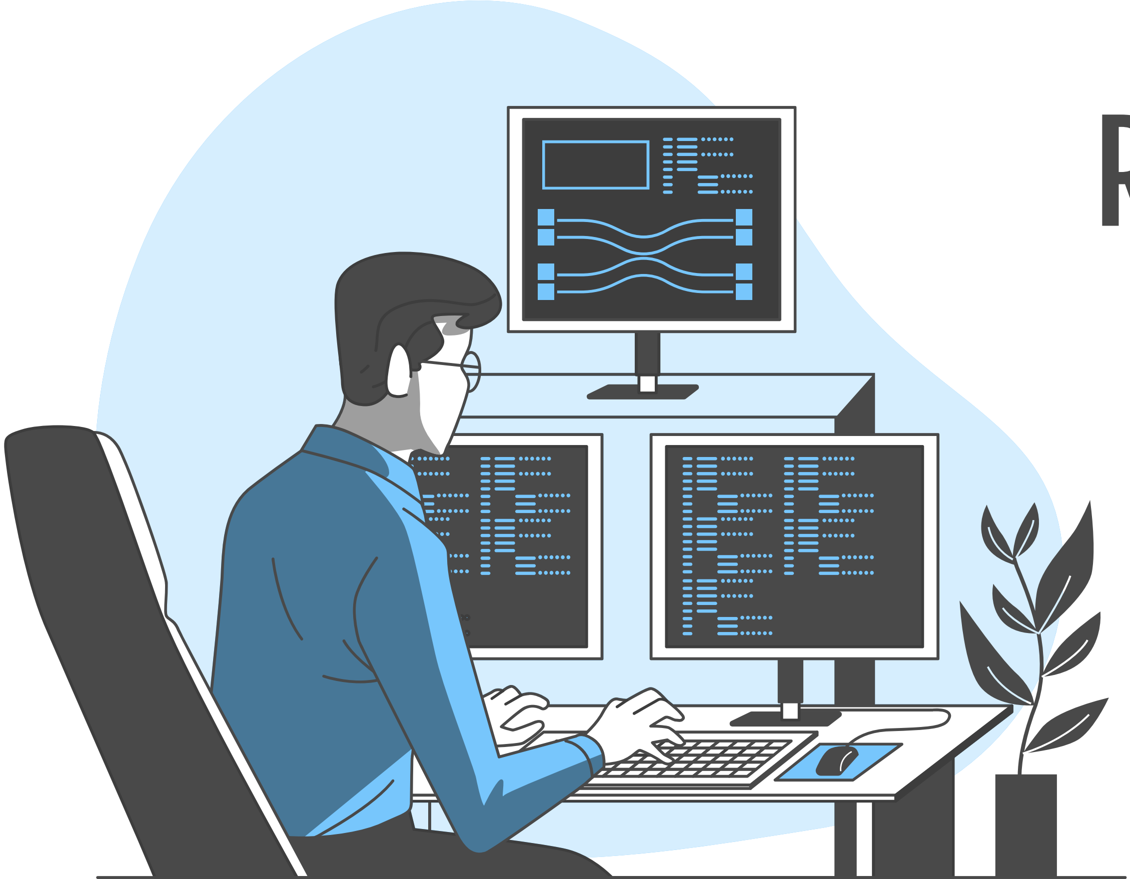
Google Colab



Recognition of traffic signs

Deep Learning project

Domenico Armillotta
Stefano Dugo

Project goal:

IDEA:

The project aims to create an image **classifie**r that recognizes traffic signs , given an image.
This type of system is particularly useful in applications of self-driving cars or driver assistance in driving.
In fact, this specific application is part of ADAS systems that are more generally able to detect, for example, speed limits, but also access and overtaking bans.

IMPLEMENTATION:

The objective is to create **two** image classifiers, one developing a CNN from scratch, and the second with a pre-trained keras
model but using feature extraction and fine tuning techniques.
Then compare the results.

Related Works:

The results found on the internet regarding the GTSRB dataset, showed similar performance to ours.
On kaggle many of the papers analyzed are superficial on some aspects such as balancing training, or comparing multiple classifiers.
While on github more complete works are present:

1	**CNN with 3 Spatial Transformers**	99.71%	×	Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods			2018
2	**Sill-Net**	99.68%	×	Sill-Net: Feature Augmentation with Separated Illumination Representation			2021
3	**MicronNet** (fp16)	98.9%	×	MicronNet: A Highly Compact Deep Convolutional Neural Network Architecture for Real-time Embedded Traffic Sign Classification			2018
4	**SEER** (RegNet10B)	90.71%	✓	Vision Models Are More Robust And Fair When Pretrained On Uncurated Images Without Supervision			2022

01

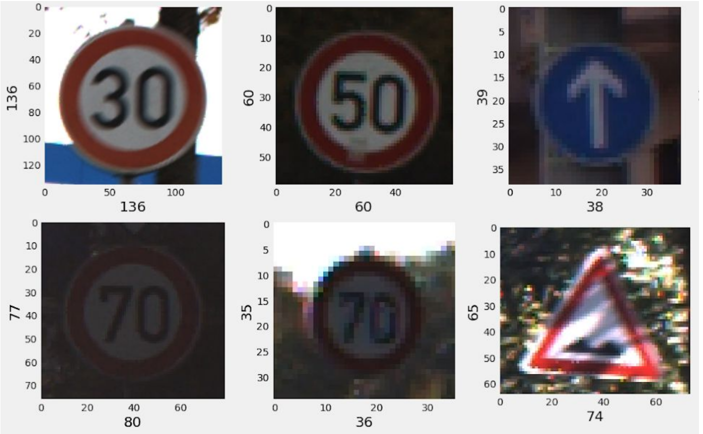
CNN FROM SCRATCH

1. Dataset

Source: [GTSRB - German Traffic Sign Recognition Benchmark | Kaggle]
Class = 43
Dimension : 50k image

Multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011.

