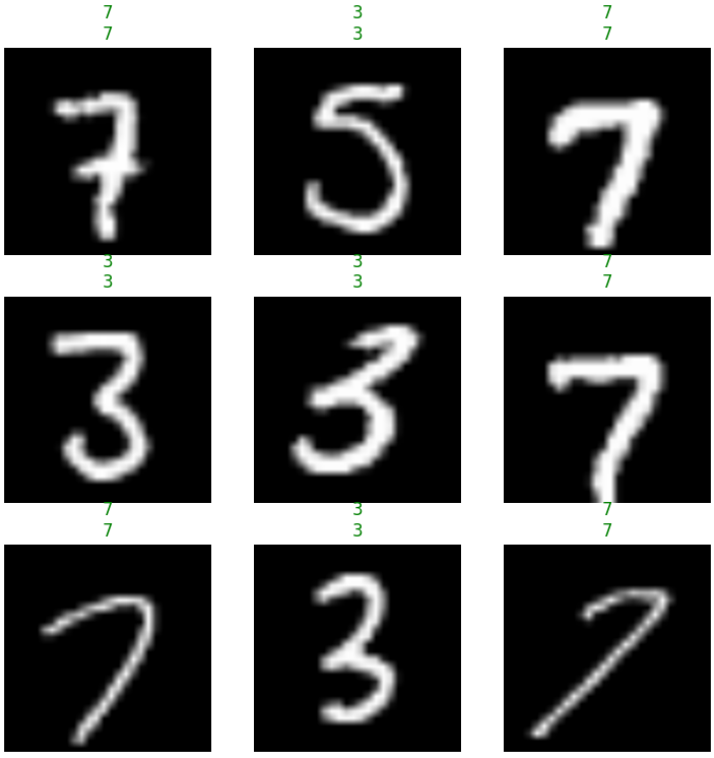

Figura 6: Alguns resultados do treinamento da rede neural

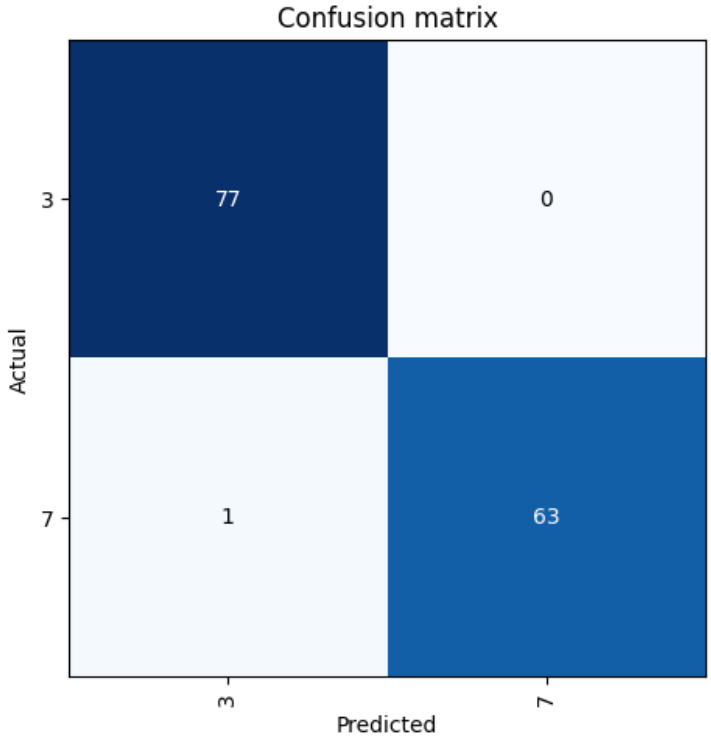

Figura 7: Matriz de confusão do treinamento

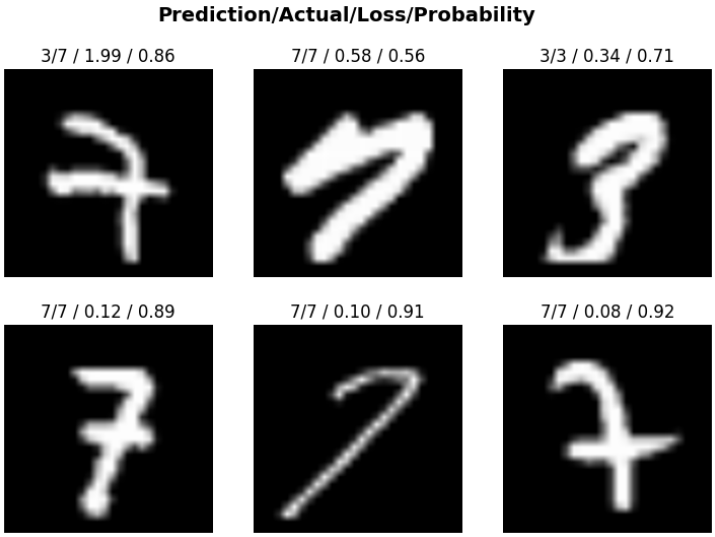

Figura 8: Principais perdas durante o treinamento

3.5 Resultados das Previsões com o Modelo

Nas figuras abaixo encontram-se as previsões das imagens do diretório de teste. As previsões foram obtidas utilizando o modelo construído. Na parte de cima foi colocado um título com o nome do arquivo que contém a imagem junto com o valor previsto pelo modelo e, a baixo, tem-se uma miniatura dessa imagem, para fins práticos de identificação e avaliação do modelo.

