# IMAGE CLASSIFICATION USING CONVOLUTION NEURAL NETWORK

**PROJECT REPORT IN**
**NEURAL NETWORK AND FUZZY CONTROL**
**FOR THE COMPLETION OF J COMPONENT**
**BY**

**SHUBHAM MITTAL, 19BEE0286,**
**YASHWARDHANA INDORIA, 19BEE0304,**
**HARSHIT KISHORE, 19BEE0306,**

Under the guidance of
**Dr. MONICA SUBASHINI M,**
**Associate Professor, School of Electrical Engineering,**
**(VIT Vellore)**

## ACKNOWLDEGEMENT

We would like to express our deep and sincere gratitude towards our teacher, Dr. Monica Subashini M, Associate Professor, School of Electrical Engineering, VIT Vellore, for giving us the golden opportunity to do the project titled: Image Classification Using Convolution Neural Network and providing invaluable guidance throughout the project. It was a great privilege and honor to do the project under her guidance.

She helped us in this endeavor and has always been very cooperative and without her help, cooperation, guidance and encouragement, the project couldn't have been what it evolved to be.

## CONTENTS


1. ABSTRACT …… 4
2. INTRODUCTION
   - I. Literature Survey …… 5
   - II. Why CNN …… 6
3. METHODOLOGY
   - I. DATASET DESCRIPTION …… 7
   - II. SAMPLE DATA …… 7
4. IMPLEMENTATION
   - I. SOFTWARE DETAILS …… 11
   - II. PSEUDOCODE …… 11
5. RESULTS …… 16
6. CONCLUSION …… 22
7. REFERENCES …… 23

## ABSTRACT

Convolution Neural Network (CNN) have been widely used in automatic image classification systems. Image classification requires generation of features capable of detecting image patterns informative of group identity. A variety of image data sets are available to test the performance of different types of CNN's. The commonly found benchmark datasets for evaluating the performance of a convolution neural network are CIFAR -10, CIFAR-100 and MNIST image data sets. One of the most popular data sets, CIFAR-10 has been taken to test the performance of the network. The main objective of this study is to get a better test accuracy with reduced overfitting which was achieved with a deeper network. The deep learning models are implemented with the use of Keras library available for Python programming language. An accuracy of around 96.99% is expected after running 50 epochs.

- Images can be classified and kept in separate folders which will avoid confusion and will be easier to locate.
- Automatic face recognition and object recognition can be used for classifying the images automatically.
- A lot of tasks can be automated using image recognition like processing cheques in banks, computer vision, space exploration, processing signal images etc.
- Various application in industrial, scientific and medical.

# INTRODUCTION

Image classification is an important topic in artificial vision systems, and has drawn a significant amount of interest over the last decades. This field aims to classify an input image based on visual content.

The ability to classify things correctly requires many hours of training. People get things wrong many times, until eventually, they get it right. The same structure applies to machine learning. By using a high-quality set of data, deep learning can classify objects comparatively well or even better than humans can. With achieving utterly accurate image classifier, some of the monotonous jobs could be replaced by machines, so that humanity could focus on the most enjoyable activities.

It is the core foundation for bigger problems such as Computer Vision, Face Recognition System, or Self-driving car. With the development of deep Convolutional Neural Network (CNN), researchers have achieved good performance on the image recognition task.

## ➢ Literature Survey

The CIFAR-10 dataset is a collection of images that are commonly used to train machine learning and computer vision algorithms. It is one of the most widely used datasets for machine learning research. It consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
[Image Classification – Deep Learning Project in Python with Keras]

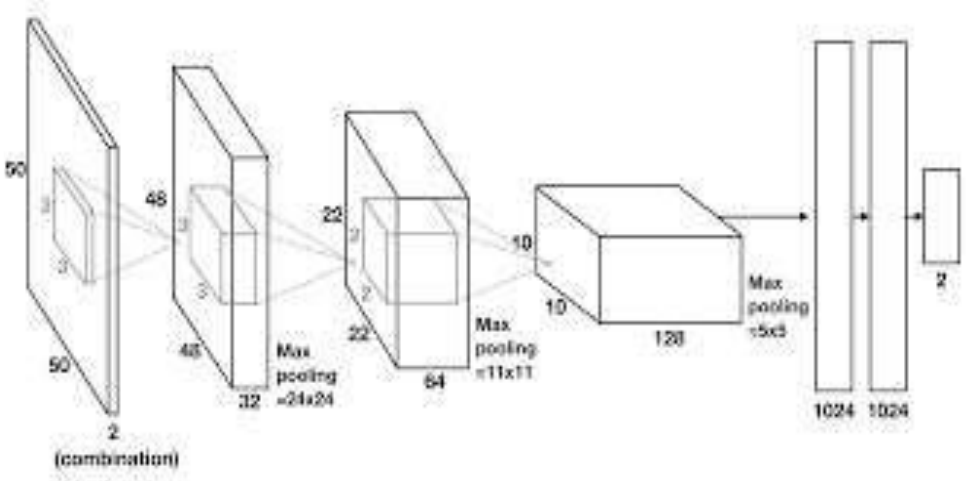
The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

