Universidade do Minho
Mestrado em Engenharia Informática

Visão por Computador e Processamento de Imagem

Deep Learning

Grupo 9
Inês Ferreira - PG53879
Marta Sá - PG54084

14 de junho de 2024

Conteúdo

1 Introdução

Este trabalho prático foi redigido no âmbito da unidade curricular de Visão por Computar e Processamento de Imagem e tem como principal objetivo explorar modelos de *Deep Learning* aplicados ao *dataset* alemão de sinais de trânsito GTSRB. O foco é obter o melhor resultado de *accuracy* possível no *dataset* de teste, sendo que o recorde publicado até à data é de 99.82%.

Assim, ao longo deste documento iremos apresentar, numa primeira fase, os diferentes modelos criados bem como as estratégias de *data augmentation* aplicadas tanto no pré-processamento como no processamento dinâmico. Numa segunda fase, serão explorados diferentes métodos de *ensemble* de redes com o objetivo de juntar os modelos obtidos anteriormente e obter resultados mais promissores. Por fim, será realizada uma discussão acerca do trabalho realizado e dos resultados obtidos.

2 Modelos

2.1 Modelo I

O primeiro modelo criado é constituído por uma rede neuronal convolucional (CNN) com quatro camadas convolucionais, cada uma seguida por um *batch normalization* e por uma função de ativação *ReLU*. Para além disso, existem duas camadas de *max pooling* para reduzir as dimensões espaciais. Após a última camada de *pooling*, os dados passam por uma *fully connected layer* responsável por produzir as previsões finais.

Desta forma, o modelo criado utiliza o algoritmo de otimização *Adam* para ajustar os seus parâmetros durante o treino e a função de perda *cross-entropy*, dado que é a mais adequada para problemas de classificação com várias classes.

```
class Conv(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)
```

Figura 1: Modelo I.

2.2 Modelo II

No que toca ao segundo modelo utilizado, as diferenças em relação ao modelo anterior são mínimas. De facto, este modelo difere apenas na utilização de mais mais nodos em cada camada, como podemos verificar na Figura 2:

```python
class Conv(torch.nn.Module):
    def __init__(self, num_classes):
        super().__init__()

        self.conv1 = torch.nn.Conv2d(3, 128, 3)
        self.bn1 = torch.nn.BatchNorm2d(128)
        self.relu1 = torch.nn.ReLU()

        self.conv2 = torch.nn.Conv2d(128, 256, 3)
        self.bn2 = torch.nn.BatchNorm2d(256)
        self.relu2 = torch.nn.ReLU()

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

        self.conv3 = torch.nn.Conv2d(256, 512, 3)
        self.bn3 = torch.nn.BatchNorm2d(512)
        self.relu3 = torch.nn.ReLU()

        self.conv4 = torch.nn.Conv2d(512, 512, 3)
        self.bn4 = torch.nn.BatchNorm2d(512)
        self.relu4 = torch.nn.ReLU()

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

        self.fc1 = torch.nn.Linear(12800, num_classes)
```

Figura 2: Modelo II.

2.3 Modelo III

O último modelo desenvolvido difere dos restantes no sentido em que é alterado o número de camadas convolucionais, passando a haver cinco. Para além disso, com o objetivo de evitar o *overfitting*, são adicionadas camadas de *dropout*.

