2020/2021

TRAFFIC SIGN CLASSIFICATION

USING CONVOLUTION NEURAL NETWORK

MARWA AL AMINE / ZEINAB KASSIR
PRESENTED TO: DR MOHAMMAD AOUDE
Mini-project

Table of Contents

Introduction

Every country has some standards set for the design of different traffic signs like U-turn, Left-turn, Right-turn, No-entry, etc. Traffic sign recognition is the process of automatically identifying which of the following class the sign belongs to. The earlier Computer Vision techniques required lots of hard work in data processing and it took a lot of time to manually extract the features of the image. Now, deep learning techniques have come to the rescue and today we will see how to build a traffic recognition system for autonomous vehicles.

The report is divided into several sections. Section1 will overview a general knowledge on Neural Networks especially the Convolutional neural networks. Then Section2 will dive into the training part of the machine explaining some theories and implementations. It will also explain some requirements to our code. Thereafter, Section 3 will feature the training part, and the accuracy plots, showing the final results as well. Final section discusses those results and the importance of this project.

Chapter 1: Convolution Neural Network:

1.1. Artificial Neural Networks (ANNs)

Artificial neural networks (ANNs) are a family of machine learning models inspired by biological neural networks.

Artificial Neural Networks vs. Biological Neural Networks

A basic model for how the neurons work goes as follows: Each synapse has a strength that is learnable and control the strength of influence of one neuron on another. The dendrites carry the signals to the target neuron's body where they get summed. If the final sum is above a certain threshold, the neuron gets fired, sending a spike along its axon.

Artificial neurons are inspired by biological neurons, and try to formulate the model explained above in a computational form. An artificial neuron has a finite number of inputs with weights associated to them, and an activation function (also called transfer function). The output of the neuron is the result of the activation function applied to the weighted sum of inputs. Artificial neurons are connected with each other to form artificial neural networks.

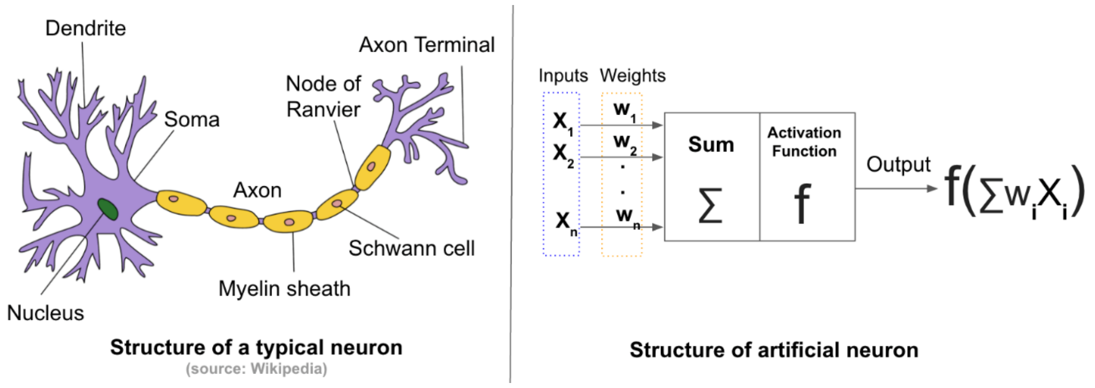


Figure -1-

1.2. Feedforward Neural Networks

Feedforward Neural Networks are the simplest form of Artificial Neural Networks.

These networks have 3 types of layers: Input layer, hidden layer and output layer. In these networks, data moves from the input layer through the hidden nodes (if any) and to the output nodes.

Below is an example of a fully-connected feedforward neural network with 2 hidden layers. "Fully-connected" means that each node is connected to all the nodes in the next layer.

Note that, the number of hidden layers and their size are the only free parameters. The larger and deeper the hidden layers, the more complex patterns we can model in theory.