Here are some examples of street sign images:



2. Dataset Splitting & Preparation

Split Phase:

Training = 32k

Validation = 7k

Testing = 18k

Training set and Validation Set according to the 80/20 proportions, while the images in the Test folder we will use to compose the Test set.

Augmentation

it's a good practice to artificially introduce sample diversity by applying random yet realistic transformations. According to the keras documentation, there are two methods to carry out augmentation, we have chosen to implement the second strategy, i.e. to apply it directly on the train dataset.

Handle Imbalanced Dataset

One way to address class imbalance in a convolutional neural network (CNN) is to use class weights. Class weights are used to adjust the loss function of the model in order to give more weight to the examples in the minority classes

In Keras library was used the class_weight argument in the compile() function.

AlexNet Model Based

Architecture:

- **Dropout :** randomly reducing the number of interconnecting neurons
- **Convolutional layer:** houses the convolution operation that occurs between the filters and the images passed
- **Batch Normalisation layer:**technique that mitigates the effect of unstable gradients
- **MaxPooling layer:** max pixel value of pixels that fall in the receptive field is taken as the output
- **Flatten layer:** flattens the input image data into a one-dimensional array.
- **Dense Layer:**A dense layer has an embedded number of arbitrary units/neurons
- **Softmax Activation Function:** a type of activation function

```
model = keras.models.Sequential()
#1st Convolutional Layer
model.add(keras.layers.Conv2D(filters=32,kernel_size=3,activation='relu', input_shape = [50, 50,3]))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=2))
model.add(keras.layers.Dropout(0.2))
#2nd Convolutional Layer
model.add(keras.layers.Conv2D(filters=64,kernel_size=3,activation='relu'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=2))
model.add(keras.layers.Dropout(0.2))
#3rd Convolutional Layer
model.add(keras.layers.Conv2D(filters=128,kernel_size=3,activation='relu'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=2))
model.add(keras.layers.Dropout(0.2))
#4th Convolutional Layer
model.add(keras.layers.Conv2D(filters=256,kernel_size=3,activation='relu'))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.MaxPooling2D(pool_size=2))
model.add(keras.layers.Dropout(0.2))
#Passing it to a Fully Connected layer
model.add(keras.layers.Flatten())

# 1st Fully Connected Layer
model.add(keras.layers.Dense(1024,activation='relu'))

# 2st Fully Connected Layer
model.add(keras.layers.Dense(1024,activation='relu'))

#Softmax layer for output
model.add(keras.layers.Dense(43,activation='softmax'))
```

Total params: 1,747,179
Trainable params: 1,746,219
Non-trainable params: 960

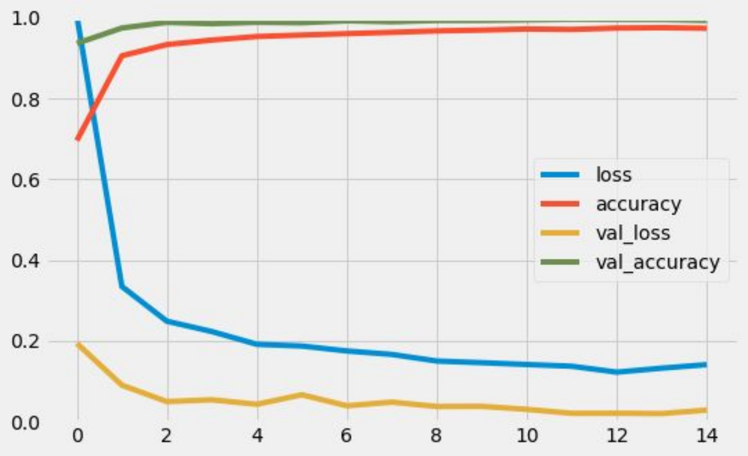
Training :

loss function : “cateorical_crossentropy”. We have trained the model in 15 epochs, but this is a hyper-parameter that we will tune to in the future

Early stopping callback function:

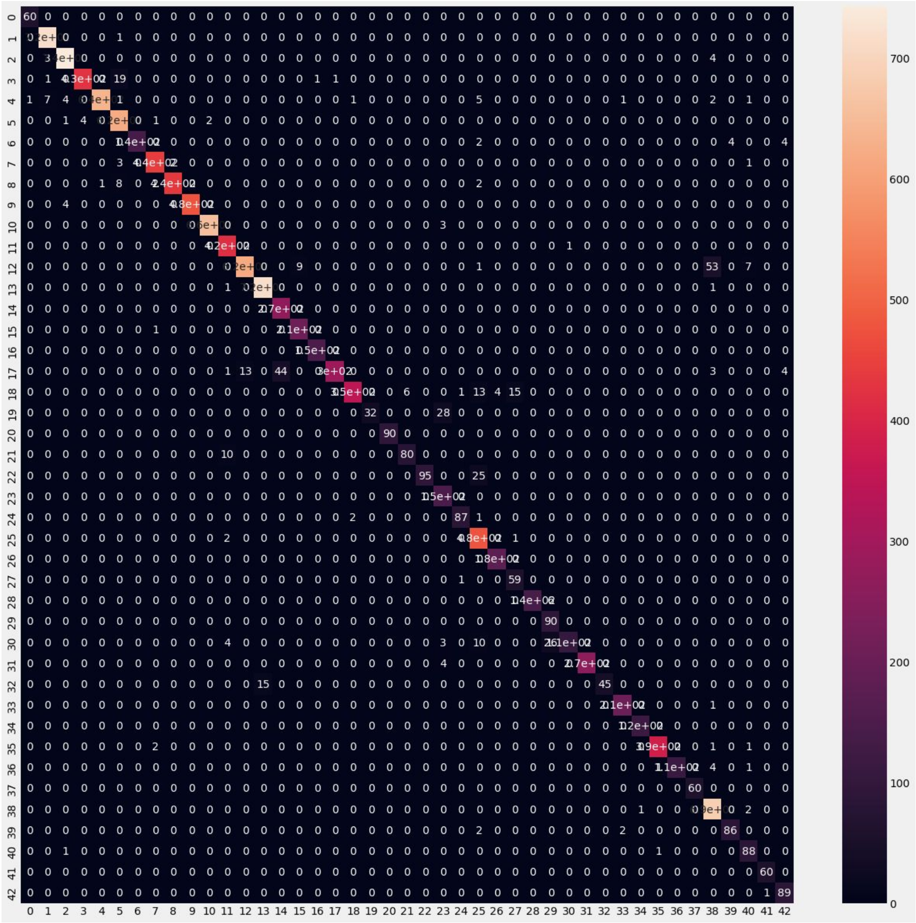
We used the early stopping that is a form of regularization that can be used to prevent overfitting. Is implemented by specifying a callback function in the .fit function

Evaluate the model:



accuracy			0.97	12630
macro avg	0.96	0.95	0.95	12630
weighted avg	0.97	0.97	0.97	12630

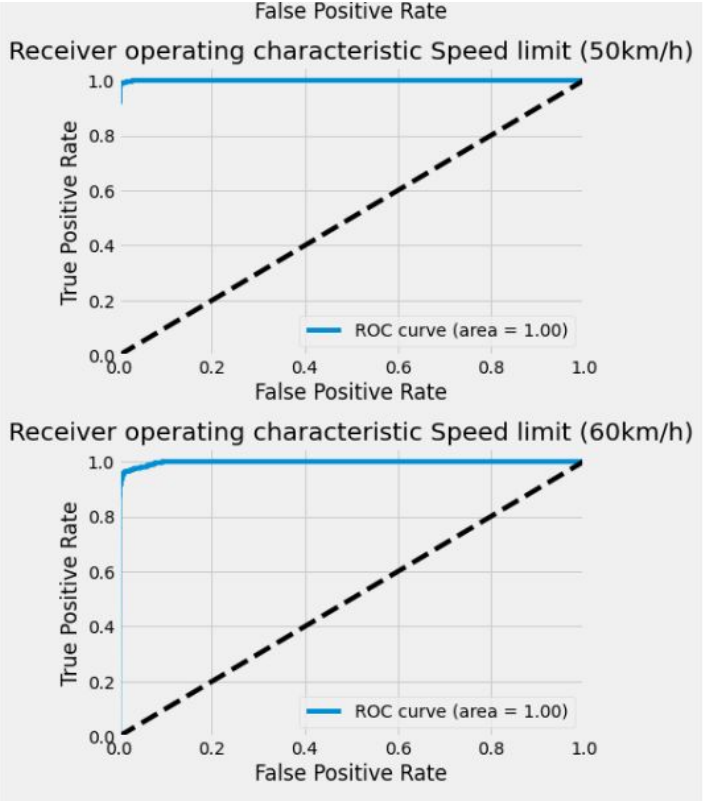
Confusion Matrix :



Visual Network Test:

15 random images of the dataset were put to the model test, and we observe how they are classified

ROC Curves (only 2 shown for example):



MNIST Model Based

Architecture:

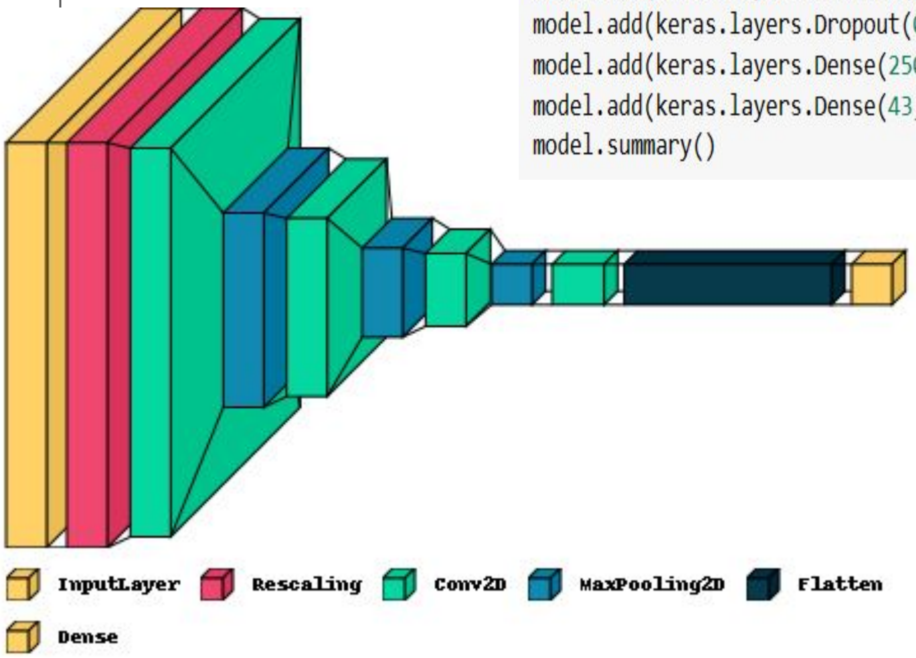
Stack of alternated Conv2D (with ReLu activation function) and MaxPooling2D layers.

