



# ***CONVOLUTIONAL NEURAL NETWORKS***

## **PROJECT REPORT**

***OUTAISS Marouane  
LAMRINI Youssef  
EL IDRISSE EL HASSANI Hamza  
EL BAKKAL Haytam  
GSEII2***

***Supervised by:***

***Pr. CHOUGRAD Hiba***

## Table of contents

Introduction: .....	3
Chapter I : Introduction & problematic.....	3
1.    Problematic:.....	3
2.    Artificial Neural Networks:.....	4
Disadvantages of Artificial Neural Networks: .....	4
3.    How computer reads an image:.....	4
4.    Why not normal ANN for image recognition? .....	5
Conclusion:.....	6
Chapter II : Convolutional neural networks: .....	7
Introduction: .....	7
1.    The Architecture of Convolutional Neural Networks:.....	7
2.    The Convolutional Layer: .....	8
Rectified Linear Unit (ReLU): .....	11
3.    Pooling layer: .....	12
4.    Fully Connected Layer: .....	14
Conclusion:.....	15
Chapter III : The implementation of CNN.....	16
Introduction: .....	16
1. Importing the dataset.....	17
2. Normalization and One hot encoding: .....	18
3. Modeling: .....	19
3.1. Base Model: .....	19
3.2. Second Model: .....	21
3.3. Other Models:.....	21
3.4. Last Model: .....	21
4. Making predictions:.....	23
5. Saving our model:.....	24
Conclusion:.....	24
Chapter IV : The deployment of the CNN model .....	25
Introduction .....	25
1. Back-End:.....	25
2. Front-end.....	27
Conclusion:.....	30

## Chapter I: Introduction & problematic

### Introduction:

Convolutional neural networks. Sounds like a weird combination of biology and math with a little CS sprinkled in, but these networks have been some of the most influential innovations in the field of computer vision. 2012 was the first year that neural nets grew to prominence as Alex Krizhevsky used them to win that year's ImageNet competition (basically, the annual Olympics of computer vision), dropping the classification error record from 26% to 15%, an astounding improvement at the time. Ever since then, a host of companies have been using deep learning at the core of their services. Facebook uses neural nets for their automatic tagging algorithms, Google for their photo search, Amazon for their product recommendations, Interest for their home feed personalization, and Instagram for their search infrastructure.

### 1. Problematic:

Image classification is the task of taking an input image and outputting a class (a cat, dog, etc.) or a probability of classes that best describes the image. For humans, this task of recognition is one of the first skills we learn from the moment we are born and is one that comes naturally and effortlessly as adults. Without even thinking twice, we're able to quickly and seamlessly identify the environment we are in as well as the objects that surround us. When we see an image or just when we look at the world around us, most of the time we can immediately characterize the scene and give each object a label, all without even consciously noticing. These skills of being able to quickly recognize patterns, generalize from prior knowledge, and adapt to different image environments are ones that we do not share with our fellow machines.



What We See

```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 40 87 17 40 98 43 49 48 04 56 42 00
81 49 31 75 55 79 14 29 93 71 40 47 53 88 30 03 49 13 36 45
32 70 95 23 04 40 11 42 49 24 68 54 01 32 54 71 37 02 36 91
22 31 16 71 51 47 63 89 41 92 34 54 22 40 40 28 46 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 95 81 29 44 23 47 10 26 38 40 47 59 54 70 66 18 38 44 70
67 24 20 68 02 42 12 20 95 43 94 39 43 08 40 91 46 49 34 21
24 55 58 05 46 73 99 26 97 17 78 78 36 83 14 88 34 89 43 72
21 36 23 09 75 00 76 44 20 45 35 14 00 41 33 97 34 31 33 95
78 17 53 28 22 75 31 47 15 94 03 80 04 42 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 54 00 48 35 71 89 07 05 44 44 37 44 40 21 58 51 54 17 58
19 80 81 48 05 94 47 49 28 73 32 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 14 07 97 57 32 14 24 24 79 33 27 98 44
88 34 48 87 57 42 20 72 03 46 33 47 46 55 12 32 43 93 53 49
04 42 14 73 38 29 39 11 24 94 72 18 08 14 29 32 40 42 76 34
20 49 34 41 72 30 23 88 34 42 99 49 82 47 59 85 74 04 36 14
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 49 14 92 33 48 41 43 52 01 89 19 47 48
```

What Computers See

Figure 1: How is an image seen to a computer

## 2. Artificial Neural Networks:

The term "**Artificial Neural Network**" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.

### Disadvantages of Artificial Neural Networks:

- *Assurance of proper network structure:*

There is no particular guideline for determining the structure of artificial neural networks. The appropriate network structure is accomplished through experience, trial, and error.

- *Unrecognized behavior of the network:*

It is the most significant issue of ANN. When ANN produces a testing solution, it does not provide insight concerning why and how. It decreases trust in the network.

- *Hardware dependent:*

Artificial neural networks need processors with parallel processing power, as per their structure. Therefore, the realization of the equipment is dependent.

- *Difficulty of showing the issue to the network:*

ANNs can work with numerical data. Problems must be converted into numerical values before being introduced to ANN. The presentation mechanism to be resolved here will directly impact the performance of the network. It relies on the user's abilities.

- *The duration of the network is unknown:*

The network is reduced to a specific value of the error, and this value does not give us optimum results.

## 3. How computer reads an image:

A computer sees an image as 0s and 1s. Pixel is the smallest unit in an image.

When we take a digital image, it is stored as a combination of pixels. Each pixel contains a different number of channels.

If it is a grayscale image, it has only one pixel, whereas if it is a colored image, it contains three channels: red, green and blue.

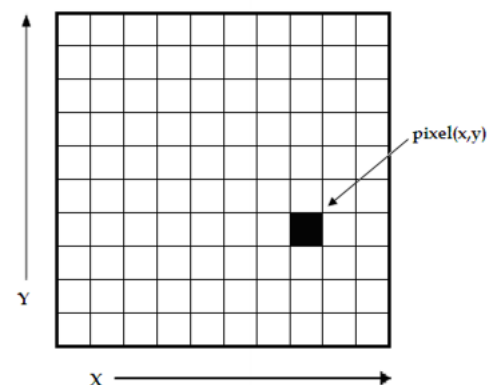
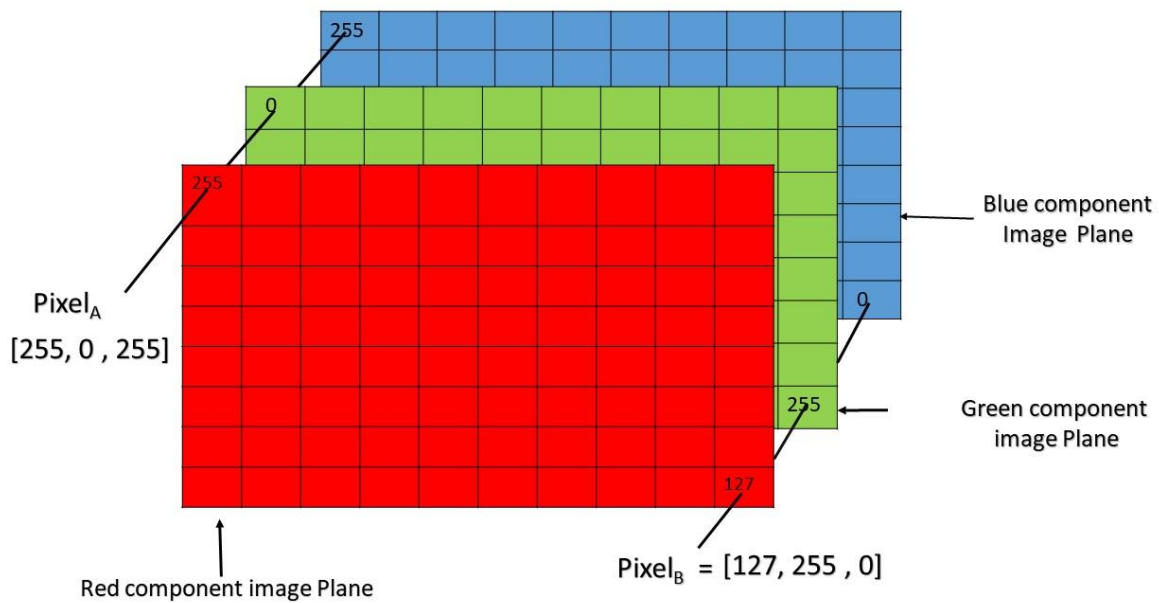


Figure 2 : a pixel represented by x and y coordinates



Pixel of an RGB image are formed from the corresponding pixel of the three component images

Figure 3 : a pixel representation in an RGB image

As shown in the above representation of a digital-colored image, each channel of each pixel has a value between 0 and 255. Each of these values represented in binary before a computer can understand the image.

A machine (or a computer) can be taught how to understand an image and say what the image contains. This is an example of machine learning, teaching a computer to understand and describe an image. It is similar to how we teach kids to identify different alphabets or differentiate between an apple and a banana by showing examples of each case. This is exactly how a computer learns to identify objects in an image.

Computers have machine learning models, which can be thought of as a skill, to perform the same task. As humans need to be trained to perform a particular skill, computers need to train the machine learning model as well.

