University of Science and Technology at Zewail City

Machine Learning Project Report

Image classification Using CNN

Submitted to

Dr. Mohamed Elshenawy

From:

Mai Tarek	**201401578**
Shrouk Shalaby	**201400455**
Yomna Shreif	**201401302**

Project Idea

- Our project idea was to perform images classification using Convolution Neural Networks. the classes the images are trained to be classified to are trees , signs , cars , humans , dogs cats , buses and bikes which are common scenes in streets.

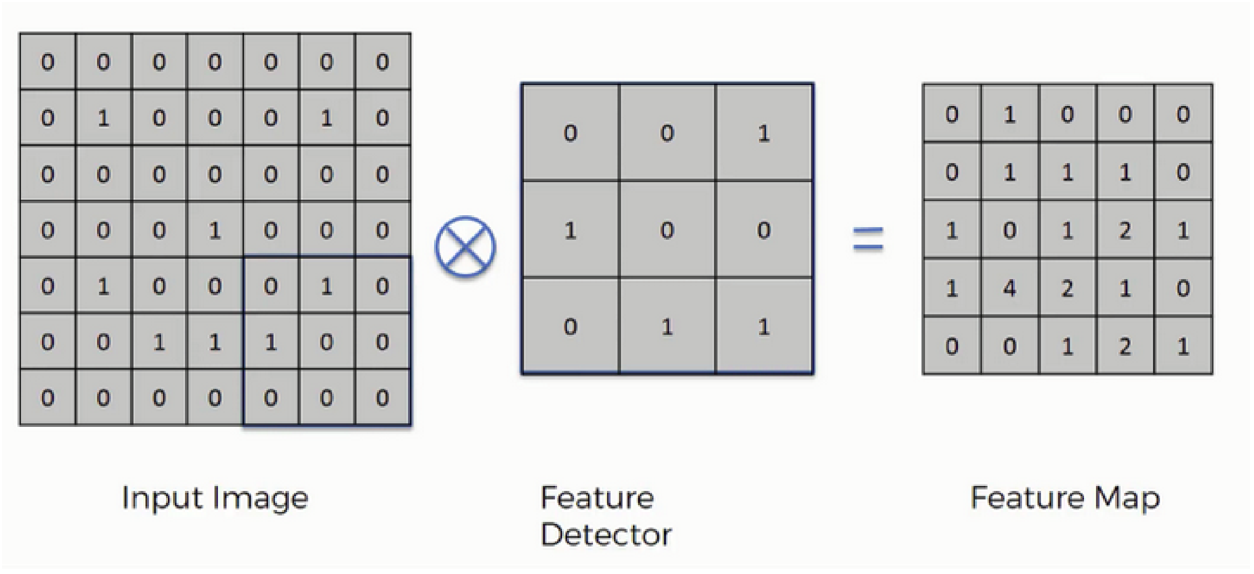
Introduction

First of all , keras library was used to generate a sequential model in order to build up a suitable CNN.

- **Types of Layers used in CNN :**

1. **Convolution Layer :**

   It is comprised of three types of inputs for it's operation input image , feature detector (often referred to as kernel) and feature map. The feature detector is placed over the image from the top left corner and the coinciding elements are multiplied and the result matrix is summed into one element in the feature map , and then the feature detector is slid to the right with a certain number of strides and the process is repeated until the feature map is built up. The convolution layer is useful in that it reduces the image size, while extracting the features that somewhat represents the images.