A implementação do modelo 3 está, portanto, definida na seguinte figura:

```python
class Conv(torch.nn.Module):
    def __init__(self, num_classes):
        super().__init__()

        self.conv1 = torch.nn.Conv2d(3, 128, 3)
        self.bn1 = torch.nn.BatchNorm2d(128)
        self.relu1 = torch.nn.ReLU()
        self.drop1 = torch.nn.Dropout2d(0.3)

        self.conv2 = torch.nn.Conv2d(128, 256, 3)
        self.bn2 = torch.nn.BatchNorm2d(256)
        self.relu2 = torch.nn.ReLU()
        self.drop2 = torch.nn.Dropout2d(0.3)

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

        self.conv3 = torch.nn.Conv2d(256, 512, 3)
        self.bn3 = torch.nn.BatchNorm2d(512)
        self.relu3 = torch.nn.ReLU()
        self.drop3 = torch.nn.Dropout2d(0.3)

        self.conv4 = torch.nn.Conv2d(512, 512, 3)
        self.bn4 = torch.nn.BatchNorm2d(512)
        self.relu4 = torch.nn.ReLU()
        self.drop4 = torch.nn.Dropout2d(0.3)

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

        self.fc1 = torch.nn.Linear(12800, num_classes)
```

Figura 3: Modelo III.

2.4 Comparação Entre Modelos

Na Tabela 1 estão apresentadas os valores de *accuracy* de cada modelo treinado sem qualquer tipo de transformação.

De acordo com os resultados obtidos, podemos concluir que o terceiro modelo desenvolvido foi o que teve melhor desempenho. Para além disso, dado que o modelo I é o mais simples e rápido de executar, os testes foram realizados nele e, numa fase posterior, são utilizados os restantes modelos.

Modelo	Modelo I	Modelo II	Modelo II
Accuracy	96,5601	97.5693	97.9256

Tabela 1: *Accuracy* de cada modelo.

3 Pré-processamento

3.1 Resize

Na primeira fase do pré-processamento aplicamos a transformação *Resize* a todas as imagens do *dataset* de treino e de teste para um tamanho 32x32 pixeis de forma a ser possível treinar os modelo.

3.2 Contraste

Na segunda fase do pré-processamento, com o objetivo de uniformizar e realçar os detalhes de cada imagem, foi aplicada a transformação *RandomAutocontrast* responsável por ajustar automaticamente os pixeis de uma imagem.

De acordo com a Tabela 2 podemos verificar que os modelos tiveram um aumento pouco significativo.

Modelo	Modelo I	Modelo II	Modelo II
Sem Contraste	96,5601	97.5693	97.9256
Com Contraste	96,8013	97,601	98,1156

Tabela 2: *Accuracy* de cada modelo após o pré-processamento.

É importante referir que, para além das transformações mencionadas, testamos também outras transformações, tais como, *RandomEqualize* e *RandomHorizontalFlip* que acabaram por ser removidas por não apresentarem melhorias nos resultados.

4 Data Augmentation

Com o objetivo de aumentar a diversidade e a robustez do conjunto de dados, aplicamos um processo de *data augmentation*, tanto de forma dinâmica como estática. Uma vez que este processo é particularmente desafiador devido ao grande tamanho do conjunto de dados e aos limitados recursos computacionais, os testes foram organizados de acordo com a categoria de cada transformação.

Desta forma, explicamos ao longo do capítulo as diferentes transformações aplicadas às imagens.

4.1 Color Transformations

O primeiro conjunto de transformações aplicadas diz respeito a transformações a nível da cor da imagem a partir da técnica *color jittering*. Esta técnica permite alterar os valor de *brightness*, *hue* e *saturation* de um imagem. Assim, para além do *contrast* aplicado na fase de pré-processamento, as três transformações são aplicadas num novo *dataset* que é posteriormente concatenado ao original a partir da função *ConcatDataset* do *package torch*.

4.1.1 Brightness

Para a variação do brilho de uma imagem atribuímos à função *ColorJitter* o valor 0.5 que resulta no intervalo [0.5,1.5], do qual é escolhido uniformemente um valor. As transformações das restantes componentes, isto é, da saturação e da *hue* não mostraram melhorias no valor da *accuracy*, pelo que foram removidas.

Assim, podemos observar os resultados obtidos na Tabela 3. As diferenças nos valores de *accuracy* foram mais significativos no modelo I do que nos restantes modelos.

Modelo	Modelo I	Modelo II	Modelo II
Sem *Color Transf.*	96,8013	97,601	98,1156
Com *Color Transf.*	98,0127	98,0364	98.3135

Tabela 3: *Accuracy* de cada modelo após aplicar *color transformations*.

