Universidade do Minho
Escola de Engenharia

Visão por Computador e Processamento de Imagem
Grupo 20
2023/2024

Tomás Cardoso Francisco pg54263
Ana Filipa da Cunha Rebelo pg53624

Maio 2024

1 Introdução

O presente relatório descreve o processo de desenvolvimento e avaliação de modelos de Deep Learning aplicados ao dataset GTSRB , com o objetivo de alcançar a melhor precisão possível no conjunto de teste. O dataset GTSRB consiste em imagens de sinais de trânsito, onde cada imagem é classificada em uma das classes correspondentes aos diferentes tipos de sinais.

O trabalho foi dividido em três fases distintas. Na primeira fase, procedemos à divisão dos conjuntos de dados fornecidos em conjuntos de treino, validação e teste. De seguida, focamos no treino dos modelos atráves da Data Augmentation onde exploramos uma variedade de filtros e métodos de processamento de imagem para entender o seu impacto no desempenho final da rede. Por fim, estudamos o potencial de utilizar ensembles de redes neuronais, explorando a sinergia entre eles para melhorar ainda mais a precisão na classificação dos sinais de trânsito. Esta abordagem visa explorar a diversidade de aprendizagem entre os modelos individuais para obter uma classificação mais robusta e confiável.

Ao longo deste relatório iremos fazer uma descrição detalhada das decisões tomadas em cada etapa do processo, bem como uma análise crítica dos testes realizados.

2 Preparação dos dados

A preparação cuidadosa dos dados é essencial para garantir a qualidade e a confiabilidade dos resultados em projetos de análise de dados e machine learning. Uma preparação adequada dos dados também contribui para otimizar a eficiência computacional e facilitar a interpretação dos resultados, essencial para tomadas de decisão informadas.

Para preparar os dados foram feitas as seguintes tarefas:

- Extração de ficheiros Zip: Inicialmente, os conjuntos de dados de treino e teste estavam armazenados em ficheiros ZIP. Para ser possível aceder e utilizar esses dados, implementamos um método dedicada à extração dos arquivos PPM contidos nos arquivos ZIP, permitindo uma organização eficiente dos dados em diretórios específicos, prontos para serem processados.

```python
def extract_ppm_files(zipfile_path, extract_dir):
    os.makedirs(extract_dir, exist_ok=True)

    with zipfile.ZipFile(zipfile_path, "r") as zip_ref:
        ppm_files = [
            name for name in zip_ref.namelist() if name.lower().endswith(".ppm")
        ]
        for ppm_file in ppm_files:
            zip_ref.extract(ppm_file, path=extract_dir)


test_zipfile_path = "test_images.zip"
train_zipfile_path = "train_images.zip"

test_extract_dir = "test_images"
train_extract_dir = "train_images"

extract_ppm_files(test_zipfile_path, test_extract_dir)
extract_ppm_files(train_zipfile_path, train_extract_dir)
```

Figure 1: Extração de ficheiros Zip

- Conversão do Formato de Imagem: Efetuamos a conversão das imagens para o formato PNG para permitir o uso das técnicas de processamento de imagem e manipulação de dados lecionadas na disciplina.

```python
def convert_ppm_to_png(ppm_file):
    img = Image.open(ppm_file)
    png_file = os.path.splitext(ppm_file)[0] + ".png"
    img.save(png_file, "PNG")


def convert_and_cleanup_folder(input_dir):
    for root, _, files in os.walk(input_dir):
        for file in files:
            if file.lower().endswith(".ppm"):
                ppm_file = os.path.join(root, file)
                convert_ppm_to_png(ppm_file)
                os.remove(ppm_file)
            elif file.lower().endswith(".png"):
                os.path.join(root, file)


test_dir = "test_images"
train_dir = "train_images/GTSRB/Final_Training/Images"

convert_and_cleanup_folder(test_dir)
convert_and_cleanup_folder(train_dir)
```

Figure 2: Converter ppm para png

- Definição de parâmetros de treino: O tamanho da batch refere-se ao número de amostras que são utilizadas na rede em apenas uma iteração, tendo sido definido como 32. O tamanho da imagem no dataset fuicou definido como 32, dado o tamanho das imagens nos ficheiros. O número de epochs refere-se ao número de vezes que o modelo passará por todo o conjunto de dados durante o treino, tendo sido definido como 20.

```python
BATCH_SIZE = 32
IMAGE_SIZE = 32
EPOCHS = 20
```