Similar to how a kid is taught to identify an apple, a machine learning model can be taught how to identify an apple in an image by giving several example images that contain an apple. From these example images, the model learning features of an apple, like its shape and color. Now when a new image of an apple is presented to this computer with this model, it can use what it had learned about apples earlier and identify that this new image also contains apple.

## 4. Why not normal ANN for image recognition?

With ANN, concrete data points must be provided. For example, in a model where we are trying to distinguish between dogs and cats, the width of the noses and length of the ears must be explicitly provided as data points.



When using CNN, these spatial features are extracted from image input. This makes CNN ideal when thousands of features need to be extracted. Instead of having to measure each individual feature, CNN gathers these features on its own.

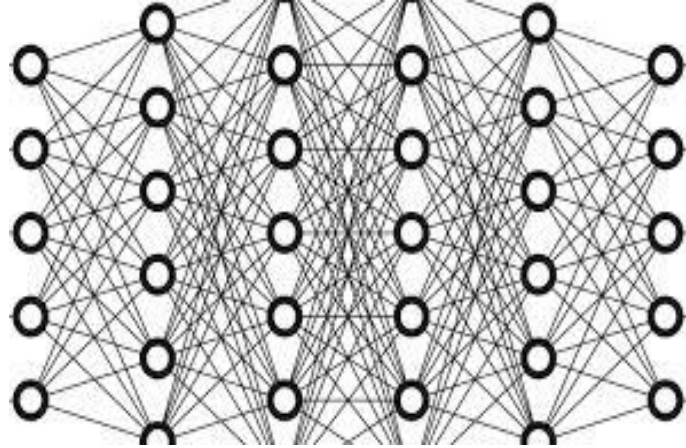


Figure 4 : the structure of artificial neural networks

Image with  
200\*200\*3 Pixels



Number of weights in the  
first hidden layer will be 120000

Using ANN, image classification problems become difficult because 2 dimensional images need to be converted to 1 dimensional vector. This increases the number of trainable parameters exponentially. Increasing trainable parameters takes storage and processing capability. In other words, it would be expensive. The main advantage of CNN compared to its predecessors is that it automatically detects the important features without any human supervision. This is why CNN would be an ideal solution to computer vision and image classification problems.

## Conclusion:

In general, CNN tend to be a more powerful and accurate way of solving classification problems. ANN is still dominant for problems where datasets are limited and image inputs are not necessary. Because of CNN's ability to view images as data, it's the most prevalent solution for computer vision and image-dependent machine learning problems.

## Chapter III

### Convolutional neural networks:

#### Introduction:

A **Convolutional Neural Network (CNN)** is a Deep Learning algorithm which can take in an input image, assign importance to various aspects and objects in the image which are learnable weights and biases, and be able to differentiate one from the other.

The pre-processing required in a CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, CNN have the ability to learn these characteristics.

#### 1. The Architecture of Convolutional Neural Networks:

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs.

They have three main types of layers, which are:

- Convolutional layer.
- Pooling layer.
- Fully-connected (FC) layer.

The convolutional layer is the first layer of a convolutional network.

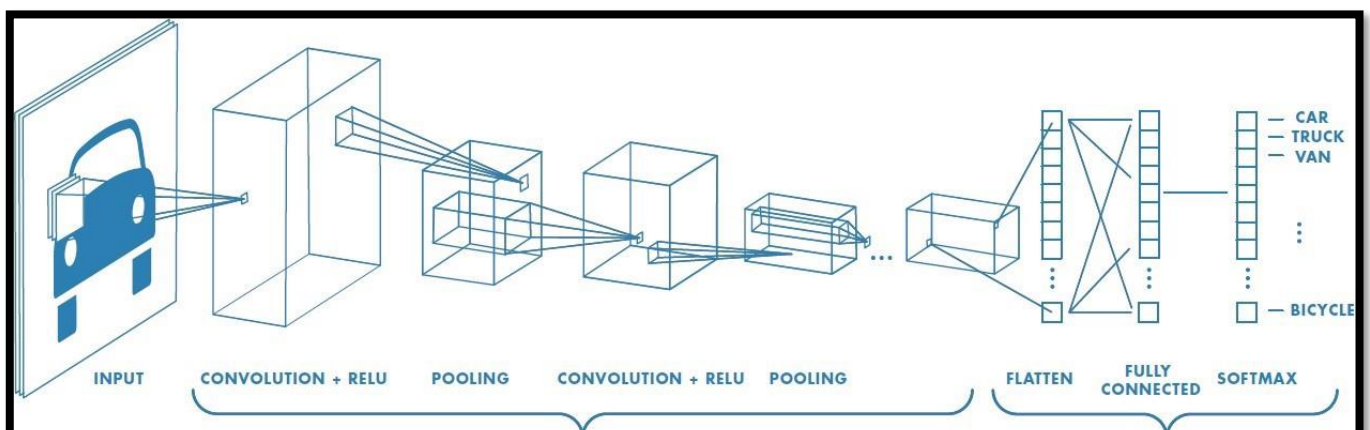


Figure 5 : The architecture of convolutional neural networks

While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer.

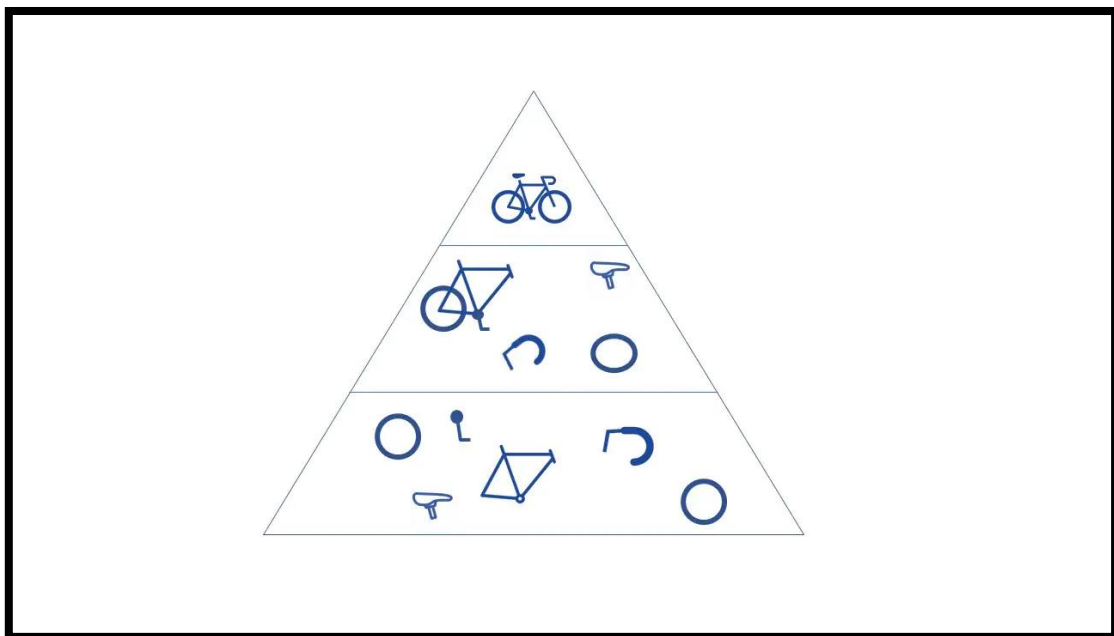
With each layer, the CNN increases in its complexity, identifying greater portions of the image.

Earlier layers focus on simple features, such as colors and edges.

As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object.

## 2. The Convolutional Layer:

The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. Its final objective is to extract characteristics specific to each image. For example, let's assume that we're trying to determine if an image contains a bicycle. You can think of the bicycle as a sum of parts. It is comprised of a frame, handlebars, wheels, pedals, et cetera. Each individual part of the bicycle makes up a lower-level pattern in the neural net, and the combination of its parts represents a higher-level pattern, creating a feature hierarchy within the CNN.



*Figure 6 : decomposing a bicycle into significant parts*

The convolutional layer requires a few components, which are input data, a filter, and a feature map and it has a few components and parameters that need to be clarified, which are:

- The kernel/feature detector/Filter.
- The feature map/activation map/convolved feature.
- The stride.
- The padding.

So, they will be explained and explored as things progress.



In our case, the input is a color image, which is made up of a matrix of pixels in 3D. This means that the input will have three dimensions—a height, width, and depth—which correspond to RGB in an image. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.

The feature detector is a two-dimensional (2-D) array of weights, which represents part of the image. Its parameters are automatically chosen by the model during the training.

While they can vary in size, the filter size is typically a 3x3 matrix. This also determines the size of the receptive field.

The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter. This dot product is then fed into an output array.