2. The rectified Linear unit

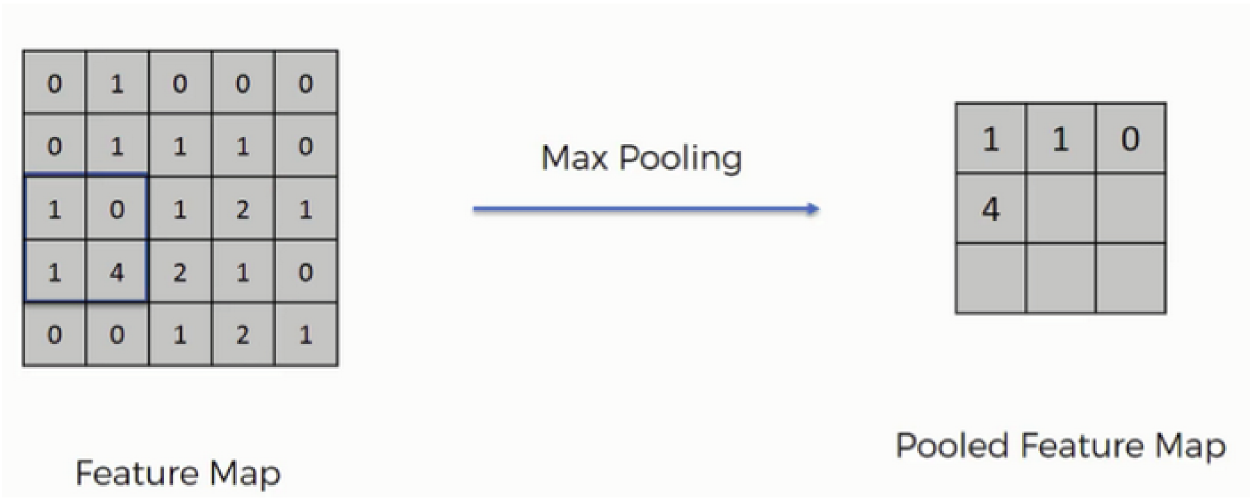
Namely RELU function ,it is not necessary a layer by itself but rather an additional but mandatory step to the convolution layer. It serves to increase the non-linearity in the image , even though images are normally non-linear , training the images using CNN might impose some linearity so we proceed to increases non-linearity even further.

3. Pooling Layer

There are several types of pooling , but we used Max pooling in our model since it’s the most relevant to out task.

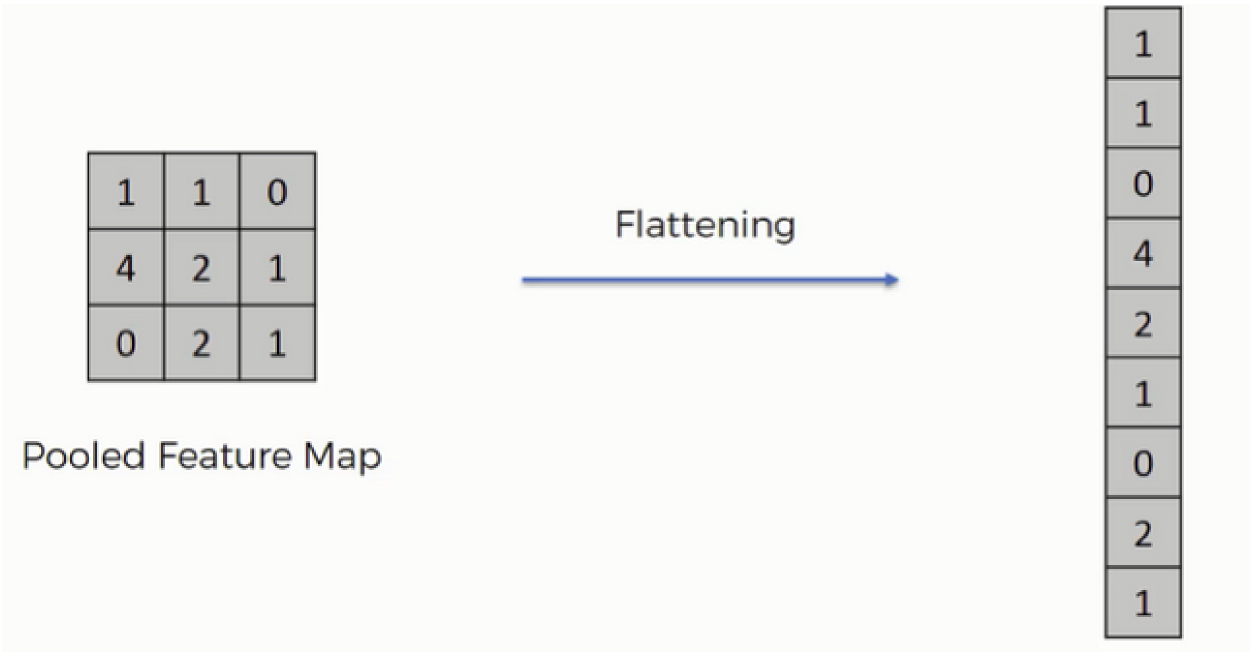
The process is done by placing a box of a certain size on the top left corner of the feature map and the largest number in the coinciding cells of the feature map will be placed in the pooled feature map.