Figure 3: Definição de parâmetros

- Transformação de imagem: Redimensiona as imagens para o tamanho definido e converte as em tensores

```python
image_reshape = transforms.Compose(
    [transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)), transforms.ToTensor()]
)
```

Figure 4: Transformação de imagem

- Carregamento dos conjuntos de dados: Para efetuar o carregamento dos dados utilizamos a classe ImageFolder da biblioteca torchvision para que os dados fossem carregados de forma organizada.

```
train_dataset = torchvision.datasets.ImageFolder(
    root=train_dir, transform=image_reshape
)
train_iterator = torch.utils.data.DataLoader(
    train_dataset, batch_size=BATCH_SIZE, shuffle=True
)

test_dataset = torchvision.datasets.ImageFolder(root=test_dir, transform=image_reshape)
test_iterator = torch.utils.data.DataLoader(test_dataset, batch_size=BATCH_SIZE)
```

Figure 5: Carregamento de dados

- Carregamento de um lote de dados: Para permitir inspecionar os dados e verificar sua forma.

```
images, labels = next(iter(train_iterator))
print("Batch shape: ", images.shape)

print(labels)
```

Figure 6: Carregamento de um lote de dados

- Exibição das imagens: Para verificar se os dados foram carregados corretamente e entender melhor o conteúdo dos conjuntos de dados.

```
vcpi_util.show_images(4, 8, images, labels, train_dataset.classes)
```

Figure 7: Exibição das imagens

3 Modelo Base e Ferramentas

3.1 Modelo

O modelo base adotado é uma rede neuronal convolucional (CNN), uma arquitetura amplamente utilizada para processamento de imagens e reconhecimento visual, composta por várias camadas convolucionais e de pooling, seguidas por camadas totalmente conectadas para a classificação final das imagens.

Principais componentes do modelo:

- Camadas Convolucionais(Conv2D): Responsáveis por extrair características das imagens de entrada. No nosso modelo existem quatro camadas convolucionais.
- Camadas de Batch Normalization (BatchNorm2d): Responsáveis pela normalização por lotes.
- Camadas de ReLU (ReLU): Responsáveis por criar uma função de ativação ReLU.
- Camadas de Pooling(MaxPool2d): Estas camadas servem para reduzir as dimensões espaciais dos mapas de características, mantendo as características mais importantes.

- Camadas Totalmente Conectadas: No final da rede, há uma ou mais camadas totalmente conectadas que realizam a classificação final com base nas características extraídas pelas camadas convolucionais.
- Funções de Ativação: Entre as camadas convolucionais, são aplicadas funções de ativação ReLU para introduzir não linearidades na rede.

```python
class Model(torch.nn.Module):
    def __init__(self, num_classes):
        super().__init__()
        self.conv1 = torch.nn.Conv2d(3, 16, 3)
        self.bn1 = torch.nn.BatchNorm2d(16)
        self.relu1 = torch.nn.ReLU()

        self.conv2 = torch.nn.Conv2d(16, 32, 3)
        self.bn2 = torch.nn.BatchNorm2d(32)
        self.relu2 = torch.nn.ReLU()

        self.maxpool1 = torch.nn.MaxPool2d(2)

        self.conv3 = torch.nn.Conv2d(32, 48, 3)
        self.bn3 = torch.nn.BatchNorm2d(48)
        self.relu3 = torch.nn.ReLU()

        self.conv4 = torch.nn.Conv2d(48, 48, 3)
        self.bn4 = torch.nn.BatchNorm2d(48)
        self.relu4 = torch.nn.ReLU()

        self.maxpool2 = torch.nn.MaxPool2d(2)

        self.fc1 = torch.nn.Linear(1200, num_classes)
```

Figure 8: Modelo Base

3.2 Ferramentas Principais

A implementação e o treino do modelo foram realizados usando principalmente o PyTorch, uma poderosa biblioteca de aprendizagem de máquina e deep learning. Principais ferramentas utilizadas:

- PyTorch: fornece uma estrutura flexível e eficiente para a construção de modelos de deep learning.
- torchvision: Um pacote em PyTorch que fornece conjuntos de dados e modelos populares para tarefas de visão computacional, como classificação de imagens, detecção de objetos, etc.
- torch.utils.data.DataLoader: Utilizado para criar iteradores eficientes para percorrer os conjuntos de dados de treino, validação e teste em lotes.
- torch.optim: Utilizado para otimizar os pesos do modelo durante o treino.
- torch.nn.CrossEntropyLoss: Uma função de perda comumente usada para problemas de classificação multi-classe.
- PIL:usada para processamento de imagem, como carregar e guardar imagens.

