

“TRAFFIC SIGNS RECOGNITION SYSTEM”

J Component Project Report for the course

CSE3017 Computer Vision

By

Bhavyajot Malhotra (17BLC1155)

Vaibhav (17BLC1033)

Sarthak Parthasarathi (17BLC1034)

Submitted to

Dr. R.VIJAYRAJAN



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

**SCHOOL OF ELECTRONICS ENGINEERING
VELLORE INSTITUTE OF TECHNOLOGY
CHENNAI - 600127**

November 2020

CERTIFICATE

This is to certify that the Project work titled “Traffic Signs Recognition System” is being submitted by **Bhavyajot Malhotra (17BLC1155)**, **Vaibhav (17BLC1033)** and **Sarthak Parthasarathi (17BLC1034)** for the course **CSE3017 Computer Vision**, is a record of bonafide work done under my guidance. The contents of this project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University.

Prof. VIJAYRAJAN R.

Guide

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Prof VIJAYARAJAN R**, School of Electronics Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Sivasubramanian. A**, Dean of the Schools of Electrical Engineering (SELECT) and Electronics Engineering (SENSE), VIT University Chennai, for extending the facilities of the School towards our project and for her unstinting support.

We express our thanks to our **Head of The Department: Dr. D Thiripurasundari - B.Tech ECM**, for her support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and wisdom imparted to us throughout the course.

Bhavyajot Malhotra (17BLC1155)
Vaibhav (17BLC1033)
Sarthak Parthasarathi (17BLC1034)

TABLE OF CONTENTS

Topic		Page no.
Bonafide Certificate		2
Acknowledgement		3
Chapter – 1 : Introduction		5
1.1	Abstract	5
1.2	Objective	6
Chapter – 2 : Design/Implementation		7
2.1	Dataset	7
2.2	Data Pre-processing	9
2.3	Convolutional Neural Networks (CNN)	11
2.4	Developing the Deep Learning Model	16
2.5	Graphical User Interface (GUI)	19
Chapter – 3 : Result and Analysis / Testing		22
Chapter – 4 : Conclusion and Future Enhancements		24
Appendix		25
References		31

CHAPTER 1

INTRODUCTION

ABSTRACT

One of the important fields of Artificial Intelligence is Computer Vision. Computer Vision is the science of computers and software systems that can recognize and understand images and scenes. Computer Vision is also composed of various aspects such as image recognition, object detection, image generation, image super-resolution and more. Object detection is probably the most profound aspect of computer vision due the number practical use cases.

Object detection refers to the capability of computer and software systems to locate objects in an image/scene and identify each object. Object detection has been widely used for face detection, vehicle detection, pedestrian counting, web images, security systems and driverless cars. There are many ways object detection can be used as well in many fields of practice. Like every other computer technology, a wide range of creative and amazing uses of object detection will definitely come from the efforts of computer programmers and software developers.

Getting to use modern object detection methods in applications and systems, as well as building new applications based on these methods is not a straight forward task. Early implementations of object detection involved the use of 4 classical algorithms, like the ones supported in OpenCV, the popular computer vision library. However, these classical algorithms could not achieve enough performance to work under different conditions.

The breakthrough and rapid adoption of deep learning in 2012 brought into existence modern and highly accurate object detection algorithms and methods such as R-CNN, Fast-RCNN, Faster-RCNN, RetinaNet and fast yet highly accurate ones like SSD and YOLO. Using these methods and algorithms, based on deep learning which is also based on machine learning require lots of mathematical and deep learning frameworks understanding.

In our project we are making specific object detection model which will detect the traffic signs and tell us what the sign means in a custom built GUI.

OBJECTIVE

In the world of Artificial Intelligence and advancement in technologies, many researchers and big companies like Tesla, Uber, Google, Mercedes-Benz, Toyota, Ford, Audi, etc. are working on autonomous vehicles and self-driving cars. So, for achieving accuracy in this technology, the vehicles should be able to interpret traffic signs and make decisions accordingly.

There are several different types of traffic signs like speed limits, no entry, traffic signals, turn left or right, children crossing, no passing of heavy vehicles, etc. Traffic signs classification is the process of identifying which class a traffic sign belongs to.

We will build a deep neural network model that can classify traffic signs present in the image into different categories. With this model, we are able to read and understand traffic signs which are a very important task for all autonomous vehicles.

Also we will build a custom graphical user interface which will allow anyone to upload an image and classify it into any of the traffic signs using our own saved neural network model.