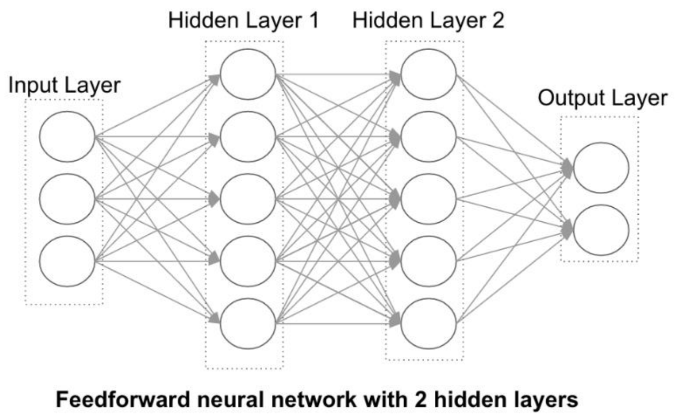


Figure-2-

1.3. Activation Functions

Activation functions transform the weighted sum of inputs that goes into the artificial neurons. These functions should be non-linear to encode complex patterns of the data. The most popular activation functions are Sigmoid, Tanh and ReLU. ReLU is the most popular activation function in deep neural networks.

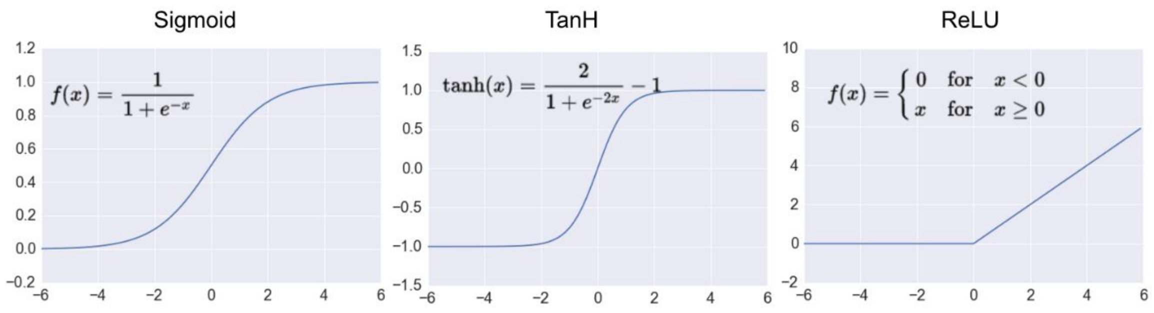


Figure-3-

1.4. Training Artificial Neural Networks

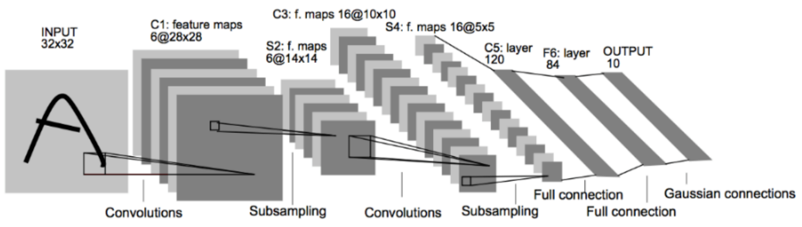
The goal of the training phase is to learn the network's weights. We need 2 elements to train an artificial neural network:

Training data: In the case of image classification, the training data is composed of images and the corresponding labels.

Loss function: A function that measures the inaccuracy of predictions.

1.5. Convolutional Neural Networks (CNNs or ConvNets)

Convolutional neural networks are a special type of feed-forward networks. These models are designed to emulate the behavior of a visual cortex. CNNs perform very well on visual recognition tasks. CNNs have special layers called convolutional layers and pooling layers that allow the network to encode certain images properties.

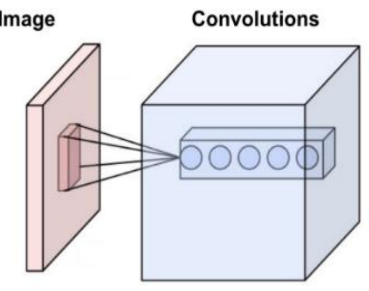


CNN called LeNet by Yann LeCun (1998)

Figure-4-

Convolution Layer

This layer consists of a set of learnable filters that we slide over the image spatially, computing dot products between the entries of the filter and the input image. The filters should extend to the full depth of the input image. For example, if we want to apply a filter of size 5x5 to a colored image of size 30x30, then the filter should have depth 3 (5x5x3) to cover all 3 color channels (Red, Green, Blue) of the image. These filters will activate when they see same specific structure in the images.