4 Data Augmentation

O Data Augmentation é uma técnica fundamental para melhorar o desempenho de modelos de Deep Learning, especialmente quando os conjuntos de dados são limitados. Foram utilizadas diferentes estratégias de aumento de dados, variando em níveis de complexidade, para enriquecer o conjunto de dados de treino e tornar o modelo mais robusto e generalizável. Certamente, vamos analisar cada nível de aumento de dados:

- Nível 1 - Básico: Neste nível priorizamos a consistência no formato das imagens de entrada para o modelo. Para tal, utilizamos a técnica Resize,garantindo que todas as imagens tenham as mesmas dimensões,e o Gaussian Blur dado que suaviza as transições entre os pixels na imagem, reduzindo o ruído e as irregularidades.

- Nível 2 - Moderado: Neste nível o objetivo foi aumentar a diversidade dos dados introduzindo transformações mais variadas.Usamos o Random Horizontal Flip,que cria versões espelhadas das imagens ,o Random Rotation para introduzir variação na orientação das imagens e o Color Jitter para adicionar variações nas cores das imagens. Essas transformações são mais variadas e introduzem mudanças significativas na aparência das imagens, o que pode ajudar o modelo a aprender a extrair características robustas e invariáveis em relação a essas mudanças.

- Nível 3 - Avançado: Neste nivel adotamos técnicas mais complexas para aumentar ainda mais a robustez do modelo. Usamos o Random Affine que inclui rotação adicional, translação, escala e cizalhamento e o Random Erasing para tornar o modelo mais robusto e preciso nas regiões ausentes de informação.

```python
basic_augmentation = v2.Compose(
    [
        v2.ToImage(),
        v2.Resize((IMAGE_SIZE, IMAGE_SIZE)),
        v2.GaussianBlur(kernel_size=3, sigma=1.0),
        v2.ToDtype(torch.float32, scale=True),
    ]
)

moderate_augmentation = v2.Compose(
    [
        v2.ToImage(),
        v2.Resize((IMAGE_SIZE, IMAGE_SIZE)),
        v2.RandomHorizontalFlip(p=0.5),
        v2.RandomRotation(degrees=10),
        v2.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
        v2.GaussianBlur(kernel_size=3, sigma=(0.1, 1.5)),
        v2.ToDtype(torch.float32, scale=True),
    ]
)

advanced_augmentation = v2.Compose(
    [
        v2.ToImage(),
        v2.Resize((IMAGE_SIZE, IMAGE_SIZE)),
        v2.RandomHorizontalFlip(p=0.5),
        v2.RandomRotation(degrees=15),
        v2.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.15),
        v2.GaussianBlur(kernel_size=3, sigma=(0.5, 1.5)),
        v2.RandomAffine(degrees=15, translate=(0.1, 0.1), scale=(0.9, 1.1), shear=5),
        v2.RandomErasing(p=0.3, scale=(0.02, 0.1), ratio=(0.3, 3.3)),
        v2.ToDtype(torch.float32, scale=True),
    ]
)
```

Figure 9: Composições de Augmentations usadas

5 Ensembles

O ensemble é um modelo em aprendizagem de máquina onde múltiplos modelos individuais são combinados para formar um modelo mais robusto e preciso do que qualquer um dos modelos individuais. A ideia por trás dos ensembles é que, ao combinar as previsões de vários modelos, podemos produzir resultados mais confiáveis e robustos do que um único modelo.

Foram utilizados os seguintes métodos para representar o ensemble:

- get_labels_logits_and_preds - recebe uma lista de modelos treinados como entrada, percorrendo um iterador de teste (test_iterator), que contém imagens e respetivos rótulos. Para cada modelo na lista de modelos, ela

calcula as saídas do modelo para as imagens de teste e armazena essas saídas numa lista de listas chamada logits. Além disso, armazena os rótulos verdadeiros das imagens de teste numa lista chamada labels. No final, retorna os rótulos verdadeiros e os logits de cada modelo.