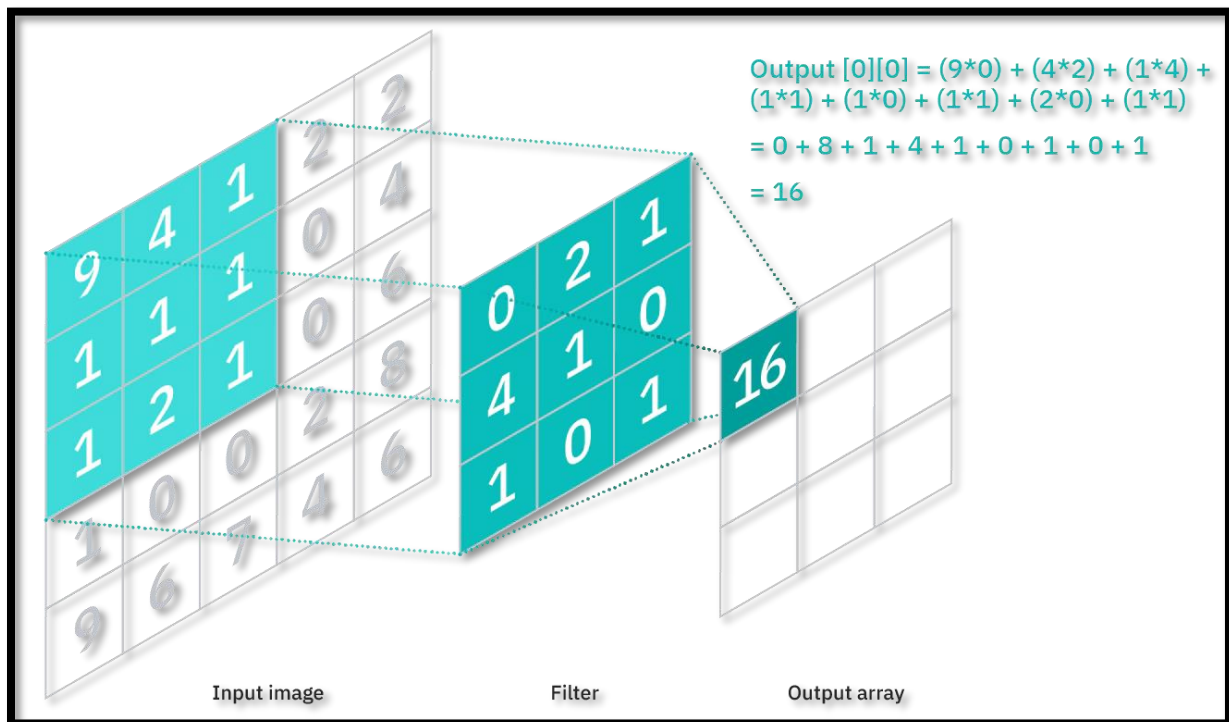


Figure 7 : The dot product of the kernel and a square from the input matrix

Afterwards, the filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

Note that the **Stride** is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values of two or greater is rare, a larger stride yields a smaller output.

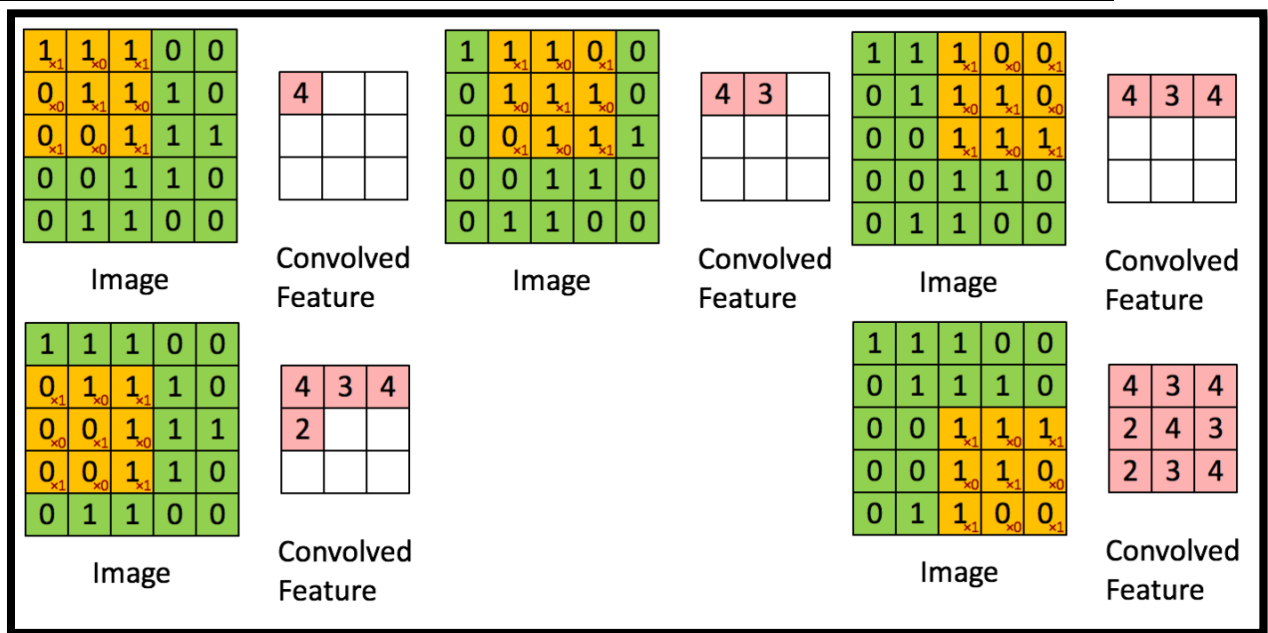


Figure 8 : the convolution process and how the feature map is produced

For example, in this figure we are using a stride that equals to 1.

The final output from the series of dot products from the input and the filter is known as a feature map, activation map, or a convolved feature, which contains the most significant characteristics of the image.

The size of the feature map can be deduced by the sizes of the input matrix and the kernel, if we suppose that the input matrix's size is  $n \times n$ , and the kernel's size is  $p \times p$ , then if we are not using any padding, the size of the feature map would be:  $(n-p+1) \times (n-p+1)$ .

In figure 8, we have a  $5 \times 5$  input matrix and a  $3 \times 3$  kernel, which results of a

$(5-3+1) \times (5-3+1)$  i.e.  $3 \times 3$  feature map.

Another parameter that we can adjust while convolving is the padding, which is usually used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero, producing a larger or equally sized output.

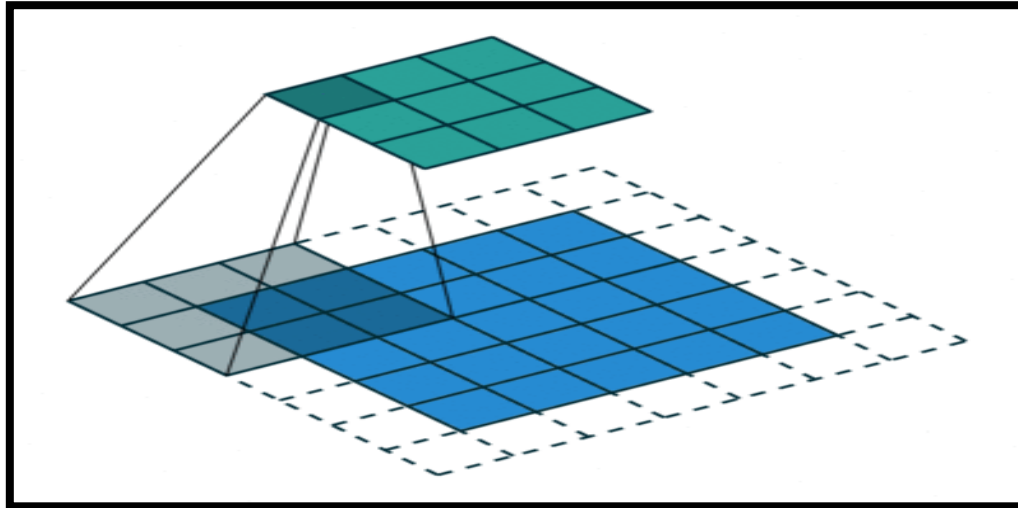


Figure 9 : the convolution process with a padding of 1 pixel

It helps avoiding data loss and preserving the whole image including its edges if they are significant. But the larger it is the larger the feature map's size would be.

For example, in figure 9 we are using a padding that equals to 1 pixel, and for figure 8 we did not use any padding.

## Rectified Linear Unit (ReLU):

After each convolution operation, a CNN applies a Rectified Linear Unit (ReLU) transformation to the feature map, RELU layer will apply an elementwise activation function, such as the  $\max(0, x)$  thresholding at zero.

This layer replaces all negative values received as inputs with zeros. The interest of these activation layers is to make the model nonlinear and therefore more complex.

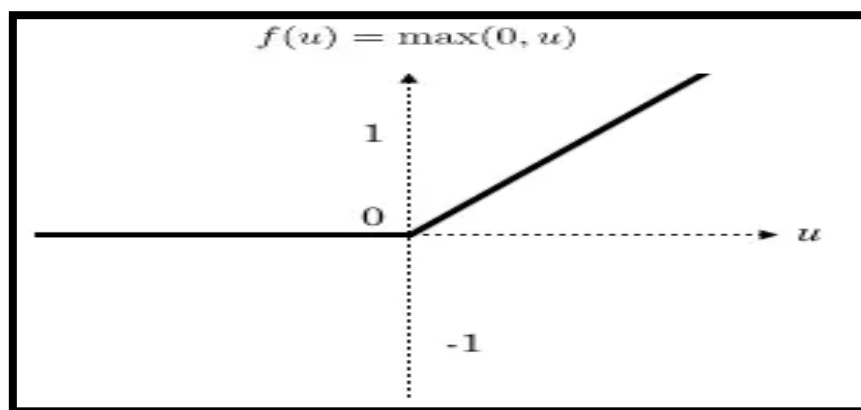


Figure 10 : the ReLU activation function  $\max(0, x)$

The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time, since only a certain number of neurons are activated, the ReLU function is far more computationally efficient when compared to other activation functions.