The purpose of the pooling layer is the same as the convolution layer which is to extract the most important features in a map so that even though information is lost the network is able to work effectively. Another reason for pooling the feature map is to make up for possible distortion in the image which can be presented as rotation of the image as an example , even though the feature map elements are rotated the most important feature namely the largest one is still chosen to be in the pooled feature map.
the pooling layer is therefore providing the network with a an ability called “spatial variance” , Moreover it further reduces the size of the image and consequently reducing the possibility of overfitting.



4. **Flattening Layer**

   After obtaining the pooled feature map , the pixels are "flattened" into one column onto each other , as it will later inserted into an Artificial Neural Network.



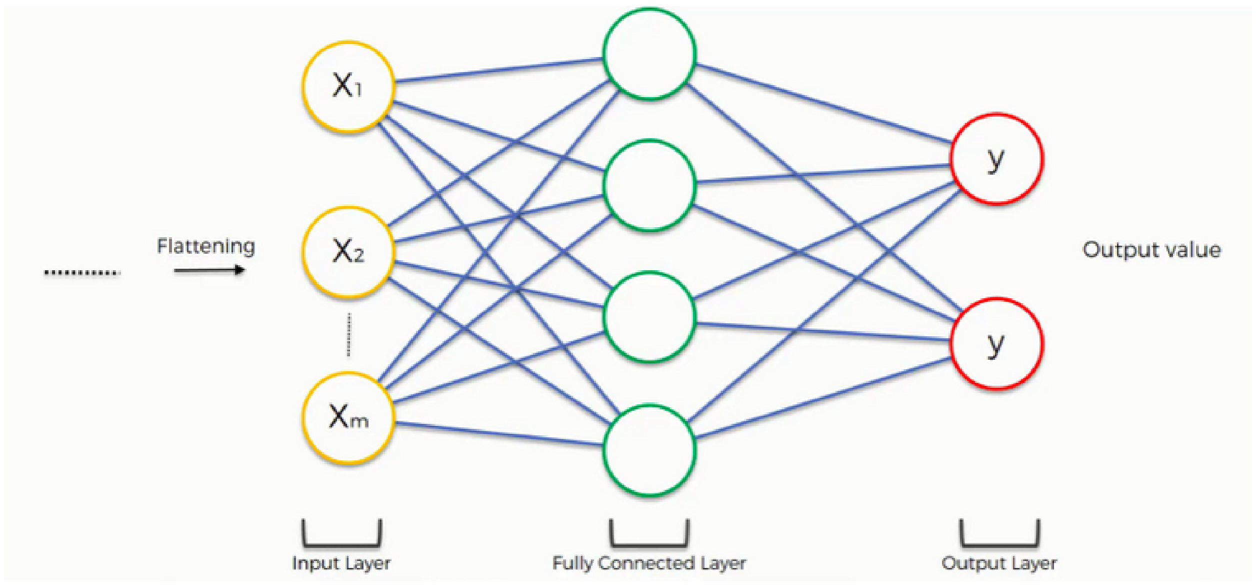
5. Fully connected layer

Or dense layer is a layer of neurons in the artificial neural network where the weights are optimized to provide the best accuracies

6. Last dense layer

It is a layer consisting of neurons of the same number of classes in the classification , and has a softmax function as an activation function. Each neuron displays the probability that the input image belongs to this class.