4.2 Geometric Transformations

Sinais de trânsito no mundo real podem não ter uma forma bem definida devido, por exemplo, a atos de vandalismo, condições atmosféricas adversas ou até mesmo ao ângulo em que a imagem foi capturada. De facto, estas imperfeições dos sinais podem ser verificadas no nosso *dataset* e, de forma a lidar com estes casos para que o modelo consiga reconhecê-los, aplicamos algumas transformações geométricas. Algumas dessas transformações são realizadas de forma dinâmica, isto é, a cada *epoch* são aplicadas ao *dataset*, enquanto que outras são adicionadas ao conjunto de dados utilizando a função *ConcatDataset*.

4.2.1 Rotation

A transformação de rotação foi feita a partir da função *RandomRotation* do módulo *torchvision*, fornecendo o intervalo [-15, 15] como argumento para representar o grau de cada rotação. O objetivo dessa transformação é simular a captura de sinais de trânsito em ângulos oblíquos ou sinais que possam estar tortos.

4.2.2 Perspective

Relativamente à transformação da perspetiva, utilizamos a função *RandomPerspective* do módulo *torchvision* com um fator de distorção de 0.5. Esta função permite alterar de forma aleatória o ângulo em que a foto foi tirada.

4.2.3 Crop

Para a transformação *crop* foi utilizada a função *RandomCrop* do módulo *torchvision* com um *padding* de 4.

Para além das transformações mencionadas, foram testadas as transformações *translate*, *shear* e *randomSizedCrop* que, devido aos maus resultados, acabaram por ser removidas.

Desta forma, como podemos observar na Tabela 4, as transformações geométricas melhoraram significativamente a *accuracy* dos modelos.

Modelo	Modelo I	Modelo II	Modelo II
Sem *Geometric Transf.*	98,0127	98,0364	98.3135
Com *Geometric Transf.*	99,3666	99.5012	99.6516

Tabela 4: *Accuracy* de cada modelo após aplicar *geometric transformations*.

4.3 Oclude Transformations

Nesta secção, são exploradas as transformações relacionadas com imagens ocluídas, com o objetivo de tratar imagens cujos sinais têm uma menor visibilidade devido, por exemplo, a postes, árvores, carros e até vandalismo.

4.3.1 Random Erasing

Para simular as situações acima descritas, utilizamos a função *RandomErasing* que, dada uma imagem, irá selecionar de forma aleatória uma região retangular para apagar os seus pixeis. Esta função foi aplicada ao *dataset* adicional, sendo que os seus resultados estão apresentados na Tabela 5. Como podemos observar, apesar de não muito significativa, houve uma melhoria geral com destaque para o modelo III que conta até ao momento com a melhor *accuracy*.

Modelo	Modelo I	Modelo II	Modelo II
Sem *Oclude Transf.*	99,3666	99.5012	99.6516
Com *Oclude Transf.*	99.4220	99.5329	99,6675

Tabela 5: *Accuracy* de cada modelo após aplicar *oclude transformations*.

4.4 Distortion Transformations

O último conjunto de transformações aplicadas diz respeitos a transformações de distorção, que têm como objetivo simular os diversos casos presentes no *dataset* onde a imagem do sinal de trânsito apresenta uma baixa qualidade ou algum desfoque.

4.4.1 Motion Blur

No que toca à transformação *Motion Blur*, esta foi aplicada às imagens do *dataset* através da função *MotionBlur* da biblioteca *kornia*, própria para ser utilizada juntamente com o *pytorch*. Com esta transformação fomos capazes de simular imagens desfocadas.

4.4.2 Gaussian Noise

Com o objetivo de simular imagens de pouca qualidade utilizamos a função *random_noise* da biblioteca *skimage* para aplicar diferentes tipos de ruído, nomeadamente, o *Gaussian noise*, o *Sperkle noise* e o *Salt&Pepper noise*.