Note that the depth of the feature maps is progressively increasing in the network (from 32 to 256), while the size of the feature maps is decreasing.

We have trained the model in 15 epochs, but this is a hyper-parameter that we will tune to in the future.

```
Total params: 616,291
Trainable params: 616,291
Non-trainable params: 0
```

```python
model = keras.models.Sequential()
model.add(keras.layers.Conv2D(filters=32,kernel_size=3,activation='relu', input_shape = [50, 50,3]))
model.add(keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(keras.layers.Conv2D(filters=64 , activation='relu',kernel_size=3))
model.add(keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(keras.layers.Conv2D(filters=128 , activation='relu' ,kernel_size=3))
model.add(keras.layers.MaxPooling2D(pool_size=(2,2)))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(250,activation='relu'))
model.add(keras.layers.Dense(43,activation='softmax'))
model.summary()
```



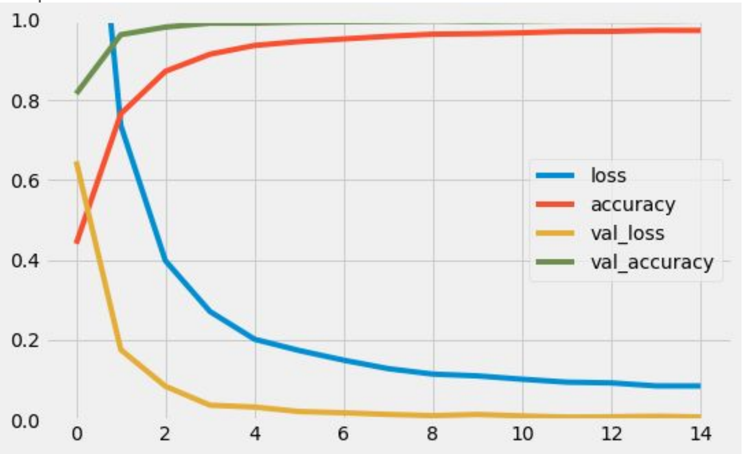
Training :

loss function : “cateorical_crossentropy”. We have trained the model in 15 epochs, but this is a hyper-parameter that we will tune to in the future

Early stopping callback function:

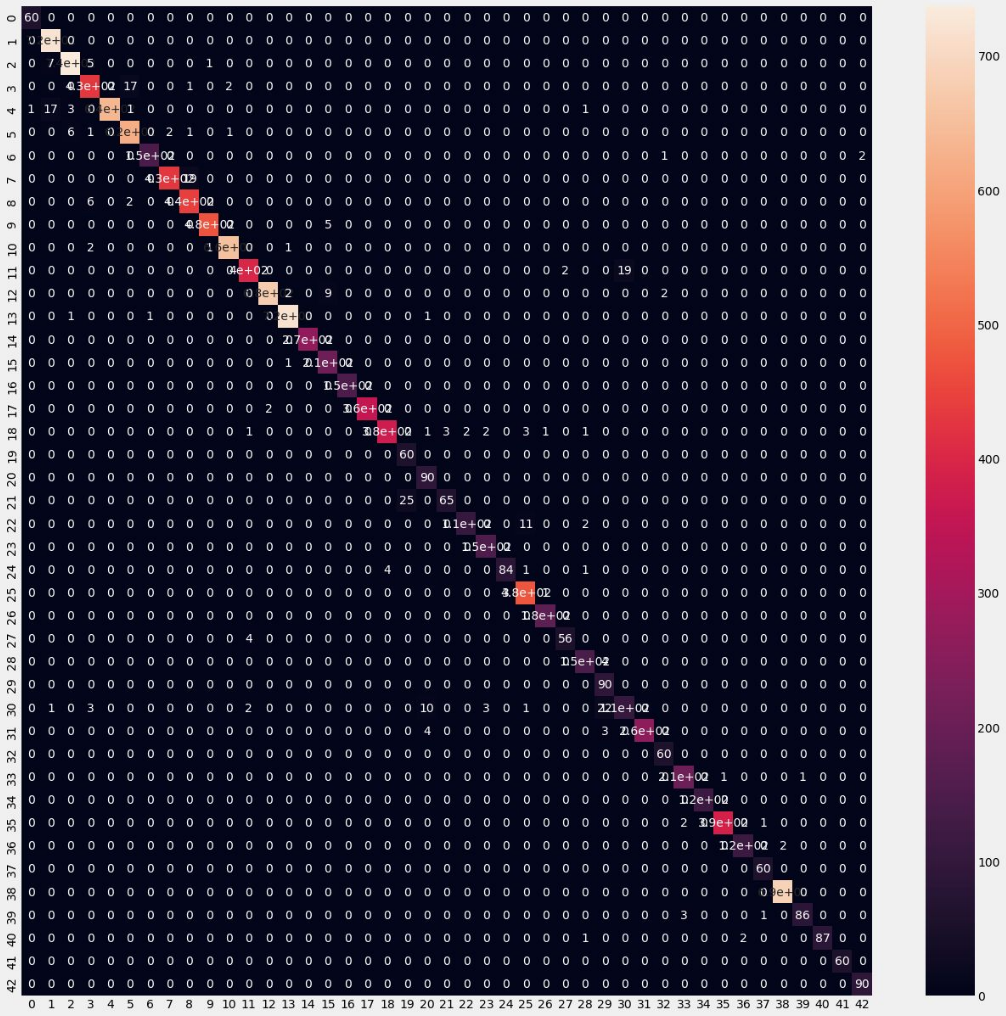
We used the early stopping that is a form of regularization that can be used to prevent overfitting. Is implemented by specifying a callback function in the .fit function