### 3. Pooling layer:

Similar to the Convolutional Layer, the Pooling layer or the downsampling layer is responsible for reducing the spatial size of the Convolved Feature which is the input into the fully connected neural networks. This is to **decrease the computational power required to process the data** through dimensionality reduction.

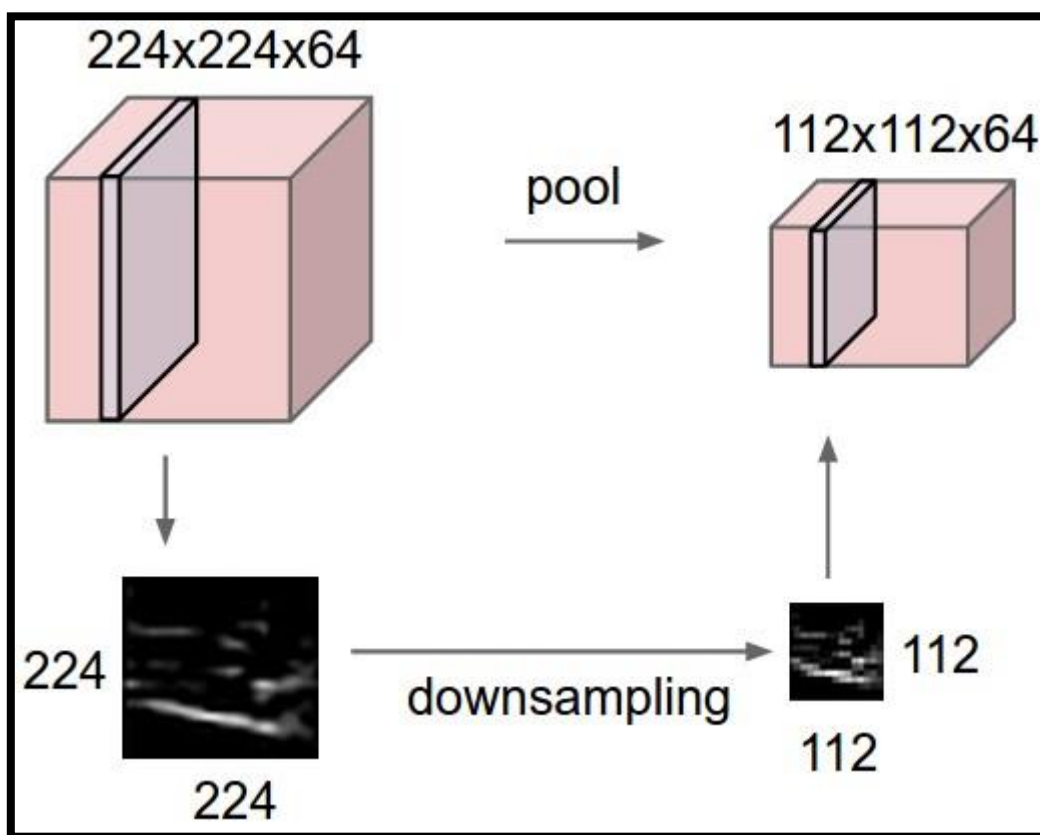


Figure 11 : downsampling and reducing the size of a volume by a filter size 2 and stride 2

Furthermore, it is useful for **extracting dominant features** which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

The pooling operation sweeps a filter across the entire input like the convolutional layer, but the difference is that this filter does not have any weights. Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array.

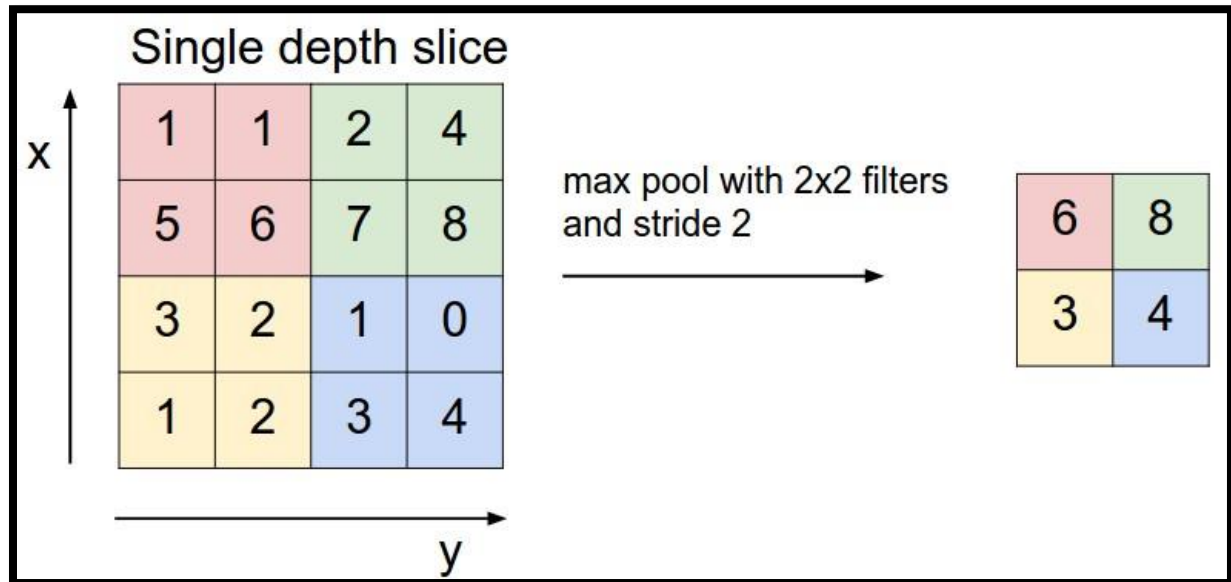


Figure 12 : max pooling with a filter of 2x2 and a stride of 2

There are two main types of pooling:

- **Max pooling:** As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. As an aside, this approach tends to be used more often compared to average pooling.
- **Average pooling:** As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.

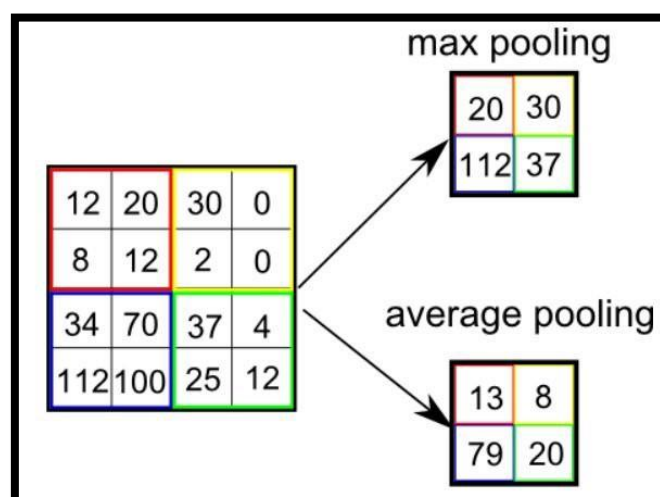


Figure 13 : the difference between max and average pooling



Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. So, we can say that Max Pooling performs a lot better than Average Pooling.

While a lot of information is lost in the pooling layer, it also has a number of benefits to the CNN. They help to reduce complexity, improve efficiency, and limit risk of overfitting.

## 4. Fully Connected Layer:

The name of the full-connected layer aptly describes itself. As mentioned earlier, the pixel values of the input image are not directly connected to the output layer in partially connected layers. However, in the fully-connected layer, each node in the output layer connects directly to a node in the previous layer.

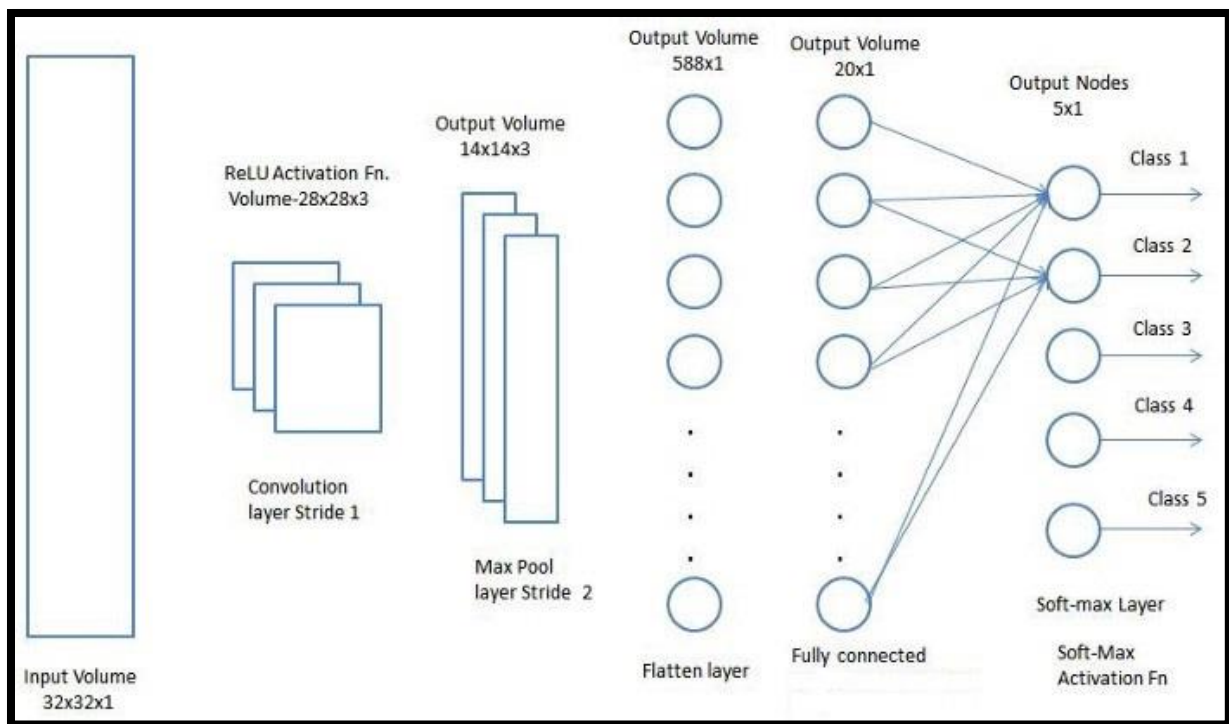


Figure 14 : the classification procedure of 32x32x3 images.