The traditional convolutional neural network usually initializes the weights of all network layers at one time before network training, and then updates the weights of the network by back-propagation algorithm to improve the accuracy of the network during network training. However, with the increase of network depth, the computational cost of this method will increase dramatically and the test accuracy will be affected. In order to solve this problem, a method of gradually reinitializing the weights of each layer is proposed, that is, after a certain training period, the weight of the previous layer is determined and remain unchanged, then initialize the weights of all subsequent layers, repeat this step until the weights of all layers are determined. In order to verify the performance of the method, a series of experiments were carried out on the CIFAR10 dataset. The results show that the accuracy of the network is improved by 9% and the training time is reduced by 29%. It shows that the method can improve the accuracy of the network and reduce the training time. [Improvement in Convolutional Neural Network for CIFAR-10 Dataset Image Classification]

Training neural networks is a computationally challenging problem that requires significant time efforts. Two approaches are proposed that improve efficiency of this task by actively selecting most relevant points from a training data set. The first approach forms a batch that maximizes the reduction of the estimator's entropy, while the second approach only trains on datapoints whose predicted probability is below a predetermined threshold. Both techniques rely on data metrics to speed up training while retaining the epoch based neural network training framework. The results demonstrate that the proposed methods enable significant reduction of training time in experiments on the CIFAR10 dataset without compromising the accuracy. [Improvement in Convolutional Neural Network for CIFAR-10 Dataset Image Classification]

## ➢ Why CNN

CNN often shortened as ConvNets, are complex feed forward neural networks. CNN's are used for image classification and recognition because of its high accuracy. It was inspired from the human visual perception of recognizing things. The CNN follows a hierarchical model which works on building a network, like a funnel, and finally gives out a fully-connected layer where all the neurons are connected to each other and the output is processed.




**Basic CNN Architecture**

## METHODOLOGY

### ➢ Dataset description

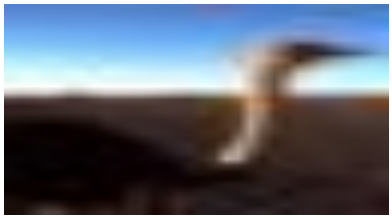
The CIFAR-10 dataset is a collection of images that are commonly used to train machine learning and computer vision algorithms. It is one of the most widely used datasets for machine learning research. The CIFAR-10 dataset contains 60,000 32x32 colour images in 10 different classes. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 6,000 images of each class.
Computer algorithms for recognizing objects in photos often learn by example. CIFAR-10 is a set of images that can be used to teach a computer how to recognize objects. Since the images in CIFAR-10 are low-resolution (32x32), this dataset can allow researchers to quickly try different algorithms to see what works. Various kinds of convolutional neural networks tend to be the best at recognizing the images in CIFAR-10.

The primary objective of this project is to test and compare the performance of various CNN models implemented with the use of scripts written in Python programming language. The deep neural network models are applied on one of the most popular image classification datasets, CIFAR-10 dataset, and afterwards the results are compared with other published solutions.