Neurons of a convolutional layer, connected to their receptive field
(source: Wikipedia)

Figure-5-

Activation

Without going into further details, we will use ReLU activation function that returns 0 for every negative value in the input image while it returns the same value for every positive value. Shape remains unchanged, it’s still [30x30x12**]**

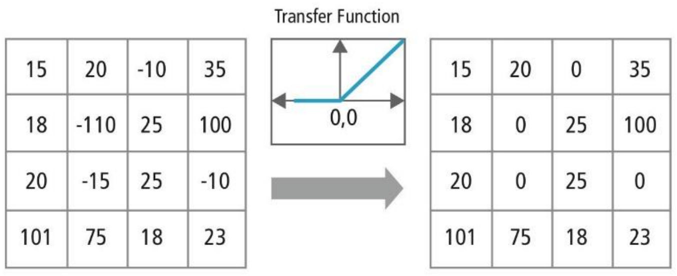


Figure-6- RELU Activation function

Pooling Layer

Pooling is a form of non-linear down-sampling. The goal of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. There are several functions to implement pooling among which max pooling is the most common one. Pooling is often applied with filters of size 2x2 applied with a stride of 2 at every depth slice. A pooling layer of size 2x2 with stride of 2 shrinks the input image to a 1/4 of its original size.

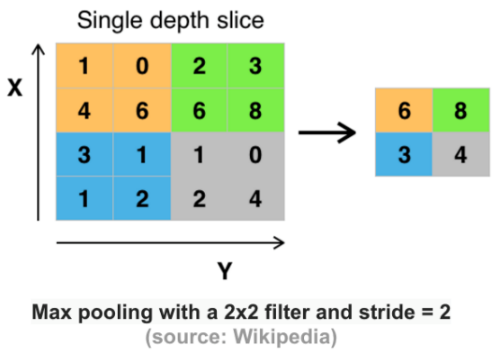


Figure-7-

Flattening

It is converting the data into a 1-dimensional array for inputting it to the next layer. We **flatten** the output of the convolutional layers to create a single long feature vector.And it is connected to the final classification model, which is called a fully-connected layer.

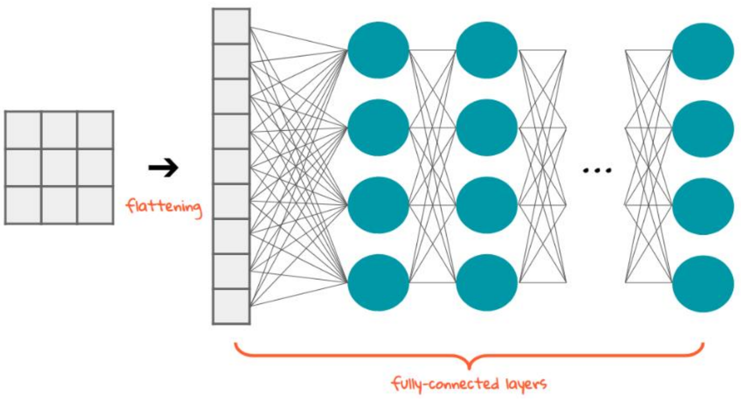


Figure-8-

Convolutional Neural Networks Architecture

The simplest architecture of a convolutional neural networks starts with an input layer (images) followed by a sequence of convolutional layers and pooling layers, and ends with fully-connected layers. The convolutional layers are usually followed by one layer of ReLU activation functions.

The convolutional, pooling and ReLU layers act as learnable features extractors, while the fully connected layers acts as a machine learning classifier. Furthermore, the early layers of the network encode generic patterns of the images, while later layers encode the details patterns of the images.

Note that only the convolutional layers and fully-connected layers have weights. These weights are learned in the training phase.