As we can see in the figure, the pooling output is flattened and then injected into the fully connected layer which is a typical multi-layer perceptron that has  $n$  outputs, with  $n$  representing the number of classes, this layer performs the task of classification based on the features extracted through the previous layers and their different filters.

In the figure below we can understand the flattening process from flattening a simple 3x3 matrix:

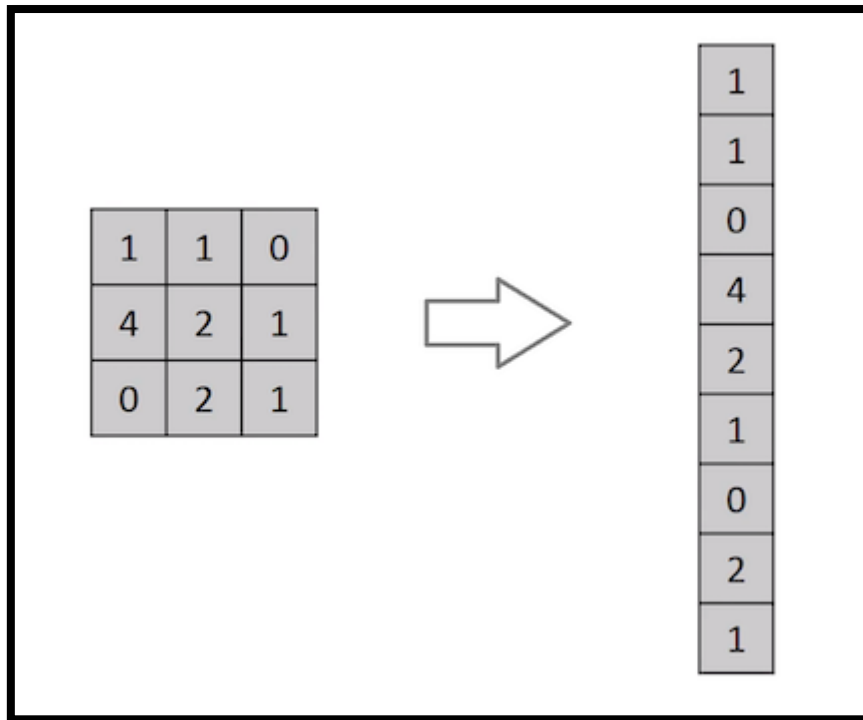


Figure 15 : Flattening of a 3x3 image matrix into a 9x1 vector

While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a SoftMax activation function to classify inputs appropriately, producing a probability from 0 to 1.

So, the output of this layer which is the final output would be an nx1 vector containing values between 0 and 1 that represent the probability for each class.

## Conclusion:

There are various architectures of CNNs available which have been key in building algorithms which power and shall power AI as a whole in the foreseeable future. Some of them have been listed below:

- LeNet
- AlexNet
- VGGNet
- GoogLeNet
- ResNet
- ZFNet

## Chapter III

### The implementation of CNN

#### Introduction:

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 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.

Here are the classes in the dataset, as well as 10 random images from each:

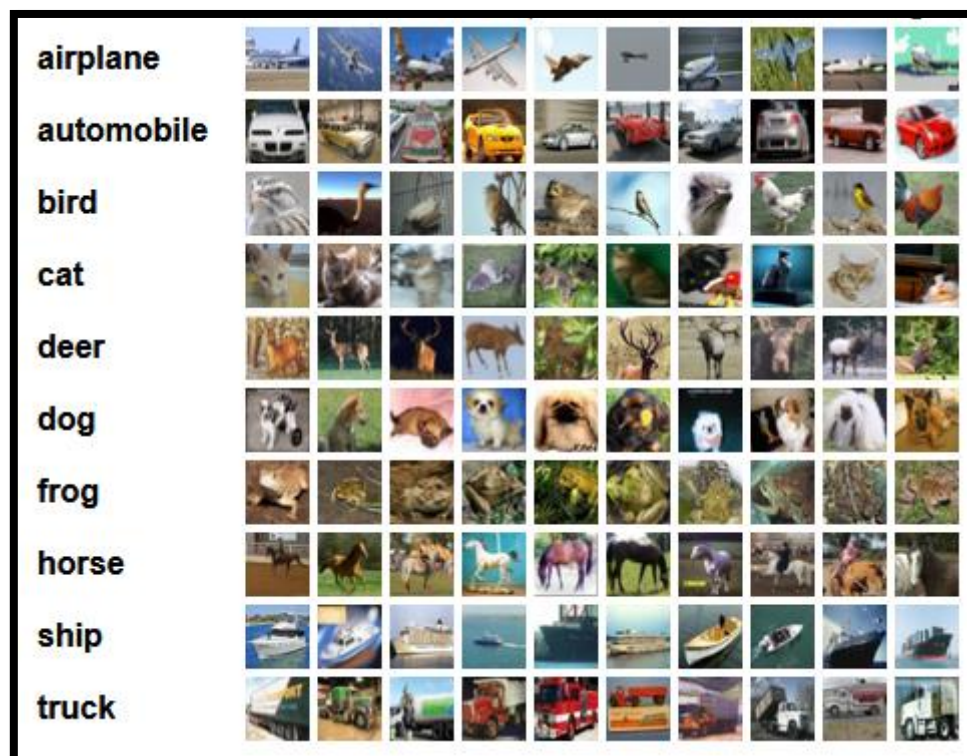


Figure 16 : the CIFAR-10 dataset classes and 10 random images from each class

## 1. Importing the dataset

We are going to import data directly from the Keras datasets instead of getting it from Kaggle or anywhere else, because Keras already has the cleaned version of the same data, so it's better to use it than losing time on data cleaning.

So, our data contains 60K images, divided into train data ( $x_{train}$ ,  $y_{train}$ ) and test data ( $x_{test}$ ,  $y_{test}$ ). We can directly unpack the data into respective variables as shown below:

```
from tensorflow.keras.datasets import cifar10

(x_train, y_train), (x_test, y_test) = cifar10.load_data()

print('Shape of x_train is {}'.format(x_train.shape))
print('Shape of x_test is {}'.format(x_test.shape))
print('Shape of y_train is {}'.format(y_train.shape))
print('Shape of y_test is {}'.format(y_test.shape))
```

Downloading data from <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>  
170500096/170498071 [=====] - 2s 0us/step  
Shape of x\_train is (50000, 32, 32, 3)  
Shape of x\_test is (10000, 32, 32, 3)  
Shape of y\_train is (50000, 1)  
Shape of y\_test is (10000, 1)

Figure 17 : unpacking the train and test data from the dataset

So, we can see that our data was divided into 50K images for the train and 10K for the test set.

Then we write a code to see the first 25 images in the dataset, to secure that the dataset was imported successfully:

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i], cmap=plt.cm.binary)