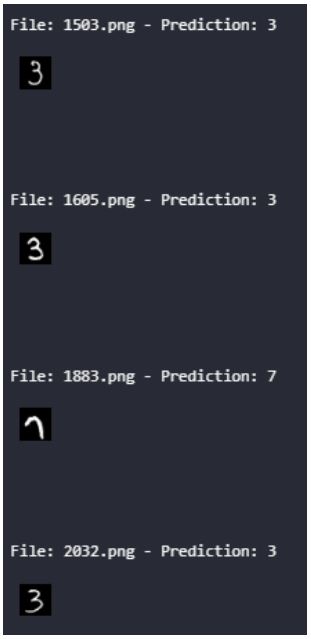

Figura 9: Previsão das figuras contidas na pasta de teste da biblioteca

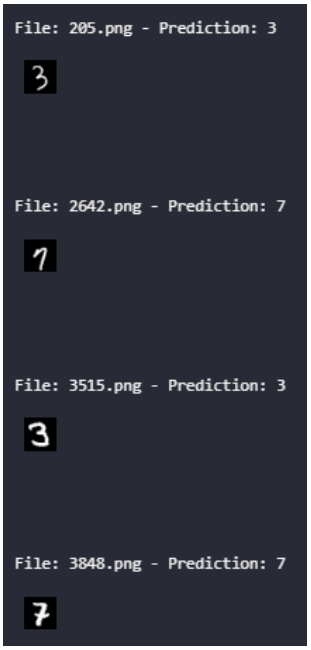

Figura 10: Previsão das figuras contidas na pasta de teste da biblioteca

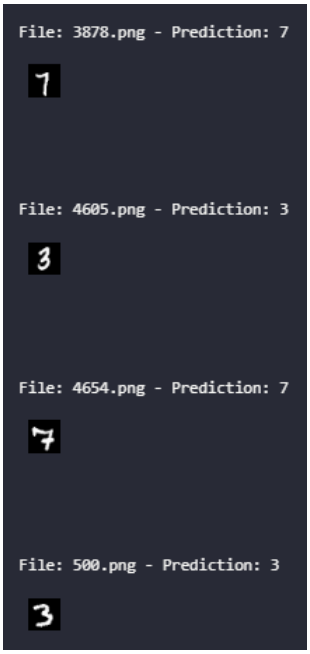

Figura 11: Previsão das figuras contidas na pasta de teste da biblioteca

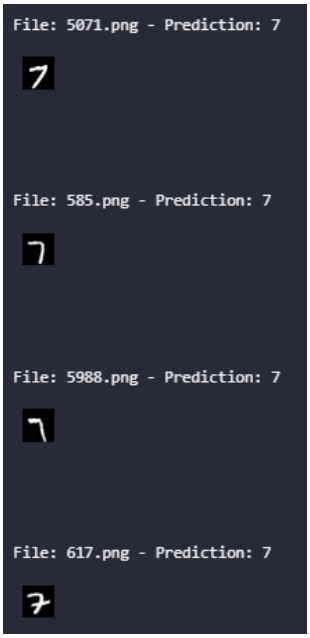

Figura 12: Previsão das figuras contidas na pasta de teste da biblioteca

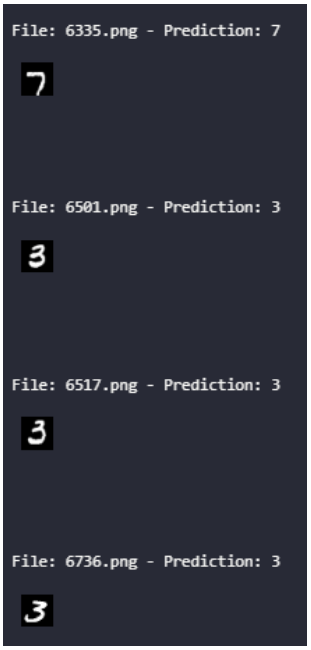

Figura 13: Previsão das figuras contidas na pasta de teste da biblioteca

4 Conclusão

O modelo demonstrou um desempenho notável ao realizar previsões, evidenciado pela análise da matriz de confusão do treinamento que revelou apenas uma previsão errada dentre 141 imagens. Além disso, a baixa perda do modelo destaca a eficácia de suas previsões. Ao analisar as previsões feitas para o conjunto de testes, foi possível constatar que todas as previsões estão em concordância com os dígitos manuscritos presentes nas imagens. Esses resultados indicam uma precisão satisfatória do modelo no reconhecimento de dígitos manuscritos.