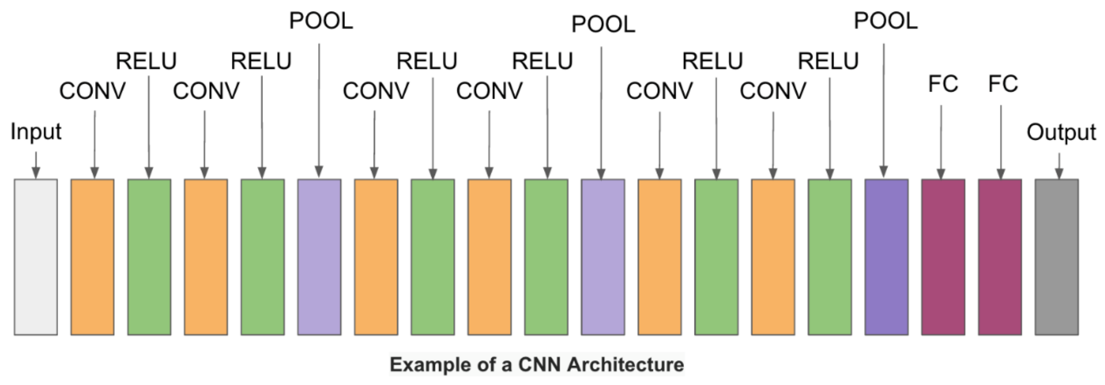


Figure-9-

1.6. Conclusion

In a nutshell, convolutional neural networks use learnable filters in its different layers; the convolutional and pooling layers. The input image will go through more than one layer in order for the cnn to extract a feature from it. The output of those layers will then be fully connected, we will be able to have a good and accurate decision.

Chapter 2: Machine Training; Overview and Coding

2.1. Introduction

Our machine can learn how to recognize and predict an image when loaded. That may sound weird, however as explained previously, deep learning is our go-to plan. The first thing a machine will need to start deep learning is a good dataset. Then a model must be constructed, with the help of different functions and optimized.

2.2. Dataset

Definition: a collection of related sets of information that is composed of separate elements but can be manipulated as a unit by a computer.

The dataset we have used for this project is the GTSRB (German traffic sign recognition benchmark). It contains a Train folder that has traffic sign images in 43 different classes, a Test folder that has over 12,000 images for testing purposes. A test.csv file that contains the path of the test images along with their respective classes.

The next step is to go over the whole set and load image by image, with doing frame adjustments to each; we try to feed the requirements of our CNN.

2.3. Create the data:

An image is made up of pixels and each pixel has 3 values to specify its color i.e. RGB. In order for machines to understand the image, we have to convert the image into numbers. For this purpose, we use the PIL library that can perform many image manipulation tasks. If you have observed clearly then you will see that the images are of different width and heights. So we also have to resize all the images to a fixed size like 30x30.

Define "Data" as a list and add the modified images to it. We also define with the data another set and put our pre-defined categories in it and name that list as "Label"

2.4. Constructing the Model

The Convolutional Neural Networks have proved the state of the art in image classification tasks and this is what we will be using for our model. A Convolutional Neural Network(CNN) is made up of convolutional and pooling layers. At each layer, the features from the image are extracted that helps in classifying the image.

We have also used the dropout layer which is used to handle the overfitting of the model. The dropout layer drops some of the neurons while training but not when we are predicting. We compile the model with categorical_crossentropy because our dataset has multi classes to be classified.

2.5. Compile our model:

In order to reduce the losses and increase the accuracy, it is best to apply an optimizer on the model. One of the best optimizers in the image processing domain is Adam optimizer.

Optimization algorithms or strategies are responsible for reducing the losses and to provide the most accurate results possible. Adam is different to classical stochastic gradient descent. Stochastic gradient descent maintains a single learning rate for all weight updates and the learning rate does not change during training. A learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds.

Adam’s method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

2.6. Train the head of the network

Train our data for a certain number of epochs; in our project we chose 15 epochs. One Epoch is basically when an ENTIRE dataset is passed once through the forward and backward of the neural network. If the Epoch is too big to feed to the computer at once, you will then need to divide it into several smaller batches.

Note that we save our model in the format of h5.

2.7. Plot the accuracy graph

With the help of matplotlib functions, we will plot the graph of training and validation accuracy.

```
plt.plot(history.history[‘accuracy’], label=’training accuracy’)
plt.plot(history.history[‘val_accuracy’], label=’val accuracy’)
```

The accuracy we received is 0.948 and the validation_accuracy is 0.98.