    plt.xlabel(class_names[y_train[i][0]])
plt.show()
```

Figure 18 : code for showing some of the images from the dataset

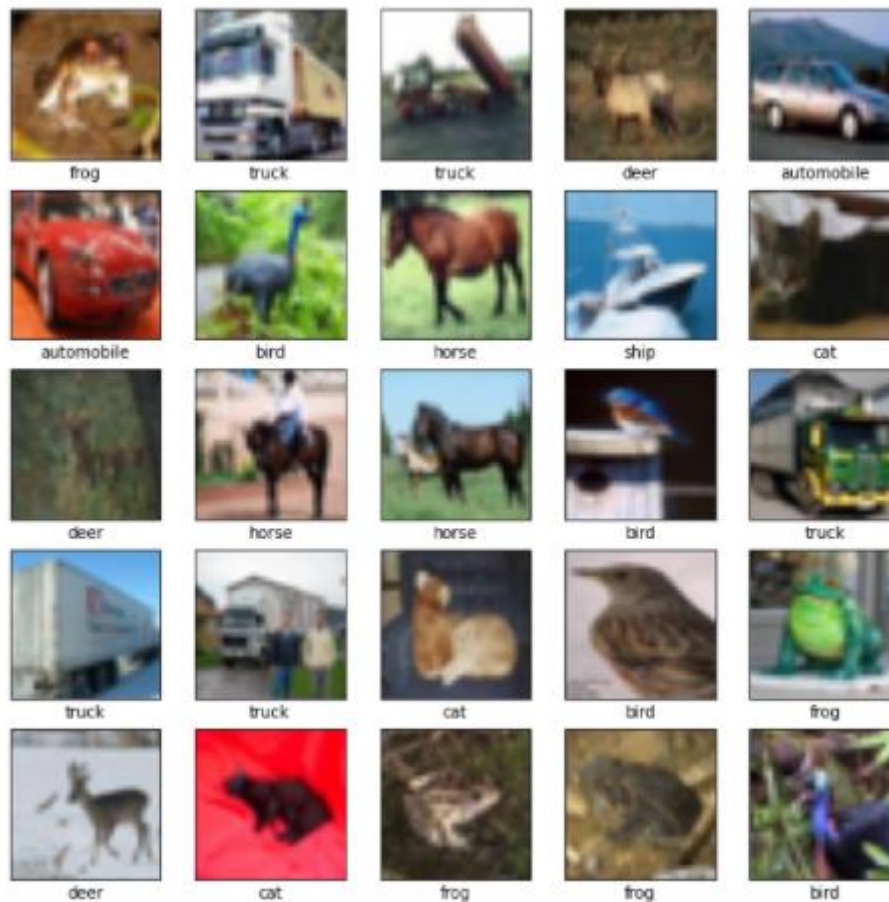


Figure 19 : the result of the code from figure 18

## 2. Normalization and One hot encoding:

Now the dataset is ready, but we still need to normalize the data (data normalization).

Because normalizing the dataset in deep learning will give us very good results.

We know that the pixel values are between 0-255. So, to normalize our images, we can simply divide the whole pixel values by 255.

Because if we divide any number between 0-255, we will get a number between 0 and 1.

```
from tensorflow.keras.utils import to_categorical

# Normalisation
x_train = x_train/255
x_test = x_test/255

#One hot encoding
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)
```

Figure 20 : code for normalizing the data



Now our dataset is normalized. And to make sure of it we can see that the pixels of `x_train[0]` for example have become between 0 and 1.

Figure 21 : The content of the training set after normalization

```
[5] x_train[0]
array([[0.23137255, 0.24313725, 0.24705882],
       [0.16862745, 0.18039216, 0.17647059],
       [0.19607843, 0.18823529, 0.16862745],
       ...,
       [0.61960784, 0.51764706, 0.42352941],
       [0.59607843, 0.49019608, 0.4       ],
       [0.58039216, 0.48627451, 0.40392157]],

       [[0.0627451 , 0.07843137, 0.07843137],
        [0.         , 0.         , 0.         ],
        [0.07058824, 0.03137255, 0.         ],
        ...,
        [0.48235294, 0.34509804, 0.21568627],
        [0.46666667, 0.3254902 , 0.19607843],
        [0.47843137, 0.34117647, 0.22352941]],

       [[0.09803922, 0.09411765, 0.08235294],
        [0.0627451 , 0.02745098, 0.         ],
        [0.19215686, 0.10588235, 0.03137255],
        ...,
        [0.4627451 , 0.32941176, 0.19607843],
        [0.47058824, 0.32941176, 0.19607843],
        [0.42745098, 0.28627451, 0.16470588]],

       ...,
```

## 3. Modeling:

### 3.1. Base Model:

Any deep learning model that needs to classify images (Binary or multiple classification) uses Convolutional Neural Networks. Because CNNs are proven very effective on image data classification.

Let's start with 2 basic CNN layers, where each layer is attached to a Maxpool layer. Max pooling is a great way to reduce the size of parameters without losing too much information.

Two simple convolutional layers with 32 filters each and input shape is (32x32x3) and Relu will be the activation function.

```
model1=Sequential()

model1.add(Conv2D(filters=32,kernel_size=(4,4),input_shape=(32,32,3),activation='relu'))
model1.add(MaxPool2D(pool_size=(2,2)))

model1.add(Conv2D(filters=32,kernel_size=(4,4),input_shape=(32,32,3),activation='relu'))
model1.add(MaxPool2D(pool_size=(2,2)))

model1.add(Flatten())
model1.add(Dense(256,activation='relu'))
model1.add(Dense(10,activation='softmax'))

model1.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

Figure 22 : code for training the first model

So, this is our first model. And when we trained it, we obtained an accuracy of 64% which is not good enough.

```
[ ] # Let's see the accuracy
evaluation = model1.evaluate(x_test, y_test_cat)
print('Test Accuracy: {}'.format(evaluation[1]))

313/313 [=====] - 1s 3ms/step - loss: 3.2679 - accuracy: 0.6432
Test Accuracy: 0.6431999802589417
```

Figure 23 : the accuracy the first model

And when we plot the history graphs (Accuracy, val\_accuracy, loss and val\_loss) we see that the validation loss is increasing, and this means that our model is overfitting, and that's not a good sign.

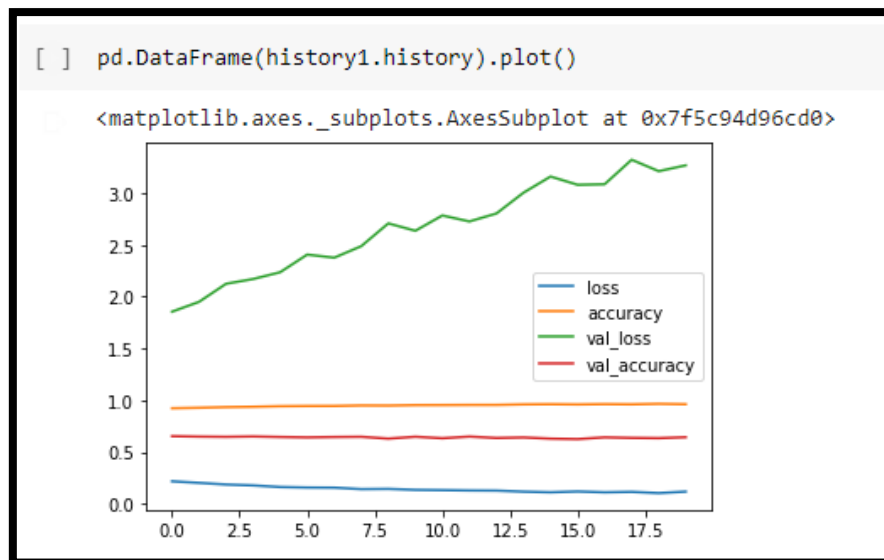


Figure 24 : history graphs of the first model

So how can we reduce Overfitting?

Generally, we have 3 options:

- Add more data
- Decrease model complexity
- Apply regularization

Since we have 60K of images, that means that we have a large dataset, and also our model is not complex. So, option 1 and 2 aren't going to serve us.

What if we regularize the model? in deep learning dropout is a very good form of regularization.

## 3.2. Second Model:

In our second model we added dropout, and that really helps a lot to improve our model performance as shown on the picture below:

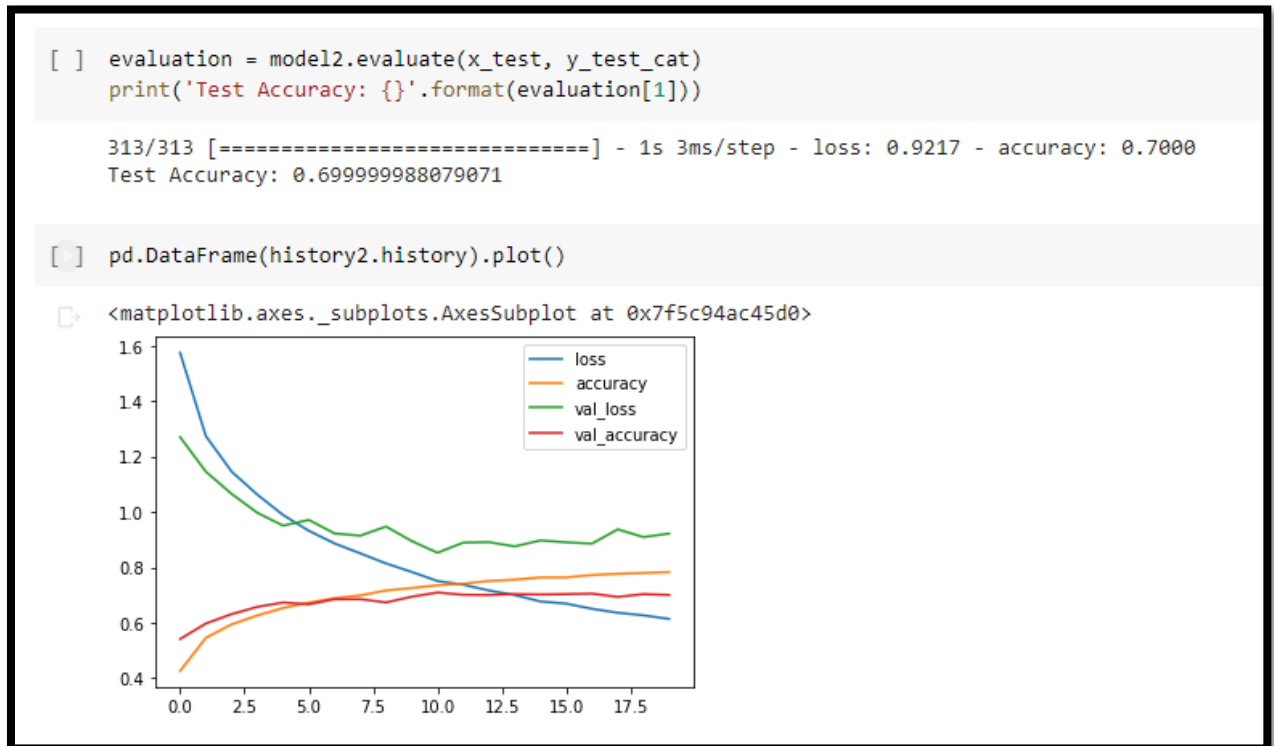


Figure 25 : the accuracy and history graphs of the second model

So, the accuracy is now around 70% and the val\_loss is decreasing.