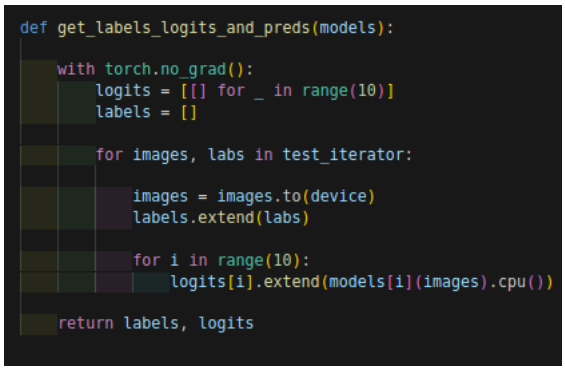

Figure 10: Função get_labels_logits_and_preds

- get_class_from_sum_of_logits - recebe os logits calculados para cada modelo, onde, para cada imagem de teste, ela soma os logits correspondentes a essa imagem de todos os modelos. Em seguida, determina a classe prevista para cada imagem, selecionando a classe com o maior valor de logit. Os resultados são armazenados numa lista chamada sum_logits, que contém as classes previstas para todas as imagens de teste.

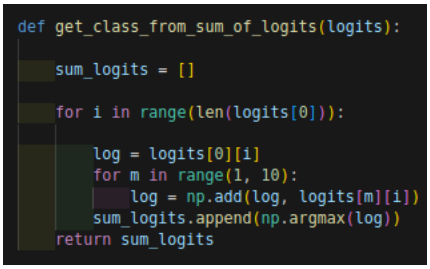

Figure 11: Função get_class_from_sum_of_logits

- get_stats - recebe os rótulos verdadeiros das imagens de teste (labels), as previsões de classe dos modelos (class_preds) e as previsões de classe obtidas pela soma dos logits dos modelos (class_logits). Ela calcula várias estatísticas de desempenho, incluindo o número total de imagens, o número de previsões corretas, o número de previsões incorretas, o número de previsões corretas pela votação da maioria, o número de empates na votação da maioria e o número de previsões incorretas pela votação da maioria. No final, retorna uma lista com esses valores estatísticos.

```python
def get_stats(labels, class_preds, class_logits):

    all_correct = 0
    all_incorrect = 0
    maj_vote = 0
    maj_wrong = 0
    tie = 0
    count = 0

    for k in range(len(labels)):

        counter = collections.Counter(class_preds[k])
        if len(counter) == 1:
            if counter.most_common(1)[0][0] == labels[k]:
                all_correct += 1
            else:
                all_incorrect += 1
        else:
            aux = counter.most_common(2)
            if aux[0][1] > aux[1][1] and aux[0][0] == labels[k]:
                maj_vote += 1
            if aux[0][1] > aux[1][1] and aux[0][0] != labels[k]:
                maj_wrong += 1
            elif aux[0][1] == aux[1][1]:
                tie += 1

        count += 1

    return [count, all_correct, all_incorrect, maj_vote, tie, maj_wrong]
```

Figure 12: Função get_stats

6 Resultados e Observações

6.1 Modelo Base

O desempenho do modelo mostrou uma tendência positiva à medida que o treino progredia, evidenciada pela redução consistente da perda e pelo aumento da precisão. No entanto, é crucial observar que, embora a precisão no conjunto de treino continuasse a melhorar, a precisão no conjunto de validação parecia estagnar após a segunda época. Essa estabilização sugere que o modelo pode estar começando a sofrer de overfitting prejudicando sua capacidade de generalização para novos dados. Em suma, enquanto o modelo se torna cada vez melhor em prever os dados de treino, a sua habilidade de se adaptar a novos dados pode estar se deteriorando, o que é um aspecto crucial a ser considerado durante o treino e ajuste do modelo.

```
Epoch: 000; Loss: 0.009644; Accuracy: 92.2785; Val Loss: 0.001445; Val Acc: 98.9033; Elapsed time: 337.8220
Epoch: 001; Loss: 0.000937; Accuracy: 99.2444; Val Loss: 0.001379; Val Acc: 98.7758; Elapsed time: 23.8758
Epoch: 002; Loss: 0.000544; Accuracy: 99.5027; Val Loss: 0.001073; Val Acc: 98.9799; Elapsed time: 26.2681
Epoch: 003; Loss: 0.000589; Accuracy: 99.4421; Val Loss: 0.000904; Val Acc: 99.3752; Elapsed time: 26.9760
Finished Training

0.9643705487251282
```

Figure 13: Resultados Obtidos-Modelo Base

6.2 Data Augmentation

Através dos resultados obtidos para os diferentes niveis de Data Augmentation podemos observar variações significativass no desempenho do modelo ao longo do treino.

