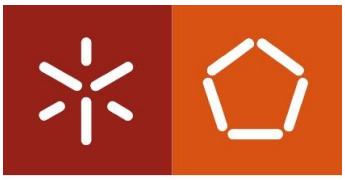

Universidade do Minho

Departamento de Informática

# Reconhecimento de Sinais de Transito com Redes Neuronais

José Ferreira (A83683), Jorge Mota (A85272)

June 17, 2021

# Contents

# Chapter 1

# Introdução

Este projeto consiste em criar uma rede neuronal convolucional com a capacidade de identificar sinais de transito com a maior precisão na classificação possível. O dataset de imagens de sinais de transito utilizados foi o *dataset* alemão GTSRB (German Traffic Sign Recognition Benchmark) ( Imagem 1.1).

Ao longo deste relatório iremos descrever as varias etapas tomadas para atingir o melhor resultado possível. Inicialmente iremos descrever a estrutura do projeto e como ele foi construído. Em seguida iremos descrever e comparar as 3 redes neuronais convolucionais desenvolvidas. Por fim iremos analisar os resultados obtidos com essas mesmas redes com o intuito de entender o que leva uma rede neuronal a classificar de forma eficaz as imagens do *dataset* alvo.


Figure 1.1: dataset GTSRB (German Traffic Sign Recognition Benchmark)

# Chapter 2

# Estrutura do Projeto

Com o intuito de facilitar e agilizar o desenvolvimento de varias redes neuronais convolucionais distintas para o projeto, assim como proporcionar um melhor ambiente de desenvolvimento, organizamos este projeto de forma a conseguir cumprir os seguintes objetivos:

- **Modularidade** do código no que diz respeito as varias etapas que constroem e treinam a rede.
- A possibilidade de **reutilização** de cada componente entre módulos para facilitar a composição de vários membros e testar várias arquiteturas.
- **Organização** de cada parte do projeto.
- Poder escrever-se código em *scripts python* e utilizar arquiteturas com os módulos num ***jupyter notebook***.

Para cumprir estes objetivos, a estrutura de ficheiros deste projeto foi organizada da seguinte forma:

```
.
├── data/
│   └── // Onde estão localizados os datasets e recursos de treino/teste
├── models/
│   └── // Onde todos os modelos treinados são guardados
└── src/
    ├── cnn1/
    ├── cnn2/
    ├── cnn3/
    └── // Todos os scripts em python, incluindo os 3 modulos das redes desenvolvidas
```

Segmentamos o ciclo de vida da rede em 3 funções principais para **obter o *dataset***, **fazer o modelo** e **treina-lo**:

- `fetch_data()` - Carrega o *dataset* alemão para um *train set*, *validation set* e *test set*.
- `make_model()` - Constrói o modelo da rede.
- `train()` - Treina o modelo ou carrega-o a partir de um ficheiro se este tiver sido treinado previamente

# Chapter 3

# Redes Desenvolvidas

Foram desenvolvidas três redes neuronais convolucionais distintas com o objectivo de as comparar e tentar obter aquela com a melhor performance possível.

## 3.1 Rede Neural Convolucional 1

Para uma primeira implementação de uma rede que classificasse sinais de transito, decidimos basear-nos num modelo que foi explorado nas aulas práticas da unidade curricular.

Esta foi a nossa primeira tentativa utilizada para o módulo *cnn1* e que nos orientou para entender os blocos estruturais necessários para a criação de uma rede neuronal convolucional e da vantagem de utilizar a função de activação *LeakyReLU* em comparação à normal *ReLU*.

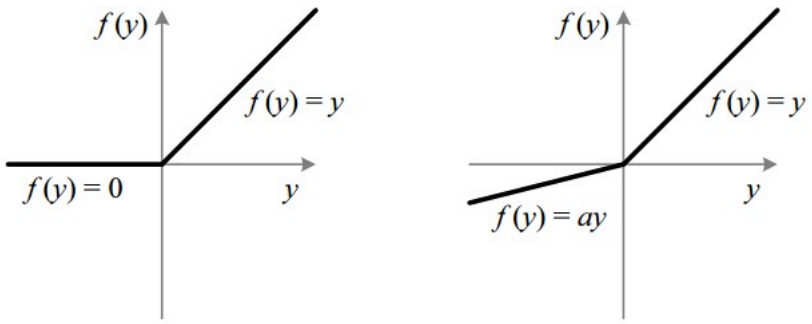


Figure 3.1: Comparação *ReLU* e *LeakyReLU*

Experimentamos algumas variações para observar os resultados durante e depois do treino, e no final obtemos resultados que são discutidos no capitulo 4.