## 3.3. Other Models:

we proceed like that on the other models until the 6th model which is our final model that gives us a good result (accuracy = 90%):

- 3rd model: add more layers and filters.
- 4th model: increasing layers and filters along with Increasing dropout.
- 5th model: adding batch normalization.
- 6th and final model: image augmentation (Adding more images from 1 image).

## 3.4. Last Model:

When we get to the 6th model, we observe that the results are very good. (You can see the code of the model in the file cifar10.ipynb, can't include the screenshot because the code is large).

- The Classification reports :

```
[ ] predictions = np.argmax(model6.predict(x_test), axis=-1)
```

```
[ ] from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test,predictions))
```

	precision	recall	f1-score	support
0	0.93	0.88	0.90	1000
1	0.94	0.96	0.95	1000
2	0.86	0.88	0.87	1000
3	0.86	0.74	0.80	1000
4	0.89	0.89	0.89	1000
5	0.85	0.86	0.86	1000
6	0.91	0.94	0.93	1000
7	0.92	0.93	0.92	1000
8	0.92	0.95	0.93	1000
9	0.90	0.96	0.93	1000
accuracy			0.90	10000
macro avg	0.90	0.90	0.90	10000
weighted avg	0.90	0.90	0.90	10000

Figure 26 : the classification report of the final model

- The Confusion matrix:

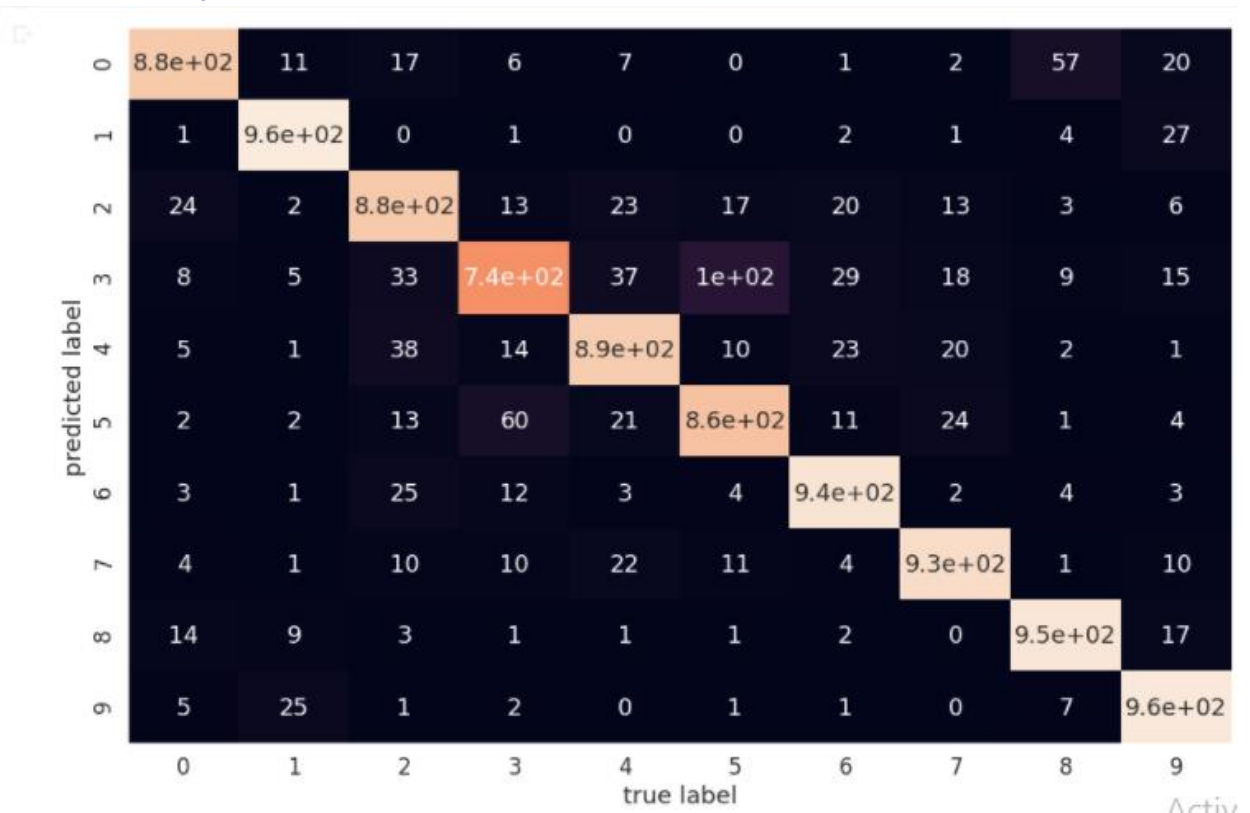


Figure 27 : the confusion matrix of the last model

The confusion matrix is showing some very good predictions, in most classes the number of FN and FP is very small.

## 4. Making predictions:

Now it's time to make predictions with our model, first we create a dictionary that contains our 10 classes:

```
[ ] classes = [0,1,2,3,4,5,6,7,8,9]
    class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

    d = dict(zip(classes, class_names))
    d

{0: 'airplane',
 1: 'automobile',
 2: 'bird',
 3: 'cat',
 4: 'deer',
 5: 'dog',
 6: 'frog',
 7: 'horse',
 8: 'ship',
 9: 'truck'}
```

Figure 28 : creating a dictionary of classes

We choose an image from the test set and we make a prediction of what it is, and the result is shown below:

```
[ ] my_image = x_test[30]

    plt.imshow(my_image)

<matplotlib.image.AxesImage at 0x7f01e9e4aad0>
0
10
20
30
0 10 20 30
```



```
[ ] input_img = my_image.reshape(1,32,32,3)

    predictions = np.argmax(model6.predict(input_img), axis=-1)[0]

    print(f"True class: {d[y_test[30][0]]} \n\nPredicted class: {d[predictions]}")

True class: frog
Predicted class: frog
```

Figure 29 : choosing an image from the test set



And then we wrote a code to make more predictions: (the results were very good)

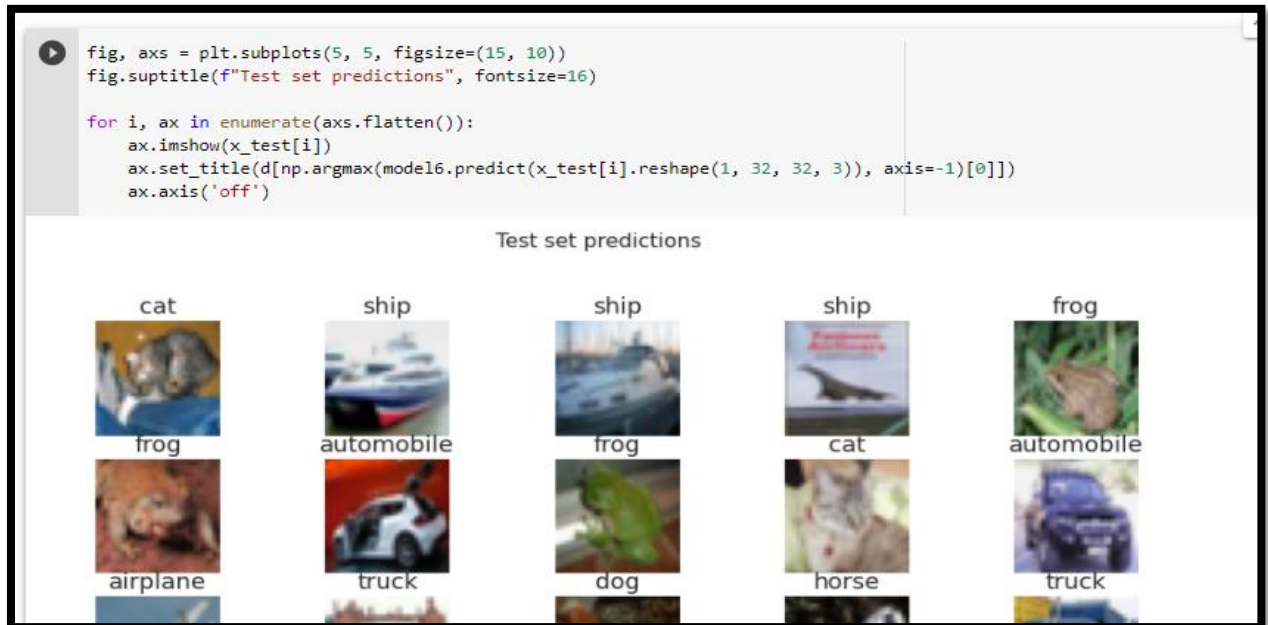


Figure 30 : an additional code for more predictions

## 5. Saving our model:

Now the final step is to save our model in order to reuse it when we shut down Colab without training the model again, and also to use it in the deployment phase.