Evaluate the model:



accuracy			0.98	12630
macro avg	0.96	0.97	0.97	12630
weighted avg	0.98	0.98	0.98	12630

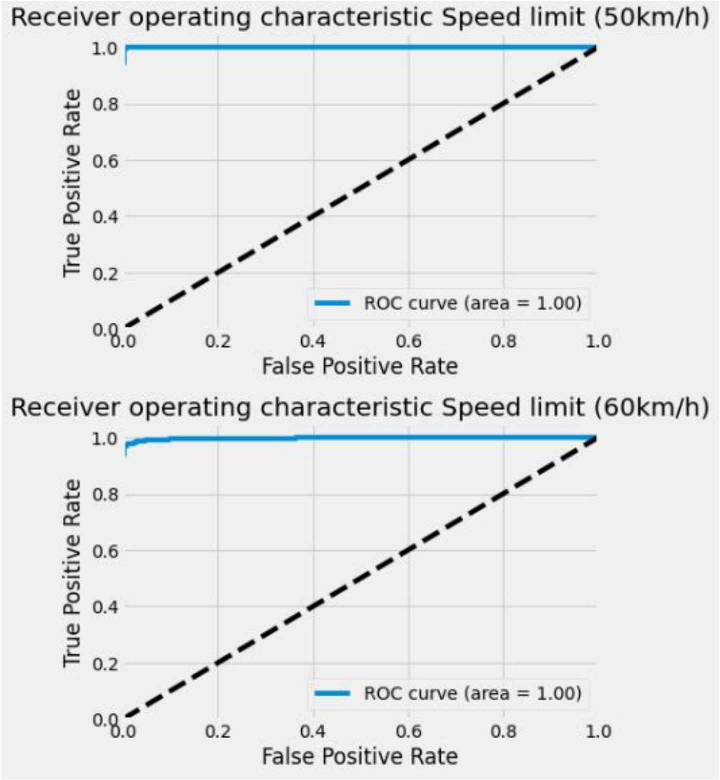
Confusion Matrix :



Visual Network Test:

15 random images of the dataset were put to the model test, and we observe how they are classified

ROC Curves (only 2 shown for example):



02

Hyper parameter Tuning

Tuning with keras Tuner

We tried to fine-tune the hyperparameters in order to improve performance using the Keras library called "keras Tuner".

This process was only carried out for Mnist.
 The optimized parameters were:
the number of training epochs, the learning rate, and the number of units in the dense layers of the network.
All other parameters remained unchanged
This are the result of the search:

```
Best epoch: 37
The optimal number of units in the first densely-connected layer is 288
the optimal learning rate for the optimizer is 0.001
```

03

CNN USING VCC19

Approach

We used the VGG19 architecture to test a pre-trained network on our dataset through transfer learning. VGG19 is a convolutional neural network (CNN) trained on the ImageNet dataset for image classification. It consists of 19 layers and has been trained to recognize 1000 different classes of objects.

Feature Extraction (without data augmentation)

We applied the preprocessing of VGG19 to the input data and then predicted with the convolutional base, then used the densely connected classifier to train, validate and test the model.

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 50, 50, 3)]       0

 block1_conv1 (Conv2D)       (None, 50, 50, 64)        1792

 block1_conv2 (Conv2D)       (None, 50, 50, 64)        36928

 block1_pool (MaxPooling2D)  (None, 25, 25, 64)        0

 block2_conv1 (Conv2D)       (None, 25, 25, 128)       73856

 block2_conv2 (Conv2D)       (None, 25, 25, 128)       147584

 block2_pool (MaxPooling2D)  (None, 12, 12, 128)       0

 block3_conv1 (Conv2D)       (None, 12, 12, 256)       295168

 block3_conv2 (Conv2D)       (None, 12, 12, 256)       590080

 block3_conv3 (Conv2D)       (None, 12, 12, 256)       590080

 block3_conv4 (Conv2D)       (None, 12, 12, 256)       590080

 block3_pool (MaxPooling2D)  (None, 6, 6, 256)         0

 block4_conv1 (Conv2D)       (None, 6, 6, 512)         1180160

 block4_conv2 (Conv2D)       (None, 6, 6, 512)         2359808

 block4_conv3 (Conv2D)       (None, 6, 6, 512)         2359808

 block4_conv4 (Conv2D)       (None, 6, 6, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 3, 3, 512)         0

 block5_conv1 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_conv2 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_conv3 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_conv4 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________
```

Feature Extraction (with data augmentation)

We use feature extraction with a data augmentation layer, freezing all the layers before doing that, and applying the VGG19 convolutional base directly to our model (also applying input scale with *vgg19.preprocess_input*).