2.8. Conclusion

In this chapter we illustrated some machine training necessities in coding. We expose the need of a good dataset that is large enough to help build a good model. Furthermore, we explained how the data must be processed and used to finalize the model.

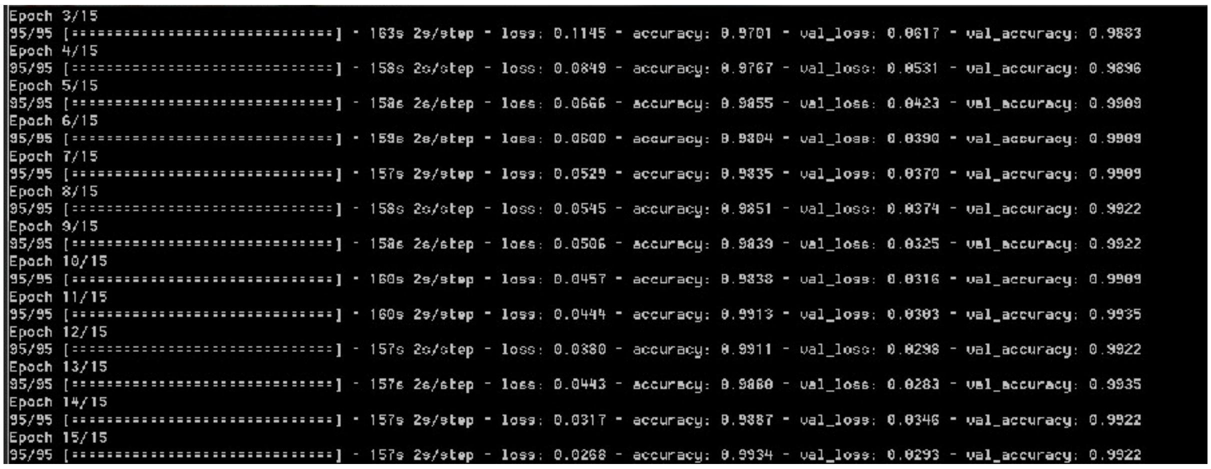
Chapter 3: Project Results

3.1. Introduction

In this chapter, we will introduce the results of the training phase on Machine. We will analize the accuracy and losses of training with the help of a plot. The final result will be visualized on the livestream video by model applying.

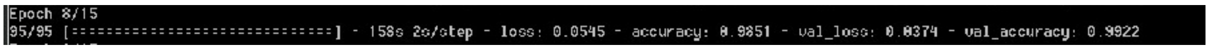
3.2. Machine Training

We generate output predictions for the input samples. We use as explained 15 epochs. This means we passed the entire data 15 times forward and backwards through the neural network. Each epoch does some calculations.



```
Epoch 3/15
95/95 [==============================] - 163s 2s/step - loss: 0.1145 - accuracy: 0.9701 - val_loss: 0.0617 - val_accuracy: 0.9883
Epoch 4/15
95/95 [==============================] - 158s 2s/step - loss: 0.0849 - accuracy: 0.9767 - val_loss: 0.0531 - val_accuracy: 0.9896
Epoch 5/15
95/95 [==============================] - 158s 2s/step - loss: 0.0666 - accuracy: 0.9855 - val_loss: 0.0423 - val_accuracy: 0.9909
Epoch 6/15
95/95 [==============================] - 159s 2s/step - loss: 0.0600 - accuracy: 0.9804 - val_loss: 0.0390 - val_accuracy: 0.9909
Epoch 7/15
95/95 [==============================] - 157s 2s/step - loss: 0.0529 - accuracy: 0.9835 - val_loss: 0.0370 - val_accuracy: 0.9909
Epoch 8/15
95/95 [==============================] - 158s 2s/step - loss: 0.0545 - accuracy: 0.9851 - val_loss: 0.0374 - val_accuracy: 0.9922
Epoch 9/15
95/95 [==============================] - 158s 2s/step - loss: 0.0506 - accuracy: 0.9839 - val_loss: 0.0325 - val_accuracy: 0.9922
Epoch 10/15
95/95 [==============================] - 160s 2s/step - loss: 0.0457 - accuracy: 0.9838 - val_loss: 0.0316 - val_accuracy: 0.9909
Epoch 11/15
95/95 [==============================] - 160s 2s/step - loss: 0.0444 - accuracy: 0.9913 - val_loss: 0.0303 - val_accuracy: 0.9935
Epoch 12/15
95/95 [==============================] - 157s 2s/step - loss: 0.0380 - accuracy: 0.9911 - val_loss: 0.0298 - val_accuracy: 0.9922
Epoch 13/15
95/95 [==============================] - 157s 2s/step - loss: 0.0443 - accuracy: 0.9860 - val_loss: 0.0283 - val_accuracy: 0.9935
Epoch 14/15
95/95 [==============================] - 157s 2s/step - loss: 0.0317 - accuracy: 0.9887 - val_loss: 0.0346 - val_accuracy: 0.9922
Epoch 15/15
95/95 [==============================] - 157s 2s/step - loss: 0.0268 - accuracy: 0.9934 - val_loss: 0.0293 - val_accuracy: 0.9922
```