The purpose of the Artificial neural network is to take the data resulted from the flattening layer and combines the feature into a number of various attributes to result in a much more efficient Constitutional Neural Network and accurate image classification.



7. **Batch Normalization**

   an Additional Layer that is often added after the Convolution Layer and RELU function is batch normalization , This normalizes each batch by their mean and standard deviation.
   It is used for the networks to train faster and be able to converge more quickly. Moreover they Allow for higher learning rates and the weights are initialized easily.

8. **Dropout Layer**

   if the model has trained and result in a very high training accuracy and a relatively lower validation accuracy , this is due to over fitting which is the result of having too many parameter (ie weights) to estimate that the network becomes almost structured for the classification of the training images but is unable to perform well when faced with an image from test or validation data sets.
   Dropout layer is provided with a percentage with which it randomly removes weights , for example if a dropout layer of 0.5 is added to the sequential model , half of the weights are randomly removed from the layer avoiding overfitting of the model.

- **Acquiring Dataset**

We used ImageNet to get our training and validation sets, but the images cannot be downloaded directly from ImageNet. So, a code was developed to do so.

We first defined directories in google colaboratory dynamic memory for the training and validation sets. The code takes input as 8 links (one for each class) each lead to a file full of image URLs for the specific class, the code then parse the URLs and downloads n_training images and m_validation ones in the allocated directory.

- **Data Preprocessing**

a) After downloading the dataset, we found some corrupted images, due to :

  - Some links were already expired.
  - Any interruption during the download.

So, another piece of code was developed to loop over the downloaded images to delete the corrupted ones which had relatively small size.

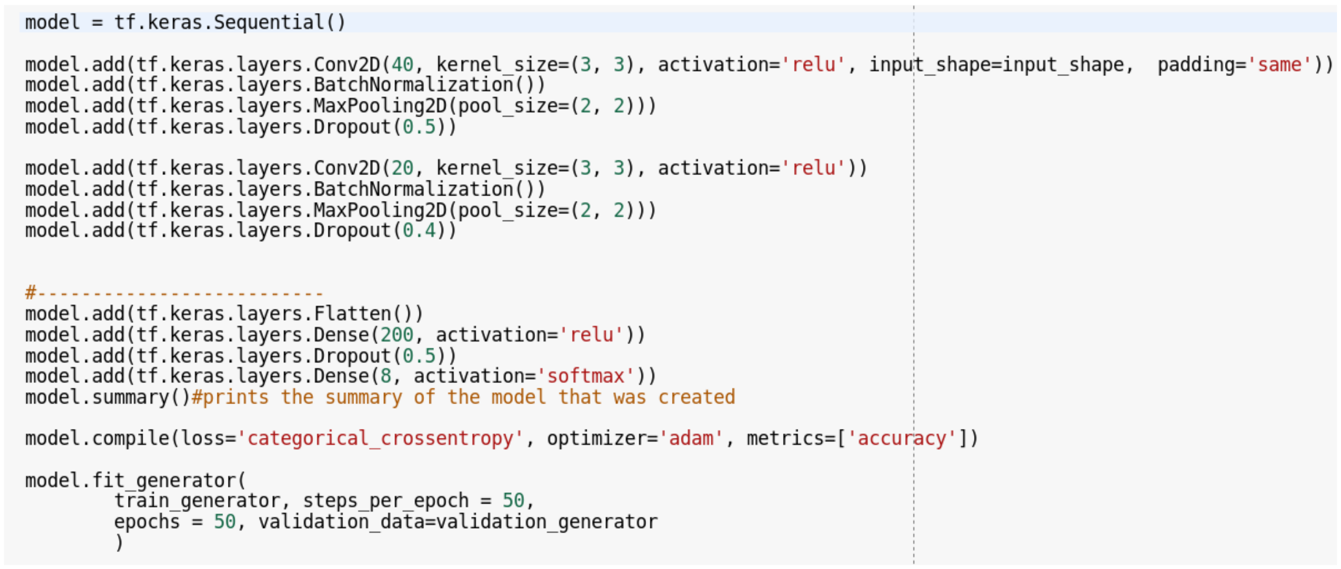
b) The images were not of the same size so each image would pass to a function of resizing to guarantee they all have the same number or rows, columns, and number of channels