```
#freezing all the layers
conv_base.trainable = False
#Setting trainable to False empties the list of trainable weights of the layer or model
print("Trainable weights after freezing", len(conv_base.trainable_weights))

conv_base.summary()
```

```
Trainable weights after freezing 0
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 50, 50, 3)]       0

 block1_conv1 (Conv2D)       (None, 50, 50, 64)        1792

 block1_conv2 (Conv2D)       (None, 50, 50, 64)        36928

 block1_pool (MaxPooling2D)  (None, 25, 25, 64)        0

 block2_conv1 (Conv2D)       (None, 25, 25, 128)       73856

 block2_conv2 (Conv2D)       (None, 25, 25, 128)       147584

 block2_pool (MaxPooling2D)  (None, 12, 12, 128)       0

 block3_conv1 (Conv2D)       (None, 12, 12, 256)       295168

 block3_conv2 (Conv2D)       (None, 12, 12, 256)       590080

 block3_conv3 (Conv2D)       (None, 12, 12, 256)       590080

 block3_conv4 (Conv2D)       (None, 12, 12, 256)       590080

 block3_pool (MaxPooling2D)  (None, 6, 6, 256)         0

 block4_conv1 (Conv2D)       (None, 6, 6, 512)         1180160

 block4_conv2 (Conv2D)       (None, 6, 6, 512)         2359808

 block4_conv3 (Conv2D)       (None, 6, 6, 512)         2359808

 block4_conv4 (Conv2D)       (None, 6, 6, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 3, 3, 512)         0

 block5_conv1 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_conv2 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_conv3 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_conv4 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

=================================================================
Total params: 20,024,384
Trainable params: 0
Non-trainable params: 20,024,384
_________________________________________________________________
```

Fine Tuning

For exploiting fine-tuning, we instantiated the vgg19 convolutional base and unfreezed the last 5 layers, then applied the densely connected classifier on top of the network

It's not convenient to fine tune all the layers;

```
[ ] # Freezing all layers until the fifth from the last
    conv_base.trainable = True
    for layer in conv_base.layers[:-5]:
     layer.trainable = False
```

```
[ ] conv_base.summary()
```

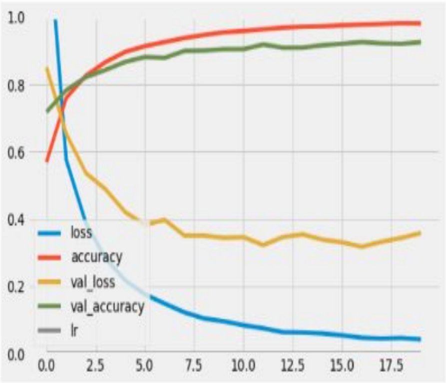
```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 50, 50, 3)]       0

 block1_conv1 (Conv2D)       (None, 50, 50, 64)        1792

 block1_conv2 (Conv2D)       (None, 50, 50, 64)        36928

 block1_pool (MaxPooling2D)  (None, 25, 25, 64)        0

 block2_conv1 (Conv2D)       (None, 25, 25, 128)       73856

 block2_conv2 (Conv2D)       (None, 25, 25, 128)       147584

 block2_pool (MaxPooling2D)  (None, 12, 12, 128)       0

 block3_conv1 (Conv2D)       (None, 12, 12, 256)       295168

 block3_conv2 (Conv2D)       (None, 12, 12, 256)       590080

 block3_conv3 (Conv2D)       (None, 12, 12, 256)       590080

 block3_conv4 (Conv2D)       (None, 12, 12, 256)       590080

 block3_pool (MaxPooling2D)  (None, 6, 6, 256)         0

 block4_conv1 (Conv2D)       (None, 6, 6, 512)         1180160

 block4_conv2 (Conv2D)       (None, 6, 6, 512)         2359808

 block4_conv3 (Conv2D)       (None, 6, 6, 512)         2359808

 block4_conv4 (Conv2D)       (None, 6, 6, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 3, 3, 512)         0

 block5_conv1 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_conv2 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_conv3 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_conv4 (Conv2D)       (None, 3, 3, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