Figure-10- a screenshot of training phase



```
Epoch 8/15
95/95 [==============================] - 158s 2s/step - loss: 0.0545 - accuracy: 0.9851 - val_loss: 0.0374 - val_accuracy: 0.9922
```

Figure-11-

The epoch gives a value for the loss; a scalar value that we attempt to minimize during our training of the model the lower the loss the closer our predictions are to the true labels. Loss of 0.0545 as an example. Validation loss is also computed. This should be less than the loss value.

It also provides knowledge for the accuracy, 0.9851 in this case which is pretty high. Notice also the validation accuracy is evaluated; 0.9922 in epoch 8.

3.3. Plotting Results

We then use MATPLOTLIB to plot the results and analyze the results. (we commanded the program to plot this while training)

As the plot shows, we have a high accuracy that reaches at the end of training an approximate value of 100% verifying the good model generation.

Not only did the model give high accuracy, but also the losses are significantly very low.

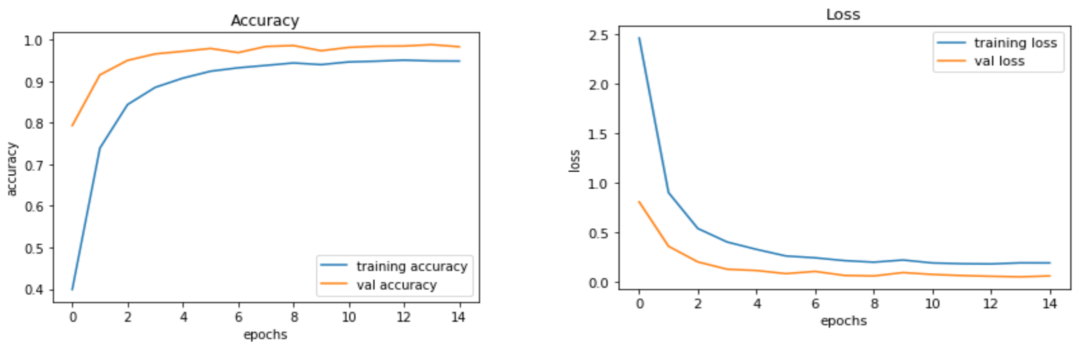


Figure-12-

3.4. Building a Graphical User Interface

Now let’s take a step ahead and build a nice graphical user interface for our deep learning model. A graphical user interface will save a lot of time in testing and seeing the results of our model prediction. The Tkinter is an inbuilt library of python to make a graphical user interface.

From the interface of the GUI application, we will ask the user for an image and extract the file path of the image. Then we use the trained model that will take the image data as input and provide us the class our image belongs to. We will then use the dictionary to see the name of the class. Create a new python file, you can name it as traffic_gui.py