CHAPTER 2

DESIGN/IMPLEMENTATION

DATASET

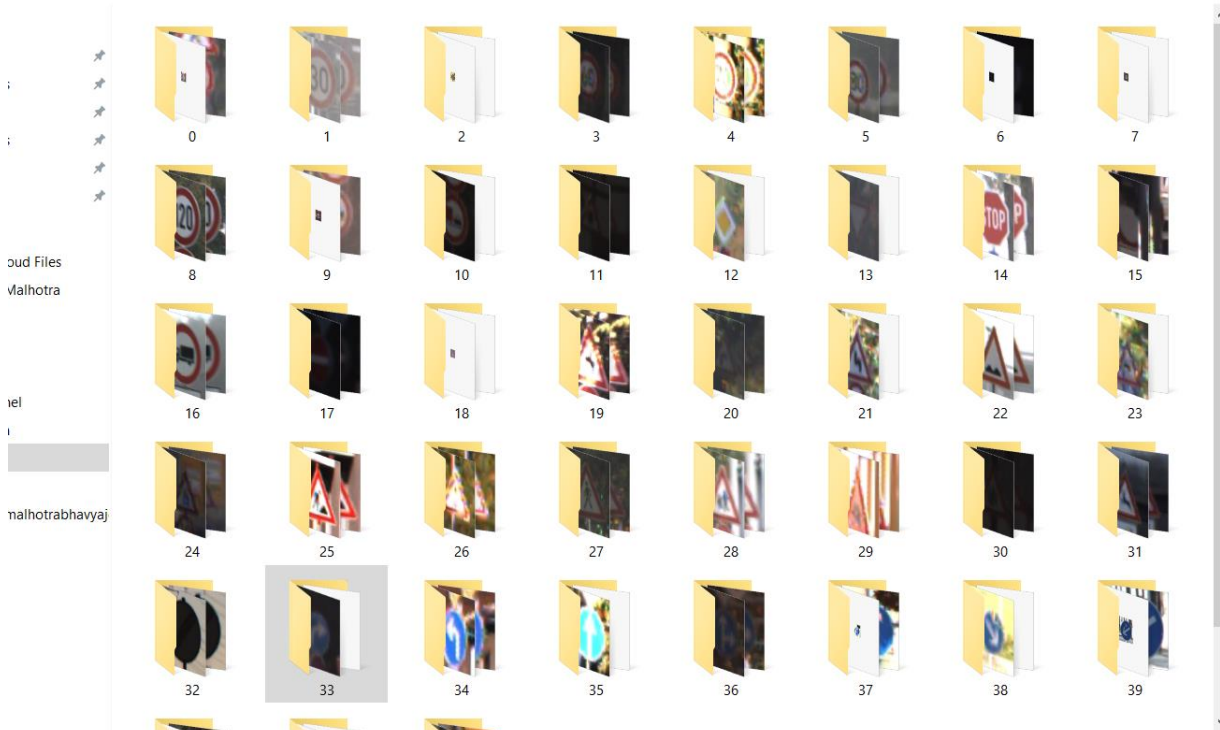
For this project, we are using the public dataset available at Kaggle: [Traffic Signs Dataset](#)

The dataset contains more than 50,000 images of different traffic signs. It is further classified into 43 different classes. The dataset is quite varying, some of the classes have many images while some classes have few images. The size of the dataset is around 300 MB. The dataset has a train folder which contains images inside each class and a test folder which we will use for testing our model.

PROJECT > Traffic_sign_classification >

Name	Date modified	Type	Size
.ipynb_checkpoints	26-10-2020 01:12 PM	File folder	
Meta	27-07-2020 02:09 PM	File folder	
Test	01-08-2020 01:50 PM	File folder	
Train	27-07-2020 02:07 PM	File folder	
Classes.txt	26-10-2020 01:03 PM	Text Document	2 KB
GUL.ipynb	26-10-2020 01:23 PM	IPYNB File	4 KB
gui.py	30-12-2019 01:28 AM	PY File	4 KB
Layers.txt	11-09-2020 10:59 PM	Text Document	1 KB
Meta.csv	13-10-2019 05:49 AM	Microsoft Excel Co...	2 KB
Sign Recog.ipynb	03-09-2020 11:30 PM	IPYNB File	45 KB
Sign Recog-1.ipynb	26-10-2020 01:41 PM	IPYNB File	20 KB
Sign Recog-Copy1.ipynb	27-10-2020 07:46 PM	IPYNB File	46 KB
Test.csv	13-10-2019 05:49 AM	Microsoft Excel Co...	418 KB
traffic.h5	27-10-2020 07:45 PM	H5 File	2,901 KB
traffic_classifier.h5	30-12-2019 01:28 AM	H5 File	2,890 KB
traffic_sign.py	30-12-2019 01:28 AM	PY File	4 KB
Train.csv	13-10-2019 05:50 AM	Microsoft Excel Co...	1,896 KB

Our 'train' folder contains 43 folders each representing a different class. The range of the folder is from 0 to 42. With the help of the OS module, we iterate over all the classes and append images and their respective labels in the data and labels list.



#Dictionary to label all traffic signs class.

```
classes = { 1:'Speed limit (20km/h)',
            2:'Speed limit (30km/h)',
            3:'Speed limit (50km/h)',
            4:'Speed limit (60km/h)',
            5:'Speed limit (70km/h)',
            6:'Speed limit (80km/h)',
            7:'End of speed limit (80km/h)',
            8:'Speed limit (100km/h)',
            9:'Speed limit (120km/h)',
            10:'No passing',
            11:'No passing veh over 3.5 tons',
            12:'Right-of-way at intersection',
            13:'Priority road',
            14:'Yield',
            15:'Stop',
            16:'No vehicles',
            17:'Veh > 3.5 tons prohibited',
            18:'No entry',
            19:'General caution',
            20:'Dangerous curve left',
            21:'Dangerous curve right',
            22:'Double curve',
            23:'Bumpy road',
            24:'Slippery road',
            25:'Road narrows on the right',
```


DATA PRE-PROCESSING

The PIL library is used to open image content into an array.

```
[9]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
```

Finally, we have stored all the images and their labels into lists (data and labels).

```
In [2]: #Retrieving the images and their labels
for i in range(classes):
    path = os.path.join(cur_path, 'train', str(i))
    images = os.listdir(path)
    for a in images:
        try:
            image = Image.open(path + '\\' + a)
            image = image.resize((30,30))
            image = np.array(image)
            data.append(image)
            labels.append(i)
        except:
            print("Error loading image")
```

Our dataset contains a train folder and in a train.csv file, we have the details related to the image path and their respective class labels. We extract the image path and labels using pandas. Then to predict the model, we have to resize our images to 30×30 pixels and make a numpy array containing all image data.