- **Model Tuning**

A basic model architecture was used as a first trial with the primary layers (convolution, maxpooling, flatten, and dense). Then, the model parameters were being changed one by one to test the effect of each one to find the optimum architecture. These parameters were:

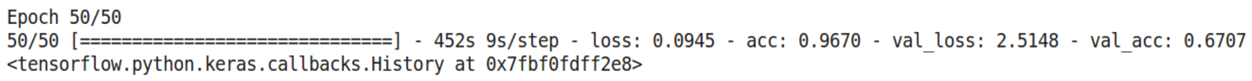
  - Drop out value
  - Dense layer weights
  - Number of filters in the convolution layer
  - Kernel size
  - Batch size and number of steps per epoch
  - With / without batch normalization

- After multiple testings a following model was created



```python
model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv2D(40, kernel_size=(3, 3), activation='relu', input_shape=input_shape,  padding='same'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Conv2D(20, kernel_size=(3, 3), activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.Dropout(0.4))


#-------------------------
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(200, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(8, activation='softmax'))
model.summary()#prints the summary of the model that was created

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit_generator(
        train_generator, steps_per_epoch = 50,
        epochs = 50, validation_data=validation_generator
        )
```

- **Results**

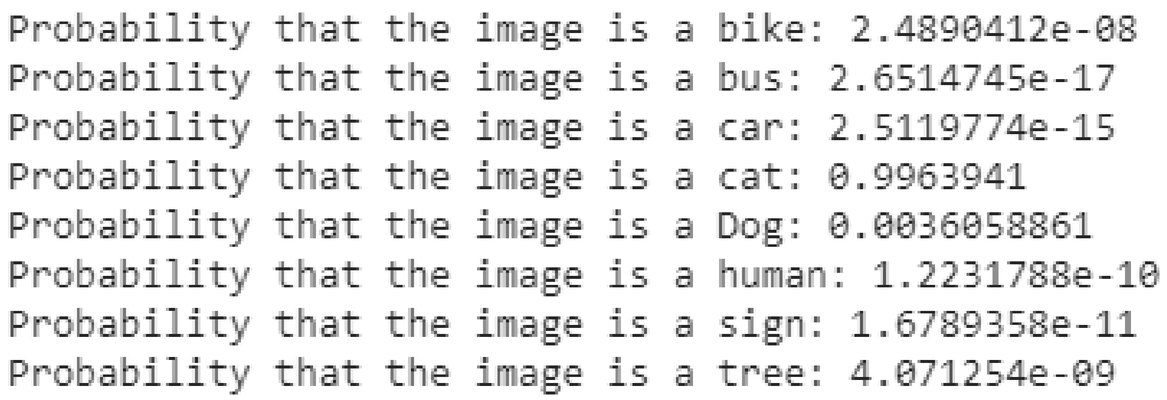


```
Epoch 50/50
50/50 [==============================] - 452s 9s/step - loss: 0.0945 - acc: 0.9670 - val_loss: 2.5148 - val_acc: 0.6707
<tensorflow.python.keras.callbacks.History at 0x7fbf0fdff2e8>
```

After 50 epochs the training accuracy was 96.7 %, while validation accuracy wa 67 % approximately.