=================================================================
Total params: 20,024,384
Trainable params: 9,439,232
Non-trainable params: 10,585,152
_________________________________________________________________
```

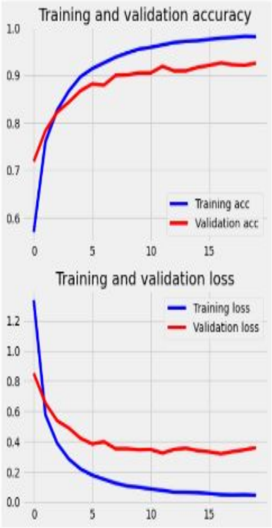
Highlighted Parameters

- *Loss function: Categorical Cross Entropy*
- *Optimizers: RMSprop vs Adam*
- *Learning Rate Reduction*
- *Batch Normalization layer*
- *Dropout layer*
- *Dense Layer: ReLU activation function*

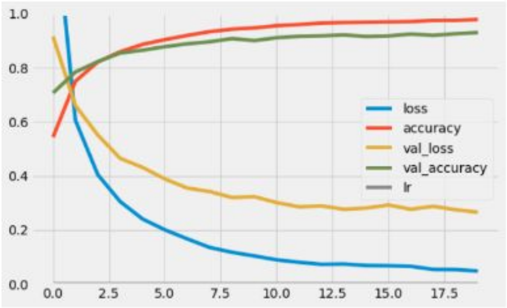
Performances – Fast Feature Extraction

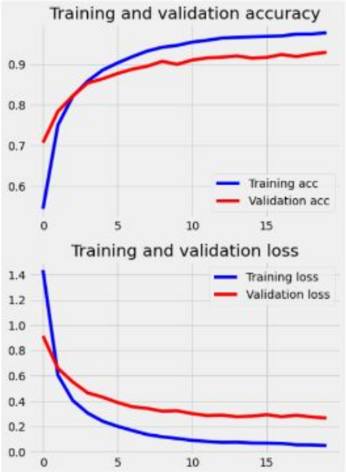
- *RMSprop optimizer*





- *Adam optimizer*



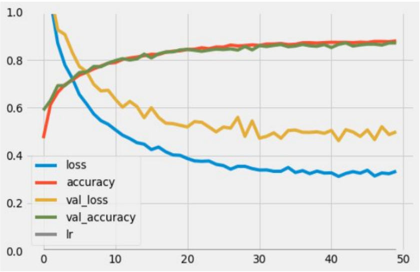


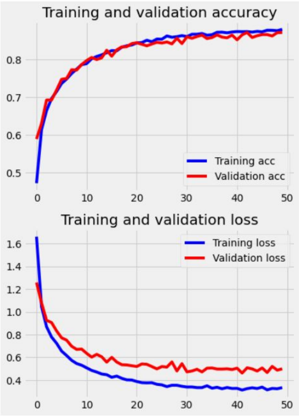

```
1s 5ms/step - loss: 0.0487 - accuracy: 0.9778 - val_loss: 0.2658 - val_accuracy: 0.9297 - lr: 0.0010
```

```
1s 7ms/step - loss: 0.0431 - accuracy: 0.9808 - val_loss: 0.3584 - val_accuracy: 0.9251 - lr: 0.0010
```

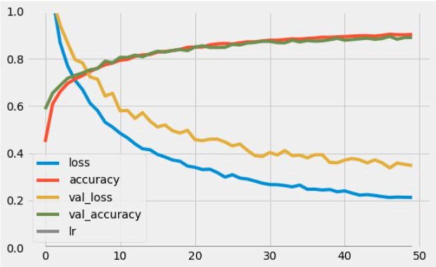
Performances – Feature Extraction with Data Augmentation: training

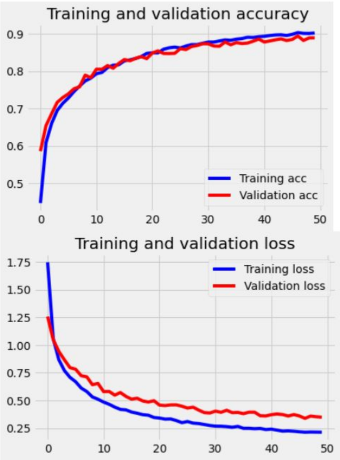
- *RMSprop optimizer*





- *Adam optimizer*





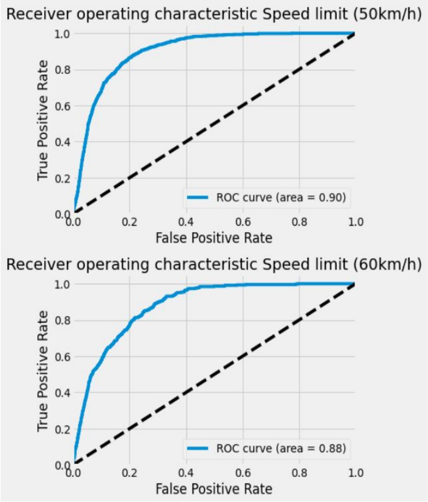
loss: 0.3338 - accuracy: 0.8794 - val_loss: 0.4988 - val_accuracy: 0.8711 - lr: 0.0010

- loss: 0.2131 - accuracy: 0.9023 - val_loss: 0.3480 - val_accuracy: 0.8893 - lr: 0.0010

Performances – Feature Extraction with Data Augmentation: testing

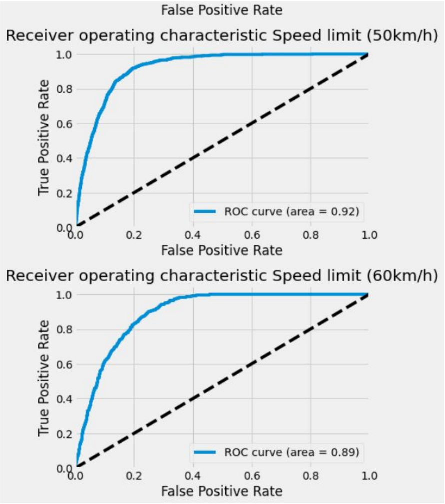
- *RMSprop optimizer*