We need to convert the list into numpy arrays for feeding to the model.

In [3]: *#Converting lists into numpy arrays*

```
data = np.array(data)
labels = np.array(labels)
print(data.shape, labels.shape)
```

```
(39209, 30, 30, 3) (39209,)
```

The shape of data is (39209, 30, 30, 3) which means that there are 39,209 images of size 30×30 pixels and the last 3 means the data contains coloured images (RGB value).

In [4]: *#Splitting training and testing dataset*

```
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)
```

In [5]: *#Converting the labels into one hot encoding*

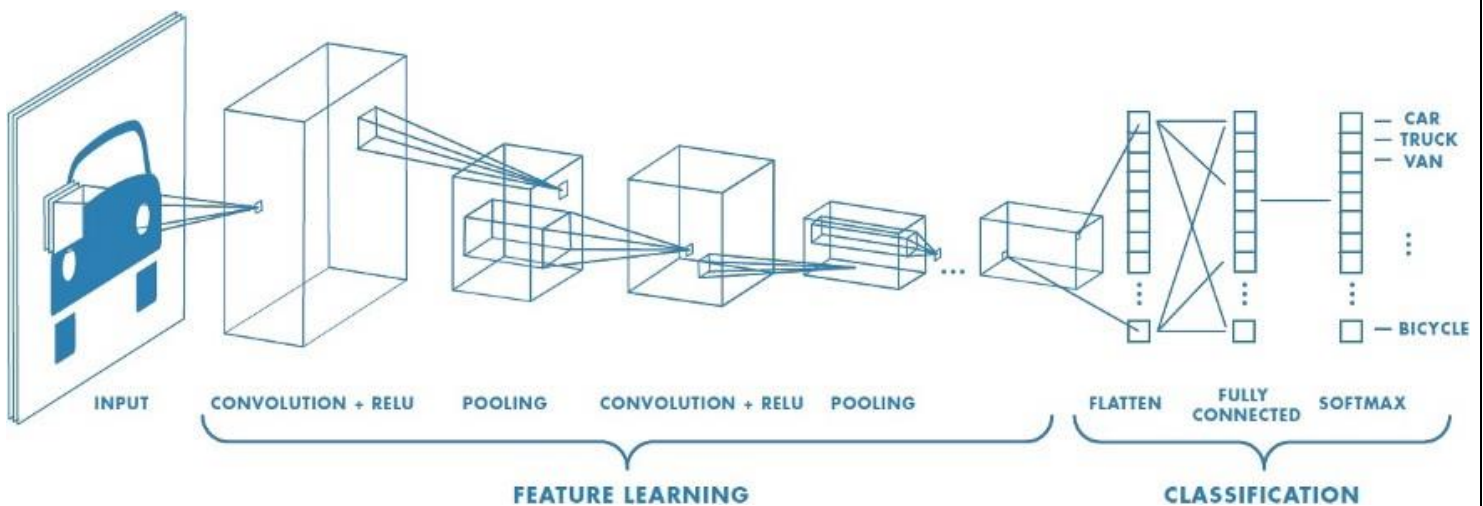
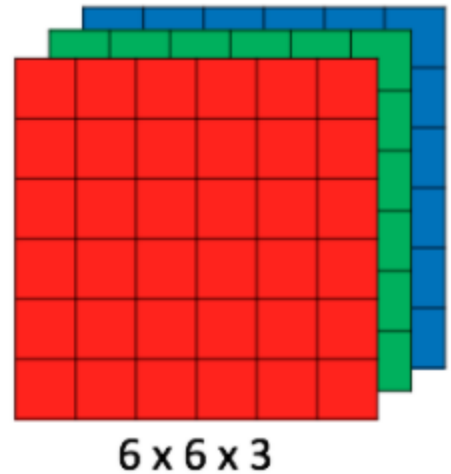
```
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)
```

With the sklearn package, we use the `train_test_split()` method to split training and testing data. From the `keras.utils` package, we use `to_categorical` method to convert the labels present in `y_train` and `y_test` into one-hot encoding.

CONVOLUTIONAL NEURAL NETWORKS (CNN)

CNN image classifications takes an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion). Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see $h \times w \times d$ (h = Height, w = Width, d = Dimension). Eg. An image of $6 \times 6 \times 3$ array of matrix of RGB (3 refers to RGB values) and an image of $4 \times 4 \times 1$ array of matrix of grayscale image.

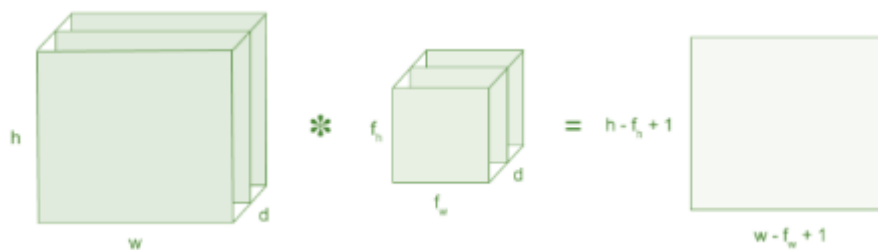
Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernels), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.