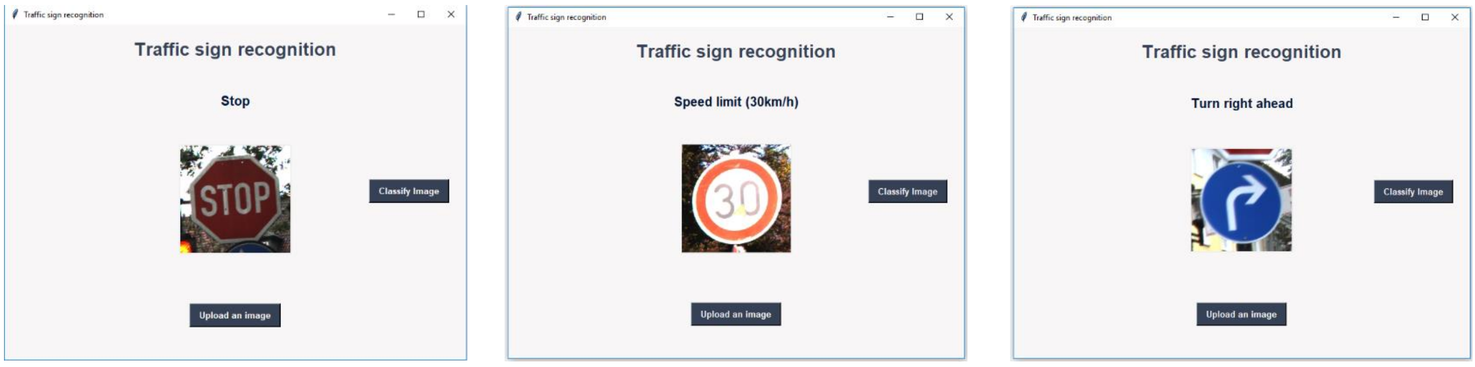


Figure -13-

3.5. Conclusion

We first train the machine, and get each epoch results of loss and accuracy. The plot generated gives an idea of the whole process’ accuracy and how were the data lost.

Chapter 4: Project Discussion

4.1. Introduction

Generally speaking, there are a lot of factors that plays a role in affecting the efficiency of a model. These factors must be taken into consideration. In this chapter we will discuss some of them, like splitting and testing of datasets.

4.2. Train-Test-Split

The train-test split procedure is appropriate when you have a very large dataset, a costly model to train, or require a good estimate of model performance quickly.

The train-test split is a technique for evaluating the performance of a machine learning algorithm.

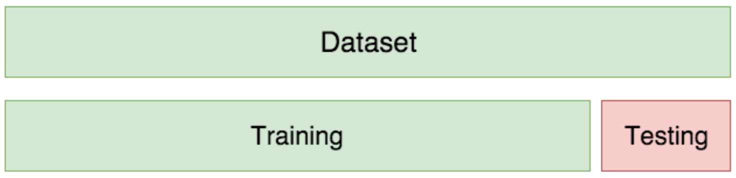
It can be used for classification or regression problems and can be used for any supervised learning algorithm.

The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model;

instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

- **Train Dataset**: Used to fit the machine learning model.
- **Test Dataset**: Used to evaluate the fit machine learning model.
- The objective is to estimate the performance of the machine learning model on new data: data not used to train the model.

Figure-14-



What is the difference between loss and validation loss? And why are both losses computed? Loss reflects the performance of the model with the training dataset. On the other hand, the validation loss gives an idea on how the generated model will perform with new, not previously seen, data. It is quite important to have a validation loss less than the loss.

If we computed a low loss, but the validation loss was larger, this will mean that our model over-fit the training data; the model in this case behaves more like a memory than an artificial intelligent circuit.

For instance, to reduce the validation loss, we can perform pooling that reduces parameters and computations to learn (as mentioned in chapter 1). Additionally, a large dataset is a good method that helps in reducing the validation losses.

Indeed, our model has satisfied these conditions proving to be an efficient, good built model; our validation loss was less than the loss as shown in the plot in chapter 3.

4.3. Conclusion

On the whole, the model built in this project has satisfied a lot of requirement to be an efficient one. Just as test-and-split process helped in this achievement. Also, our chosen dataset with the quite a large number of images of traffic signs.

General Conclusion

In a nutshell, our world is evolving and is depending more and more on machines. Today we are setting the stone to a future where automated cars are common everywhere. And the streets' rules depend a lot on the traffic signs that are everywhere.

With the help of machine learning, we were able to design a model that can classify any traffic sign uploaded to it. The model is found to be accurate and precise. Nonetheless, we need to keep enhancing and improving our projects to have full satisfaction in the future. For instance, we can work on capturing the traffic sign and recognizing it in a live-stream video. Indeed, one must always think of how the future is really promising.