### 3.1.1 Estruturas Recorrentes

Para esta arquitetura de rede, e muitas outras, podemos notar conjuntos de *layers* regulares que funcionam blocos estruturais na rede, que se podem agrupar em camadas Convolucionais e *layers* Densas.

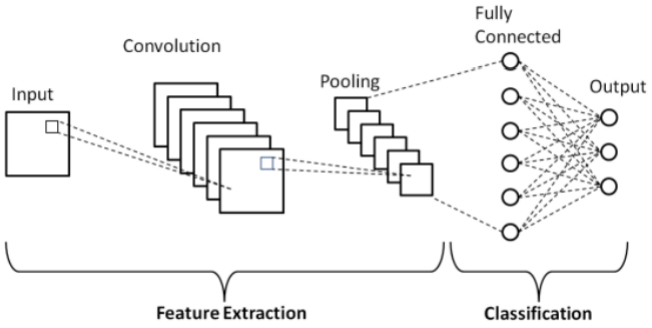


Figure 3.2: Figura ilustrativa de uma CNN típica

As *layers* convolucionais normalmente são compostas por:
**Conv2D -> BatchNormalization -> LeakyReLU -> MaxPooling2D**
e contribuem para a aprendizagem das características da imagem ao se sobrepor estrategicamente estas camadas

Depois de se extraírem as características da imagem, usa-se camadas densas que estão todas interligadas de forma a aprender a relação de combinações dessas características. Podem-se compôr estas camdas de várias maneiras, Ex: **Flatten -> Dense -> LeakyReLU**.

Com o objetivo de obter o melhor resultado possível e aprimorar os resultados obtidos experimentamos diversas variacoes de valores.

## 3.2 Rede Neural Convolucional 2

Depois de se testar para uma implementação que já estávamos familiarizados, decidimos tomar uma abordagem mais completa e procuramos basear-nos no desenvolvimento de uma rede inspirada nos modelos que dominam a classificação de sinais para este *dataset*, que neste caso escolhemos a **MicronNet** [1].

| Rank | Model | Accuracy ↑ |
|---|---|---|
| 1 | **CNN with 3 Spatial Transformers** | 99.71% |
| 2 | **Sill-Net** | 99.68% |
| 3 | **MCDNN** | 99.5% |
| 4 | **MicronNet** (fp16) | 98.9% |

Figure 3.3: Top 4 Redes Neuronais do dataset alemão

A micronnet é uma *Deep Convolutional Neural Network* compacta que realiza classificações de sinais em tempo útil a ponto de se usar em sistemas computacionalmente limitados. Esta rede encaixa perfeitamente nos requisitos que queríamos sendo que a arquitetura da rede é reduzida, e não leva tanto tempo para treinar.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 1)         4
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 1)         4
_________________________________________________________________
re_lu (ReLU)                 (None, 32, 32, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 29)        754
_________________________________________________________________
batch_normalization_1 (Batch (None, 28, 28, 29)        116
_________________________________________________________________
re_lu_1 (ReLU)               (None, 28, 28, 29)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 29)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 59)        15458
_________________________________________________________________
batch_normalization_2 (Batch (None, 13, 13, 59)        236
_________________________________________________________________
re_lu_2 (ReLU)               (None, 13, 13, 59)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 59)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 6, 6, 74)          39368
_________________________________________________________________
batch_normalization_3 (Batch (None, 6, 6, 74)          296
_________________________________________________________________
re_lu_3 (ReLU)               (None, 6, 6, 74)          0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 74)          0
_________________________________________________________________
flatten (Flatten)            (None, 296)               0
_________________________________________________________________
dense (Dense)                (None, 300)               89100
_________________________________________________________________
batch_normalization_4 (Batch (None, 300)               1200
_________________________________________________________________
re_lu_4 (ReLU)               (None, 300)               0
_________________________________________________________________
dense_1 (Dense)              (None, 300)               90300
_________________________________________________________________
re_lu_5 (ReLU)               (None, 300)               0
_________________________________________________________________
dense_2 (Dense)              (None, 43)                12943
_________________________________________________________________
softmax (Softmax)            (None, 43)                0
=================================================================
Total params: 249,779
Trainable params: 248,853
Non-trainable params: 926
_________________________________________________________________
```

Alguns aspetos que são relevantes apontar relativamente a esta rede:

- As Layers convolucionais organizam-se de maneira semelhante ao que já tinhamos visto para a primeira rede, e ainda se aproveita de algumas otimizações sugeridas no artigo original.
- Para seguir o artigo da *Micronnet*, utilizamos o otimizador SGD (Stochastic Gradient Descent) com *learning rate* 0.007, e momentum 0.9

Este foi um recurso fundamental para compreendermos de forma mais detalhada como se deve desenvolver redes neuronais convolucionais para este tipo de *dataset*.

## 3.3 Rede Neural Convolucional 3

Finalmente, após o desenvolvimento de duas redes com o intuito de entender as principais caracterísitcas de uma rede eficaz na classificação de sinais de transito, desenvolvemos uma rede neuronal a partir dos fundamentos aprendidos que idealmente conseguisse trazer resultados mais fiáveis.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 1)         4
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 1)         4
_________________________________________________________________
re_lu (ReLU)                 (None, 32, 32, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 64)        1664
_________________________________________________________________
batch_normalization_1 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 28, 28, 64)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 128)       73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 12, 12, 128)       512
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 12, 12, 128)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 256)         295168
_________________________________________________________________
batch_normalization_3 (Batch (None, 4, 4, 256)         1024
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 4, 4, 256)         0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 256)               262400
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024
_________________________________________________________________
re_lu_1 (ReLU)               (None, 256)               0
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
re_lu_2 (ReLU)               (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 43)                5547
_________________________________________________________________
softmax (Softmax)            (None, 43)                0
=================================================================
Total params: 674,355
Trainable params: 672,945
Non-trainable params: 1,410
_________________________________________________________________
```