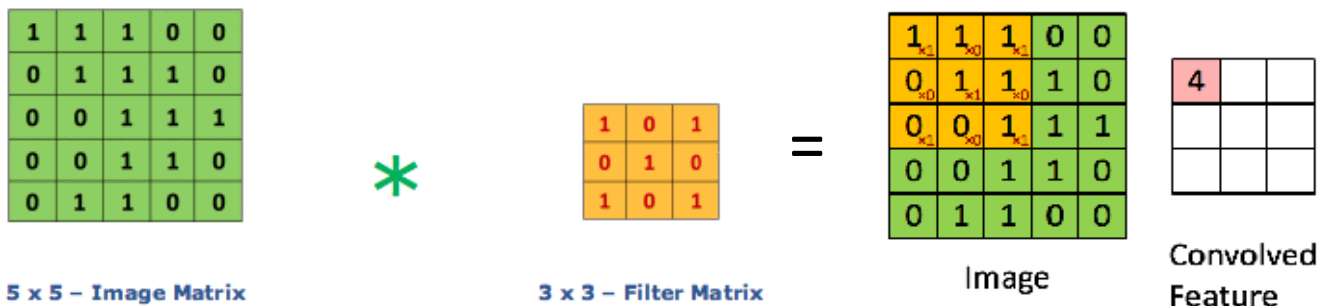
Convolution Layer

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f_h x f_w x d)**
- Outputs a volume dimension **(h - f_h + 1) x (w - f_w + 1) x 1**



Consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3 x 3 as shown in below



Then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called “**Feature Map**” as output shown. Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. The below example shows various convolution image after applying different types of filters (Kernels).








Strides

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.

Padding

Sometimes filter does not fit perfectly fit the input image. We have two options:

- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Non Linearity (ReLU)

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is $f(x) = \max(0, x)$. Why ReLU is important: ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.

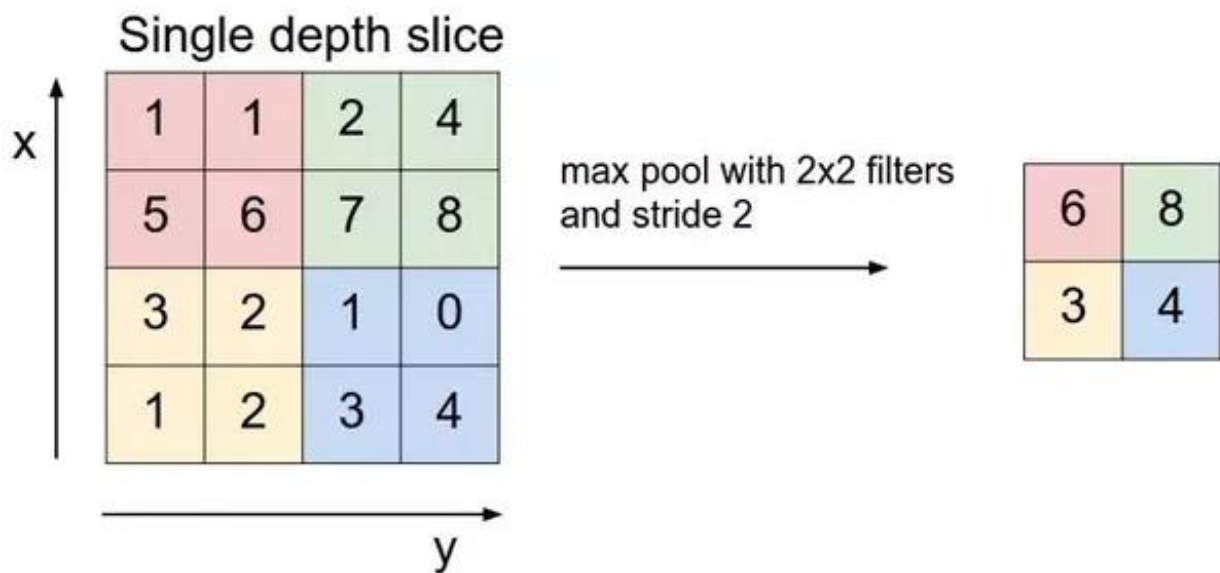
There are other nonlinear functions such as tanh or sigmoid that can also be used instead of ReLU. Most of the data scientists use ReLU since performance wise ReLU is better than the other two.

Pooling Layer

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or down sampling which reduces the dimensionality of each map but retains important information. Spatial pooling can be of different types:

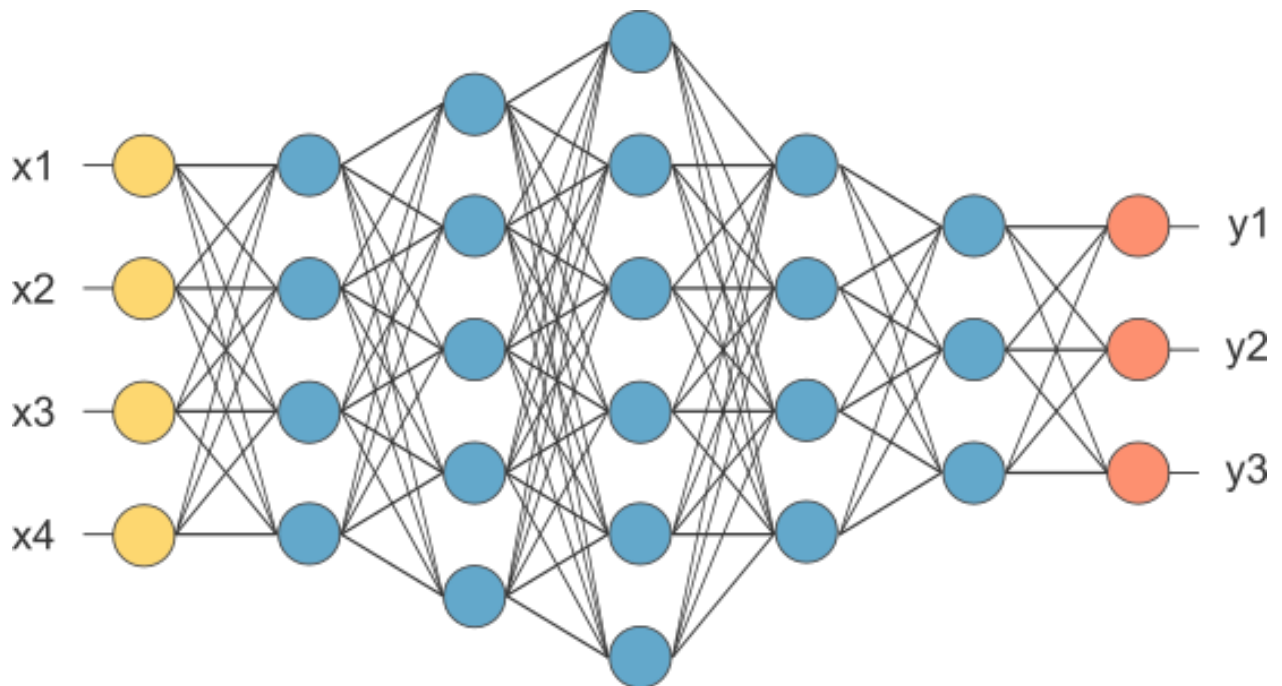
- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

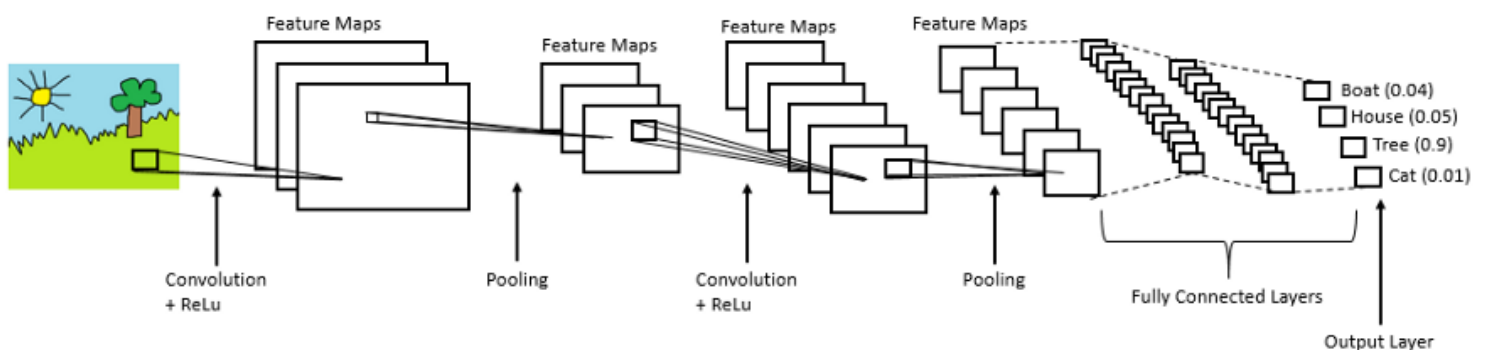


Fully Connected Layer

The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network.



In the above diagram, the feature map matrix will be converted as vector (x_1 , x_2 , x_3 ,). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as softmax or sigmoid to classify the outputs as cat, dog, car, truck etc.



DEVELOPING DEEP LEARNING MODEL

The 6 lines of code below define the convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers. As input, a CNN takes tensors of shape (image_height, image_width, color_channels), ignoring the batch size. In this we configure our CNN to process inputs of shape (30, 30, 3), which is the format of CIFAR images.

```
In [72]: #Building the model
#Creating a convolutional base

model = Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=X_train.shape[1:]))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Adding Dense layers on top

```
In [73]: #Adding a Dense layers on top
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(43, activation='softmax'))
```

To complete our model, we will feed the last output tensor from the convolutional base (of shape (4, 4, and 64)) into one or more dense layers to perform classification. Dense layers take vectors as input (which are 1D), while the current output is a 3D tensor. First, we will flatten (or unroll) the 3D output to 1D, then add one or more Dense layers on top. CIFAR has 10 output classes, so we used a final dense layer with 10 outputs and a softmax activation.

As you can see, our (4, 4, 64) outputs were flattened into vectors of shape (1024) before going through two Dense layers.

We compile the model with Adam optimizer which performs well and loss is “categorical_crossentropy” because we have multiple classes to categorize. After building the model architecture, we then train the model using `model.fit ()`.

I tried with batch size 32 and 64. Our model performed better with 64 batch size. And after 15 epochs the accuracy was stable.

In the end, we are going to save the model that we have trained using the Keras `model.save()` function.

#Compilation of the model

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
epochs = 15
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))
model.save("my_model.h5")
```

Epoch 1/15

981/981 [=====] - 16s 17ms/step - loss: 1.9982 - accuracy: 0.4944 - val_loss: 0.4041 - val_a
0.9078