```
395/395 [==============================] - 6s 15ms/step
Test Data accuracy:  54.53681710213777
```



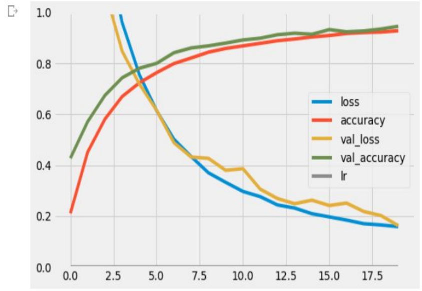
- *Adam optimizer*

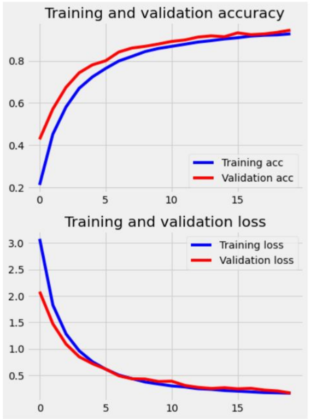
```
395/395 [==============================] - 6s 15ms/step
Test Data accuracy:  53.34125098970704
```



Performances – Fine Tuning: training

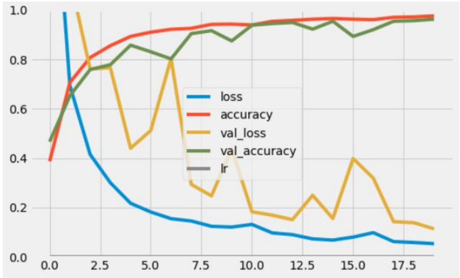
- *RMSprop optimizer*

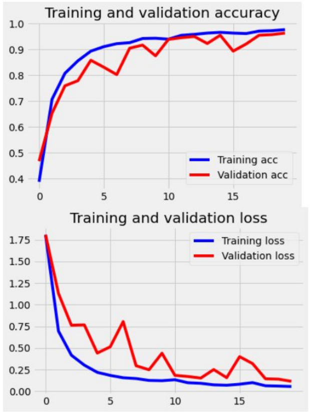




loss: 0.1591 - accuracy: 0.9277 - val_loss: 0.1630 - val_accuracy: 0.9452 - lr: 1.0000e-05

- *Adam optimizer*



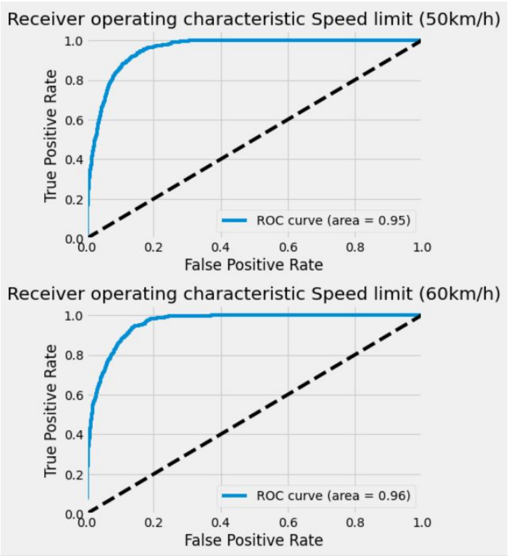


- loss: 0.0541 - accuracy: 0.9763 - val_loss: 0.1141 - val_accuracy: 0.9629 - lr: 0.0010

Performances – Fine Tuning: testing

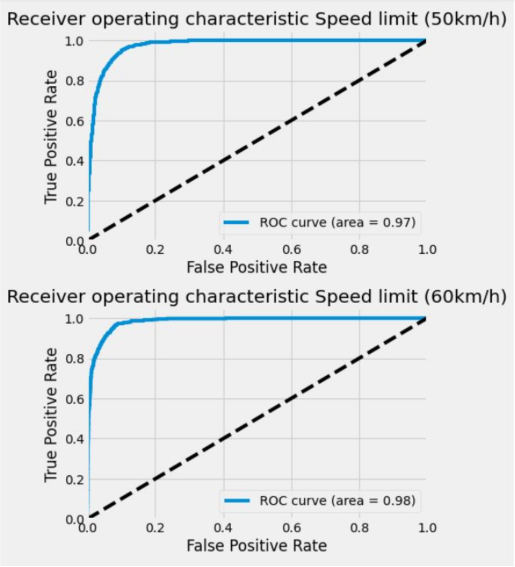
- *RMSprop optimizer*

```
395/395 [==============================] - 12s 28ms/step
Test Data accuracy:  76.70625494853523
```



- *Adam optimizer*

```
395/395 [==============================] - 6s 15ms/step
Test Data accuracy:  82.32779097387174
```



Comparison between the different approaches and conclusions

- Highest training accuracy with the fast feature extraction approach
- However, it was the most sensitive to overfitting
- With VGG19, the fine tuning with the Adam optimizer gave the best results
- Overall, the best approaches we tried for this task are the ones with training from scratch the AlexNet and the Mnist based architecture models

Possible improvements:

- Try different pre-trained models
- Try a different test set or a different data augmentation layer
- Improve hyperparameters
- Explore other existing techniques to try to carry out this task