To save our model we use `model.save` and we give it a name and it will be saved.

And to load it we can call `keras.models` and use `load_model` function.



Figure 31 : codes for saving and loading the model

## Conclusion:

Our model gave some very good statistics, and it was also good when we applied the test set on it, but will it be that good if we test it using other images with different dimensions?

That's what we will discover in the next chapter.

## Chapter IV

# The deployment of the CNN model

### Introduction

Until now, we have built an excellent model, which gives good predictions, with an accuracy of 90%. But in order to make a prediction with this model, the user must know how the code works which is not applicable. For that reason, we have decided to deploy our model on a web application.

We have used Flask to build our web application. [Flask](#) is a web framework. This means flask provides us with tools, libraries and technologies that allow us to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.

**Flask** is the framework here, while it is a Python class datatype. In other words, **Flask** is the prototype used to create instances of a web application or web applications if we want to put it simple.

### 1. Back-End:

In this part we have developed the server side of our web application and everything that communicates between the local files and the browser. Explicitly all the logic of our application. We will explain the most important parts of our back-end side application. So, first we import all the libraries that we will need:

```
1 import os
2 import uuid
3 import flask
4 import urllib
5 from PIL import Image
6 from tensorflow.keras.models import load_model
7 from flask import Flask , render_template , request , send_file
8 from tensorflow.keras.preprocessing.image import load_img , img_to_array
```

Figure 32 :Importing the libraries

The OS module provides functions for interacting with the operating system. This module provides a portable way of using operating system dependent functionalities.

UUID, Universal Unique Identifier, is a python library which helps in generating random objects of 128 bits as IDs. It provides the uniqueness as it generates IDs on the basis of time, Computer hardware (MAC...), the utility of this API will be explained in the next part.

Urllib module is the URL handling module for python. It is used to fetch URLs (Uniform Resource Locators). It uses the `urlopen` function and is able to fetch URLs using a variety of different protocols.

PIL Library adds support for opening, manipulating, and saving many different image file formats.

And finally, we import the tensorflow API in order to load our final trained model.

So, once we import Flask, we need to create an instance of the Flask class for our web app. That's what this line makes:

```
10 app = Flask(__name__)
```

Figure 33 : creating an instance of the Flask class

Then we upload our CNN model by using the predefined method 'load\_model' of the tensorflow API, as shown in these screenshot:

```
13 BASE_DIR = os.path.dirname(os.path.abspath(__file__))
14 model = load_model(os.path.join(BASE_DIR , 'cifar10_model.h5'))
```

```
25 def predict(filename , model):
26     img = load_img(filename , target_size = (32 , 32))
27     img = img_to_array(img)
28     img = img.reshape(1 , 32 ,32 ,3)
29
30     img = img.astype('float32')
31     img = img/255.0
32     result = model.predict(img)
33
34     dict_result = {}
35     for i in range(10):
36         dict_result[result[0][i]] = classes[i]
37
38     res = result[0]
39     res.sort()
40     res = res[::-1]
41     prob = res[:3]
42
43     prob_result = []
44     class_result = []
45     for i in range(3):
46         prob_result.append((prob[i]*100).round(2))
47         class_result.append(dict_result[prob[i]])
48
49     return class_result , prob_result
```

```
63 if request.method == 'POST':
64     if (request.form):
65         link = request.form.get('link')
66         try :
67             resource = urllib.request.urlopen(link)
68             unique_filename = str(uuid.uuid4())
69             filename = unique_filename+'.jpg'
70             img_path = os.path.join(target_img , filename)
71
72             output = open(img_path , "wb")
73             |
74             output.write(resource.read())
75             output.close()
76             img = filename
77
78             class_result , prob_result = predict(img_path , model)
79
80 predictions = {
81     "class1":class_result[0],
82     "class2":class_result[1],
83     "class3":class_result[2],
84     "prob1": prob_result[0],
85     "prob2": prob_result[1],
86     "prob3": prob_result[2],
87 }
```

Figure 34 : uploading the model

After that we define two principal functions in order to organize and optimize our code. The first one is called 'success'. This function uploads the user image and it makes a copy in a specific directory, then it calls the predict function. And finally, it returns a new web page "success.html " in which we display the prediction result with different probabilities. The second function is called "predict", it takes the chosen image and our trained model as arguments. Then it normalizes that image in order to get a good prediction. Finally, it returns the image category with some other data.

After that, we define a function that returns a web html page. That function is mapped to the home `'/'` URL. That means when the user navigates to the index page, the home function will run and it will return its output on the webpage. If the input to the route method was something else, let's say `'/something/'`, the function output would be shown when the user visits `'index.html/something/'`.

```
54 @app.route('/')
55 def home():
56     return render_template("index.html")
57
58 @app.route('/success' , methods = ['GET' , 'POST'])
```

Figure 35 : definition of the home function

## 2. Front-end

In this part we have developed the interface design and everything with which the user interacts.

Here is the home page:



Figure 36 : the home page of the application

The user has to choose an image that belongs to the following categories: Automobile, horse, deer, frog, airplane, dog, cat, truck, ship, bird. Because our model was trained on CIFAR-10 dataset and it performs good predictions when the input satisfies the first condition.

The user has two choices to upload his image. The first is by navigating in his file system. And the second is by entering the image URL so that it could be fetched online from the internet.

When we upload an image, we will be directed to a new web page in which we display the chosen image with the predicted classes: for example, if the user chooses this image:



Figure 37 : an example of an image that will be predicted

The next page will display the following results:

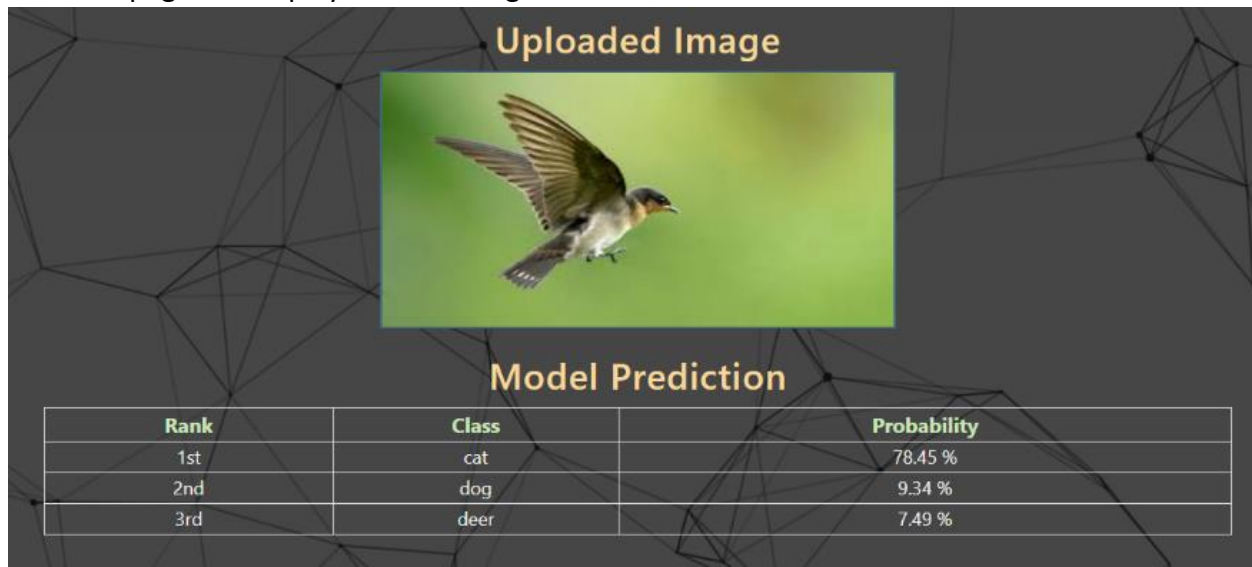


Figure 38 : the result of the prediction of figure 37

We can see here that our model gives the right class with 78.45%. and it suggest two other possible classes with low probabilities for example in the second rank we get the dog class with 9.34%, and in the third rand we have the deer class with 7.49%.


In order to see our model performance on real images. We have tested it with o lot of images that we took from the web and here are some samples:



- Truck class:



Uploaded Image




Model Prediction

Rank	Class	Probability
1st	truck	65.25 %
2nd	automobile	34.58 %
3rd	frog	0.11 %

- Dog class:



Uploaded Image




Model Prediction

Rank	Class	Probability
1st	dog	99.95 %
2nd	bird	0.04 %
3rd	horse	0.01 %

- Horse class:



Uploaded Image




Model Prediction

Rank	Class	Probability
1st	horse	77.2 %
2nd	frog	15.16 %
3rd	deer	3.84 %

- *Airplane class:*



Uploaded Image




Model Prediction

Rank	Class	Probability
1st	airplane	99.69 %
2nd	bird	0.29 %
3rd	automobile	0.01 %

- *Ship class:*



Uploaded Image



Model Prediction

Rank	Class	Probability
1st	ship	98.44 %
2nd	airplane	0.58 %
3rd	truck	0.43 %

## Conclusion:

As we can see in this chapter, our model gives the right predictions with the different images, and it shows the correspondent probabilities. We have also tried a lot of images from the different classes with various shapes and always get the expected outputs with big probabilities, and we can't wait to demonstrate that. Finally, we can conclude that our model is performant.