Epoch 2/15

981/981 [=====] - 16s 16ms/step - loss: 0.5558 - accuracy: 0.8410 - val_loss: 0.1675 - val_a
0.9543

```
model.summary()
```

Model: "sequential_7"

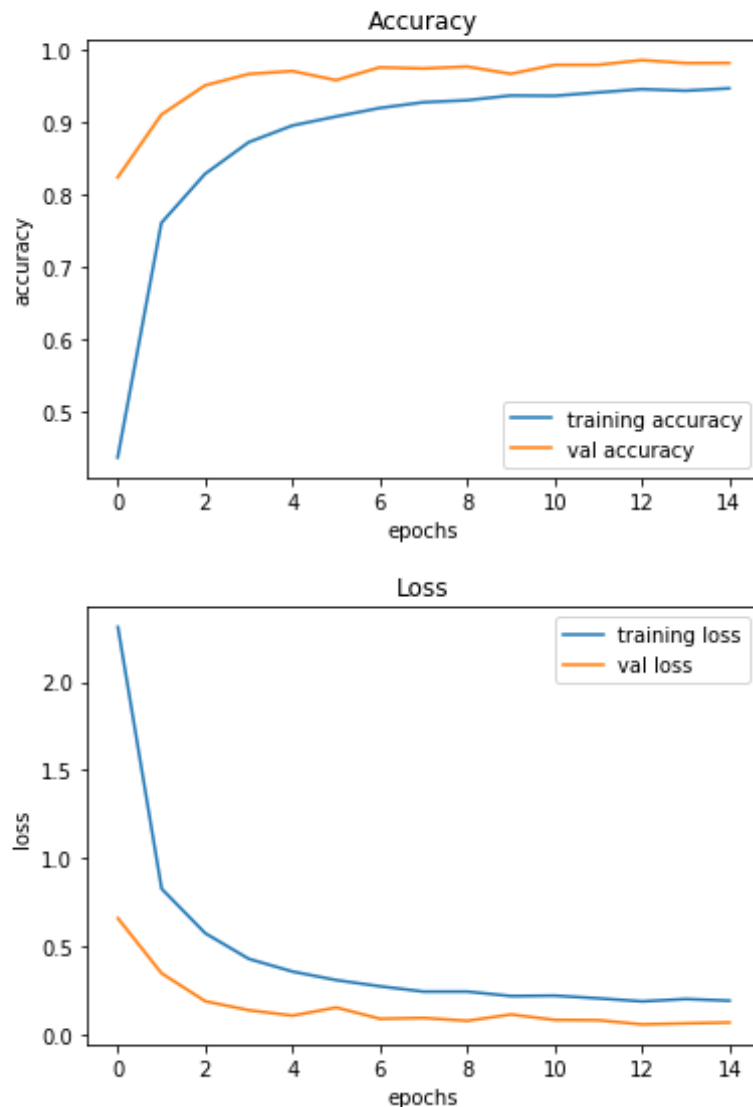
Layer (type)	Output Shape	Param #
conv2d_20 (Conv2D)	(None, 28, 28, 32)	896
max_pooling2d_12 (MaxPooling)	(None, 14, 14, 32)	0
conv2d_21 (Conv2D)	(None, 12, 12, 64)	18496
max_pooling2d_13 (MaxPooling)	(None, 6, 6, 64)	0
conv2d_22 (Conv2D)	(None, 4, 4, 64)	36928
flatten_6 (Flatten)	(None, 1024)	0
dense_12 (Dense)	(None, 256)	262400
dense_13 (Dense)	(None, 43)	11051
Total params: 329,771		
Trainable params: 329,771		
Non-trainable params: 0		

Our model got a 95% accuracy on the training dataset. With matplotlib, we plot the graph for accuracy and the loss.

```
[13]: plt.figure(0)
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()

plt.figure(1)
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title('Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
```

[13]: <matplotlib.legend.Legend at 0x24eece89e48>



GRAPHICAL USER INTERFACE (GUI)

Now we built a graphical user interface for our traffic signs classifier with Tkinter. Tkinter is a GUI toolkit in the standard python library. We have first loaded the trained model 'traffic_classifier.h5' using Keras.

```
In [9]: import tkinter as tk
        from tkinter import filedialog
        from tkinter import *
        from PIL import ImageTk, Image
        import numpy
```

```
In [10]: #Load the trained model to classify sign
        from keras.models import load_model
        model = load_model('traffic_classifier.h5')
```

We use the dictionary to get the information about the class.

```
In [11]: #Dictionary to label all traffic signs class.
        classes = { 1:'Speed limit (20km/h)',
                    2:'Speed limit (30km/h)',
                    3:'Speed limit (50km/h)',
                    4:'Speed limit (60km/h)',
                    5:'Speed limit (70km/h)',
                    6:'Speed limit (80km/h)',
                    7:'End of speed limit (80km/h)',
                    8:'Speed limit (100km/h)',
                    9:'Speed limit (120km/h)',
                    10:'No passing',
                    11:'No passing veh over 3.5 tons',
                    12:'Right-of-way at intersection',
                    13:'Priority road',
                    14:'Yield',
                    15:'Stop',
                    16:'No vehicles',
                    17:'Veh > 3.5 tons prohibited',
                    18:'No entry',
```

And then we build the GUI for uploading the image and a button is used to classify which calls the classify () function.