No nível 1,embora tenha iniciado com uma accuracy e uma perda um pouco inferiores em comparação com o modelo original, apresentou uma rápida melhoria ao longo das épocas.

```
Epoch: 000; Loss: 0.011275; Accuracy: 90.8821; Val Loss: 0.002248; Val Acc: 98.2530; Elapsed time: 41.6265
Epoch: 001; Loss: 0.001196; Accuracy: 99.0372; Val Loss: 0.001319; Val Acc: 98.8651; Elapsed time: 42.2810
Epoch: 002; Loss: 0.000684; Accuracy: 99.4548; Val Loss: 0.001512; Val Acc: 98.6611; Elapsed time: 40.6393
Epoch: 003; Loss: 0.000602; Accuracy: 99.4006; Val Loss: 0.001358; Val Acc: 98.7503; Elapsed time: 43.0751
Finished Training

0.9477434754371643
```

Figure 14: Resultados Obtidos-Nivel 1

No nível 2, o desempenho inicial é significativamente pior em comparação ao modelo original, mas o modelo melhora gradualmente ao longo das épocas. No entanto, continua com um impacto negativo, mas com a capacidade de recuperar parcialmente.

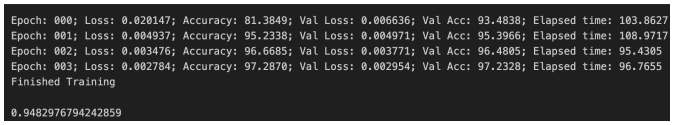

Figure 15: Resultados Obtidos-Nivel 2

Por fim, o nível 3 apresentou um desempenho inicial com uma perda alta e uma accuracy baixa, indicando dificuldades iniciais do modelo. Apesar de mostrar alguma melhoria ao longo das épocas, o desempenho permaneceu inferior ao modelo base, sugerindo que essa augmentação teve um impacto negativo mais substancial. É possível que a intensidade ou a natureza das transformações introduzidas não tenham sido adequadas para o conjunto de dados ou para o modelo em questão.

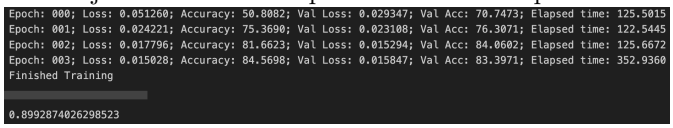

Figure 16: Resultados Obtidos-Nivel 3

6.3 Ensemble

Em termos de precisão no uso do emsemble, podemos verificar uma precisão elevada, com 97.75 das previsões sendo corretas, concluindo uma concordância entre os modelos.

Em termos de majority voting, 2193 das amostras dos modelos previram corretamente, demonstrando uma melhoria da precisão geral do emsemble, apesar das 202 amostras dos modelos que previram incorretamente e 79 amostras que resultaram em empate.

Em comparação com os modelos individuais, o emsemble supera cada um dos modelos em termos de precisão, sendo capaz de mitigar alguns erros individuais do modelo.

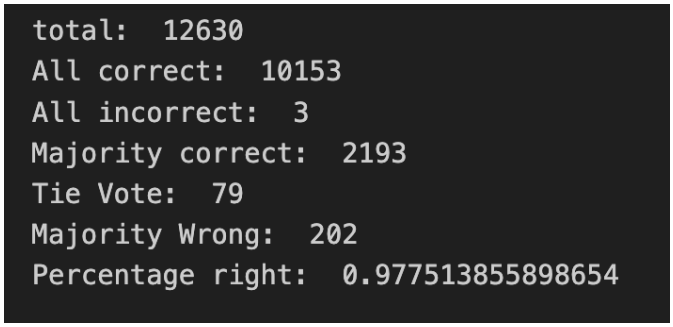

Figure 17: Resultados Obtidos-Ensemble