É importante referir que devido à crescente complexidade do *dataset*, estas transformações foram apenas aplicadas ao terceiro modelo, uma vez que é o que apresenta melhores resultados. Para além disso, tentamos aplicá-las a um novo *dataset* e juntá-lo, posteriormente, aos outros 2. No entanto, dado que esta abordagem aumentou consideravelmente a complexidade do *dataset* e tendo em conta as limitações dos recursos utilizados, decidimos voltar atrás e aplicar as transformações apenas ao segundo *dataset* e ao *dataset* original.

Após realizar vários testes, o tipo de ruído que levou a uma melhor *accuracy* por parte do modelo, foi o *Sperkle*, pelo que foi este o utilizado no terceiro modelo. Os resultados encontram-se na Tabela 6 e, como podemos observar, houve uma ligeira melhoria na *accuracy*, sendo este o melhor modelo obtido.

Modelo	Modelo III
Sem *Distortion Transf.*	99,6675
Com *Distortion Transf.*	99.7229

Tabela 6: *Accuracy* de cada modelo após aplicar *distortion transformations*.

5 Ensemble Learning

Nesta segunda fase, foi-nos proposto estudar o potencial da utilização do *ensemble* de redes. Para isto, selecionamos as cinco redes que tiveram melhor desempenho na fase anterior para fazerem parte do *ensemble*. Abaixo encontram-se com mais detalhe as redes baseadas no Modelo III escolhidas:

- Modelo 0 - tem uma *acuraccy* de 99.73% e corresponde ao melhor modelo obtido na fase anterior, com todas as transformações.
- Modelo 1 - tem uma *acuraccy* de 99.63% e corresponde ao modelo 3 com todas as transformações aplicadas exceto o *Motion Blur* .
- Modelo 2 - tem uma *acuraccy* de 98.31% e corresponde ao modelo 3 com apenas *color transformations*.
- Modelo 3 - tem uma *acuraccy* de 99.70% e corresponde ao modelo 3 com *color* e *geometric transformations*.
- Modelo 4 - tem uma *acuraccy* de 99.67% e corresponde ao modelo 3 com *color, geometric* e *oclude transformations*.

É de notar que a *accuracy* obtida no modelo 3 foi conseguida através de novas combinações de parâmetros nas transformações geométricas.

Desta forma, passamos à execução de *ensembles*, nomeadamente, do *Voting Ensemble*. Neste tipo de *ensemble*, cada modelo faz uma previsão para cada instância e, a classe que receber mais votos é a escolhida como previsão final.

Ensemble	*Voting Ensemble*
Accuracy	0.9974

Tabela 7: *Accuracy* de cada *ensemble*.

O *Voting Ensemble* provocou um ligeiro aumento da *accuracy* como podemos observar na Tabela 7.

6 Conclusão

Este relatório explorou o uso de modelos de *Deep Learning* aplicados ao conjunto de dados GTSRB (sinais de trânsito alemães) com o objetivo de obter o melhor resultado possível em termos de *accuracy* no conjunto de teste.

Na primeira etapa, foram utilizados diversos modelos com diferentes técnicas de *data augmentation*, tanto no pré-processamento como dinamicamente.

Na segunda parte do estudo, investigou-se o potencial de utilizar *ensembles* de redes neuronais. Os modelos treinados na primeira fase foram combinados através do *Voting Ensemble*, que faz uma previsão para cada instância e, escolhe a classe que receber mais votos como previsão final, de maneira a obter uma previsão final mais robusta.

Para além disso, gostaríamos de ter explorado mais as técnicas de *ensemble* disponíveis e outras combinações de transformações que não foram aplicadas devido à limitações a nível de recursos.

Por fim, concluímos que a utilização de técnicas avançadas de processamento de imagem é crucial para melhorar o desempenho de modelos de *Deep Learning* na classificação de sinais de trânsito.