The `classify ()` function is converting the image into the dimension of shape (1, 30, 30, 3). This is because to predict the traffic sign we have to provide the same dimension we have used when building the model.

```
In [12]: #Initialise GUI
top=tk.Tk()
top.geometry('800x600')
top.title('Traffic sign classification')
top.configure(background='#CDCDCD')
label=Label(top,background='#CDCDCD', font=('arial',15,'bold'))
sign_image = Label(top)
def classify(file_path):
    global label_packed
    image = Image.open(file_path)
    image = image.resize((30,30))
    image = numpy.expand_dims(image, axis=0)
    image = numpy.array(image)
    pred = model.predict_classes([image])[0]
    sign = classes[pred+1]
    print(sign)
    label.configure(foreground='#011638', text=sign)
```

Then we predict the class, the `model.predict_classes (image)` returns us a number between (0-42) which represents the class it belongs to.

```
In [13]: def show_classify_button(file_path):
    classify_b=Button(top,text="Classify Image",command=lambda: classify(file_path),padx=10,pady=5)
    classify_b.configure(background='#364156', foreground='white',font=('arial',10,'bold'))
    classify_b.place(relx=0.79,rely=0.46)
def upload_image():
    try:
        file_path=filedialog.askopenfilename()
        uploaded=Image.open(file_path)
        uploaded.thumbnail(((top.winfo_width()/2.25),(top.winfo_height()/2.25)))
        im=ImageTk.PhotoImage(uploaded)
        sign_image.configure(image=im)
        sign_image.image=im
        label.configure(text='')
        show_classify_button(file_path)
    except:
        pass
upload=Button(top,text="Upload an image",command=upload_image,padx=10,pady=5)
upload.configure(background='#364156', foreground='white',font=('arial',10,'bold'))
upload.pack(side=BOTTOM,pady=50)
sign_image.pack(side=BOTTOM,expand=True)
label.pack(side=BOTTOM,expand=True)
heading = Label(top, text="Classify Traffic Signs",pady=20, font=('arial',20,'bold'))
heading.configure(background='#CDCDCD', foreground='#364156')
heading.pack()
top.mainloop()
```

The final output is shown below in which we can upload any image and it will use our saved trained model to classify the uploaded images into different classes of the traffic signs as per the dictionary.



CHAPTER 3

RESULT AND ANALYSIS / TESTING

Our dataset contains a test folder and in a test.csv file, we have the details related to the image path and their respective class labels. We extract the image path and labels using pandas. Then to predict the model, we have to resize our images to 30×30 pixels and make a numpy array containing all image data. From the sklearn.metrics, we imported the accuracy_score and observed how our model predicted the actual labels. We achieved a 95% accuracy in this model. In this project, we have successfully classified the traffic signs classifier with 95% accuracy and also visualized how our accuracy and loss changes with time, which is pretty good from a simple CNN model.

```
[14]: from sklearn.metrics import accuracy_score
import pandas as pd
y_test = pd.read_csv('Test.csv')

labels = y_test["ClassId"].values
imgs = y_test["Path"].values

data=[]

for img in imgs:
    image = Image.open(img)
    image = image.resize((30,30))
    data.append(np.array(image))

X_test=np.array(data)

pred = model.predict_classes(X_test)

#Accuracy with the test data
from sklearn.metrics import accuracy_score
accuracy_score(labels, pred)
```

```
[14]: 0.9532066508313539
```



Now we can test any image and it will use our saved trained model to classify the uploaded images into different classes of the traffic signs as per the dictionary.



CHAPTER 4

CONCLUSION AND FUTURE ENHANCEMENT

CONCLUSION / INFERENCE

We have successfully built an end-to-end neural network system that can automatically view an image and generate a reasonable description in plain English. It is based on a convolution neural network that encodes an image into a compact representation, followed by a recurrent neural network that generates a corresponding sentence. The model is trained to maximize the likelihood of the sentence given the image. Experiments on several images show the robustness in terms of qualitative results (the generated sentences are very reasonable) and quantitative evaluation, using either ranking metrics or RELU, a metric used in machine translation to evaluate the quality of generated sentences. It is clear that, as the size of the available datasets for image description increases, so will the performance of the proposed model. Furthermore, it will be interesting to see how one can use unsupervised data, both from images alone and text alone, to improve image description approaches.

FUTURE ENHANCEMENT

- A vaster dataset can help in improving the accuracy of the dataset and also will allow the more classes of the traffic signs to be classified.
- The integration with a real time camera system will allow our model to classify images of the traffic signs in real time and also can be implemented with a video system which will constantly classify the traffic signs and store the data and location of the certain signs for future reference.
- Furthermore this can be used in more advanced artificial intelligence (AI) systems for cars and various vehicles to detect and act according to the different traffic signs.

APPENDIX

Machine Learning Model Code:

#Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
data = []
labels = []
classes = 43
cur_path = os.getcwd()
```

#retrieving the images and their labels

```
for i in range(classes):
    path = os.path.join(cur_path, 'train', str(i))
    images = os.listdir(path)
    for a in images:
        try:
            image = Image.open(path + '\\' + a)
            image = image.resize((30,30))
            image = np.array(image)
            #sim = Image.fromarray(image)
            data.append(image)
            labels.append(i)
        except:
            print("Error loading image")
```

#converting lists into numpy arrays

```
data = np.array(data)
labels = np.array(labels)
print(data.shape, labels.shape)
```

#Splitting training and testing dataset

```
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2,
random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

#converting the labels into one hot encoding

```
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)
```

#Building the model

```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu',
input_shape=X_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))
```

#Compilation of the model

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
epochs = 15
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs,
validation_data=(X_test, y_test))
model.save("my_model.h5")
```

#plotting graphs for accuracy