- **Testing**

  We input the picture of a cat as a trial of our model,



```
Probability that the image is a bike: 2.4890412e-08
Probability that the image is a bus: 2.6514745e-17
Probability that the image is a car: 2.5119774e-15
Probability that the image is a cat: 0.9963941
Probability that the image is a Dog: 0.0036058861
Probability that the image is a human: 1.2231788e-10
Probability that the image is a sign: 1.6789358e-11
Probability that the image is a tree: 4.071254e-09
```

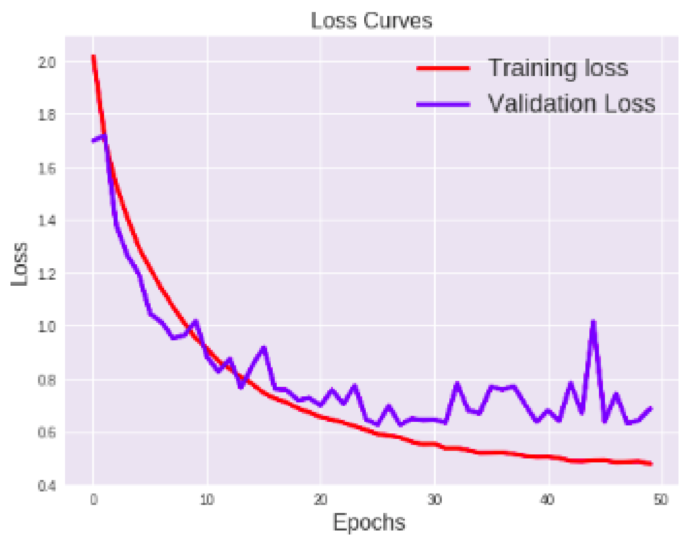
- **Limitations**

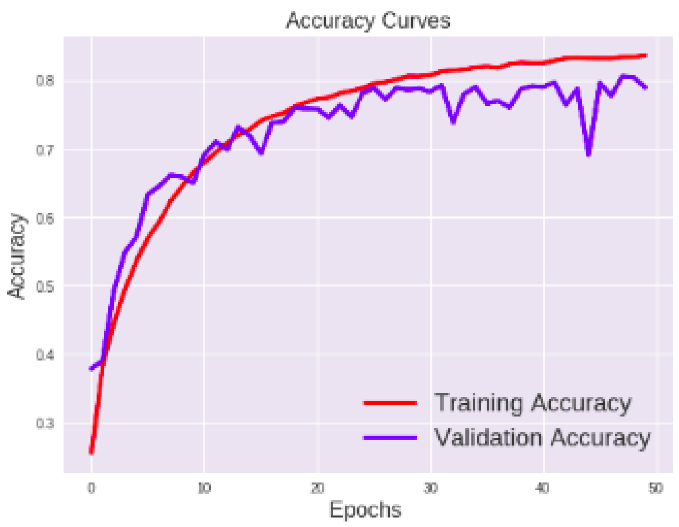
  We had to work with 600 training images in each class (4800 total training images) and 200 validation (1600 total validation) as ImageNet had a limited number of images, so the validation accuracy was not high enough. We believe if the number of validation images were bigger the validation accuracy will be higher.

- **Model Testing with different dataset**

  Due to the multiple problems faced during generating and correcting our data set we decided to test our model with a different dataset “CIFAR-10 ”, which is a ready made dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class, 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. These classes are airplane -automobile -bird -cat -deer -dog -frog -horse -ship -truck.

- **Cifar Result**





From these graphs , we can see that there is a considerable difference between the training and validation loss. This indicates that the network has tried to memorize the training data and thus, is able to get better accuracy on it. However we have already used Dropout, It is still overfitting. So we can using Data Augmentation to reduce this overfitting. Data Augmentation is the process of artificially creating more images from the images you already have by changing the size, orientation etc of the image.

So now we tested our model ,it works fine and for future enhancement we will use a better data set .