### 3.3.1 Data Augmentation

Para além desta rede, decidimos ainda ampliar o conjunto de dados e fazer *data augmentation* do *dataset* para o dobro do tamanho com o intuito de combater o *overfitting* do modelo, e obter melhores resultados no *Test Set*.

Para fazer esta *augmentation* duplicamos o conjunto de treino e transformamos essa metade com parâmetros aleatórios de:

- Brilho
- Contraste
- Saturação
- Translação da imagem
- Rotação da imagem

Ao aplicar estas transformações todas ao mesmo tempo a cada imagem ficamos com uma variante relevante da imagem original, mantendo as caraterísticas o suficiente para continuar a ser classificado pela *label* respetiva.

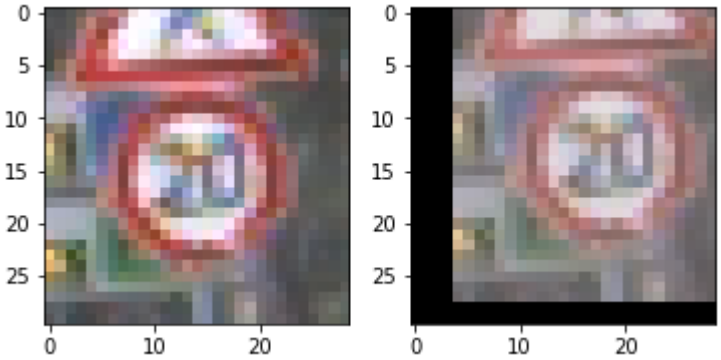


Figure 3.4: Imagem original e imagem depois de aplicar as tranformações

# Chapter 4

# Resultados

Por fim, treinamos as redes descritas anteriormente e registamos os resultamos da precisão de classificação do conjunto de teste:

| Net | Epochs | Accuracy |
|---|---|---|
| cnn1 | 20 | 97.32% |
| cnn2 | 50 | 93.42% |
| cnn2 | 72 | 95.36% |
| cnn2 DA | 150 | 94.89% |
| cnn3 | 20 | 95.49% |
| cnn3 | 50 | 95.29% |

Table 4.1: Tabela de precisão das redes

Para avaliar a eficácia das redes, tomamos a decisão de que as redes só seriam boas a classificar sinais, se tivessem uma boa percepção do que é o sinal, e não da junção de sinais com fundos do mundo real, portanto juntamos um conjunto muito limitado de teste com imagens artificiais explicitas de alguns sinais, ao qual nós chamamos de "*testes de ouro*", e caso a rede não consiga identificar corretamente estes casos, que não têm noção das caraterísticas dos sinais.


Analisando os resultados, podemos notar claramente que os melhores resultados obtidos foram para a **primeira rede** `cnn1` que não so apresenta a melhor *accuracy* com o menor número de *epochs*, como também classifica os nossos casos de ouro corretamente.

Por outro lado, a segunda rede conseguiu até 95.36% e já não cumpre o nosso requesito de ouro, apesar disso tentamos treina-la com aumento dos dados até 150 epochs, o que não se mostrou relevante.

Por último, a terceira rede que desenvolvemos dispôs de uma *accuracy* de 95.49% para apenas 20 épocas, mas infelizmente não conseguiu classificar corretamente os nossos casos de ouro

# Bibliography

[1] Alexander Wong, Mohammad Javad Shafiee, and Michael St Jules. Micronnet: A highly compact deep convolutional neural network architecture for real-time embedded traffic sign classification. *IEEE Access*, 6:59803–59810, 2018.