```
plt.figure(0)
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```

```
plt.figure(1)
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title('Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```

#testing accuracy on test dataset

```
from sklearn.metrics import accuracy_score
y_test = pd.read_csv('Test.csv')
labels = y_test['ClassId'].values
imgs = y_test['Path'].values
data=[]
for img in imgs:
    image = Image.open(img)
    image = image.resize((30,30))
    data.append(np.array(image))
X_test=np.array(data)
pred = model.predict_classes(X_test)
```

#Accuracy with the test data

```
from sklearn.metrics import accuracy_score
print(accuracy_score(labels, pred))
```

#save the Model

```
model.save('traffic_classifier1.h5')
```

GUI Code:

#Importing Libraries

```
import tkinter as tk
from tkinter import filedialog
from tkinter import *
from PIL import ImageTk, Image
import numpy
```

#load the trained model to classify sign

```
from keras.models import load_model
model = load_model('traffic_classifier.h5')
```

#Dictionary to label all traffic signs class.

```
classes = { 1:'Speed limit (20km/h)',  
2:'Speed limit (30km/h)',  
3:'Speed limit (50km/h)',  
4:'Speed limit (60km/h)',  
5:'Speed limit (70km/h)',  
6:'Speed limit (80km/h)',  
7:'End of speed limit (80km/h)',  
8:'Speed limit (100km/h)',  
9:'Speed limit (120km/h)',  
10:'No passing',  
11:'No passing veh over 3.5 tons',  
12:'Right-of-way at intersection',  
13:'Priority road',  
14:'Yield',  
15:'Stop',  
16:'No vehicles',  
17:'Veh > 3.5 tons prohibited',  
18:'No entry',  
19:'General caution',  
20:'Dangerous curve left',  
21:'Dangerous curve right',  
22:'Double curve',  
23:'Bumpy road',  
24:'Slippery road',  
25:'Road narrows on the right',  
26:'Road work',  
27:'Traffic signals',  
28:'Pedestrians',  
29:'Children crossing',  
30:'Bicycles crossing',  
31:'Beware of ice/snow',  
32:'Wild animals crossing',  
33:'End speed + passing limits',  
34:'Turn right ahead',  
35:'Turn left ahead',  
36:'Ahead only',  
37:'Go straight or right',  
38:'Go straight or left',  
39:'Keep right',  
40:'Keep left',  
41:'Roundabout mandatory',
```

```
42:'End of no passing',
43:'End no passing veh > 3.5 tons' }
```

#Initialise GUI

```
top=tk.Tk()
top.geometry('800x600')
top.title('Traffic Sign Recognition')
top.configure(background='#CDCDCD')
label=Label(top,background='#CDCDCD', font=('arial',15,'bold'))
sign_image = Label(top)
def classify(file_path):
    global label_packed
    image = Image.open(file_path)
    image = image.resize((30,30))
    image = numpy.expand_dims(image, axis=0)
    image = numpy.array(image)
    pred = model.predict_classes([image])[0]
    sign = classes[pred+1]
    print(sign)
    label.configure(foreground='#011638', text=sign)
```

#Initialise Buttons

```
def show_classify_button(file_path):
    classify_b=Button(top,text="Classify Image",command=lambda:
    classify(file_path),padx=10,pady=5)
    classify_b.configure(background='#364156', foreground='white',font=('arial',10,'bold'))
    classify_b.place(relx=0.79,rely=0.46)
def upload_image():
    try:
        file_path=filedialog.askopenfilename()
        uploaded=Image.open(file_path)
        uploaded.thumbnail(((top.winfo_width()/2.25),(top.winfo_height()/2.25)))
        im=ImageTk.PhotoImage(uploaded)
        sign_image.configure(image=im)
        sign_image.image=im
        label.configure(text='')
        show_classify_button(file_path)
    except:
        pass
upload=Button(top,text="Upload an image",command=upload_image,padx=10,pady=5)
upload.configure(background='#364156', foreground='white',font=('arial',10,'bold'))
upload.pack(side=BOTTOM,pady=50)
```

```
sign_image.pack(side=BOTTOM,expand=True)
label.pack(side=BOTTOM,expand=True)
heading = Label(top, text="Classify Traffic Signs",pady=20, font=('arial',20,'bold'))
heading.configure(background='#CDCDCD',foreground='#364156')
heading.pack()
top.mainloop()
```

REFERENCES

1. Raimonda Staniutė and Dmitrij Šešok: A Systematic Literature Review on Image Captioning (2019) Sauletekio
2. Marc Tanti, Albert Gatt and Kenneth P Camilleri: Where to put Image in an Image Caption Generator (2018) University of Malta.
3. Marc Tanti, Albert Gatt and Kenneth P Camilleri: What is the Role of Recurrent Neural Networks in an Image Caption Generator? (2018) University of Malta.
4. Jason Brownlee: Caption Generation with Inject and Merge Encode-Decoder Models (2017).
5. Kaustubh: ResNet, AlexNet, VGGNet, Inception: Understanding Various Architectures of Convolutional Networks (2019)
6. Kishore Papineni, Salim Roukos, Todd Ward, and Wei-JingZhu: BLEU: A method for Automatic Evaluation of Machine Translation (2002) IBM, New York, USA
7. Rafaella Bernardi, Ruket Cakici, Desmond Elliot: Automatic Description Generation from Images: A Survey of Models, Datasets, and Evaluation Measures (2017)