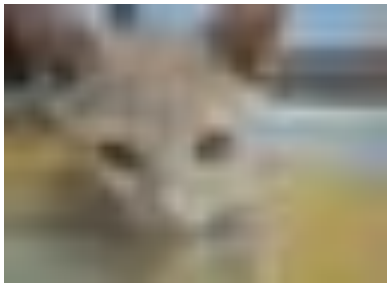
### ➢ Sample Data

**Airplane**

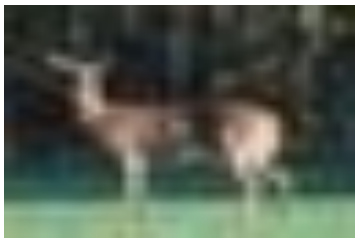
**Automobile**



**Bird**



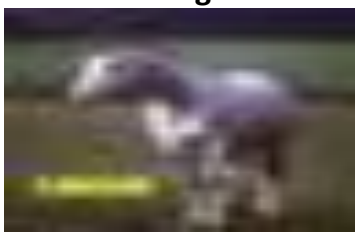
**Cat**



**Deer**

**Dog**

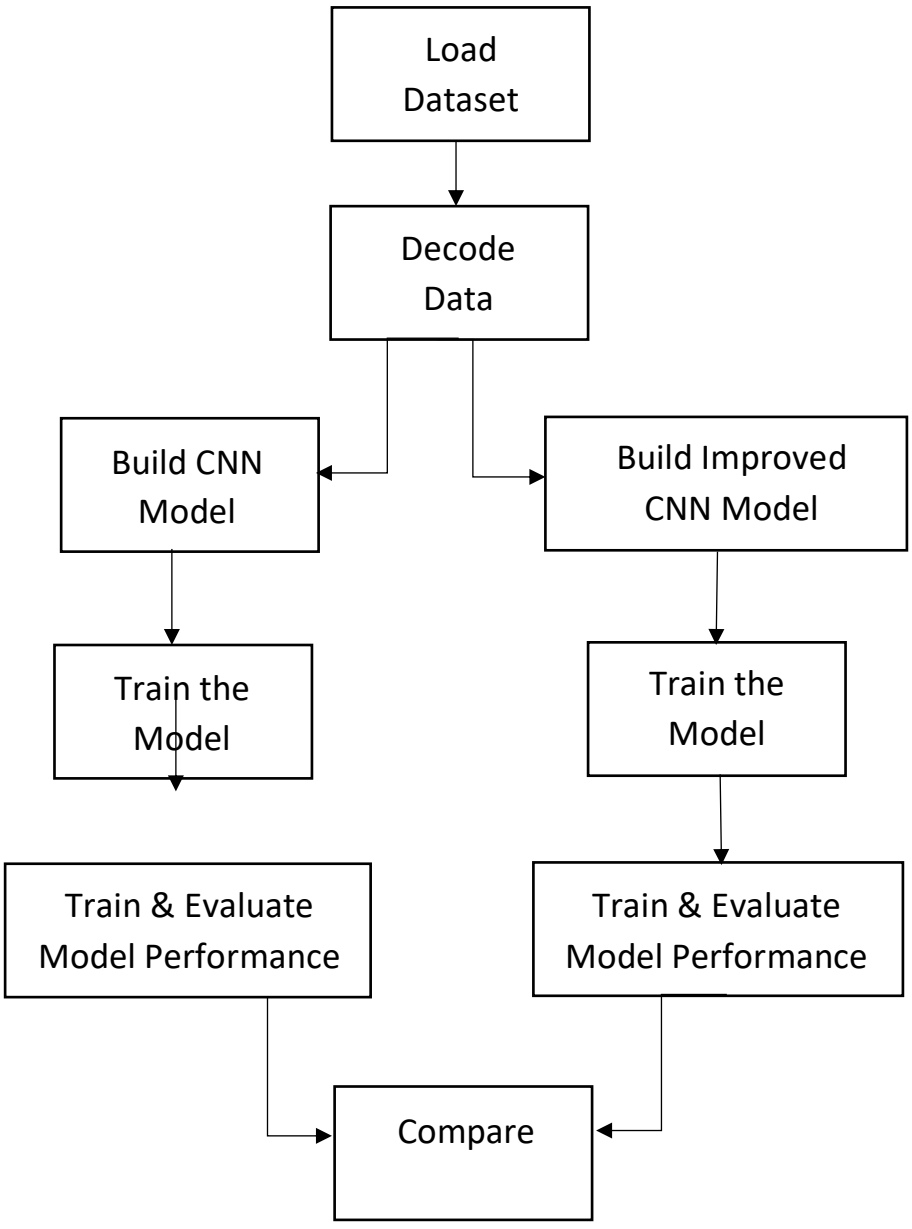
**Frog**



**Horse**

Ship

Truck




**Block Diagram**

Step 1: The CIFAR-10 dataset is downloaded from the official website

Step 2: The dataset is decoded with the use of a Python script to see the actual images and use them.

Step 3: Two CNN models are built and one is a training model with the use of training dataset.

Step 4: The model is tested with the use of a test dataset and the performances of both models are evaluated and compared.

# IMPLEMENTATION

## ➢ Software Details

In order to run CNN models, Python requires the following packages:

- Tensorflow 1.7.0
- Keras 2.1.5
- Pickle (Python's built-in package)
- NumPy 1.14.2
- Matplolib 2.2.2

## ➢ Pseudocode

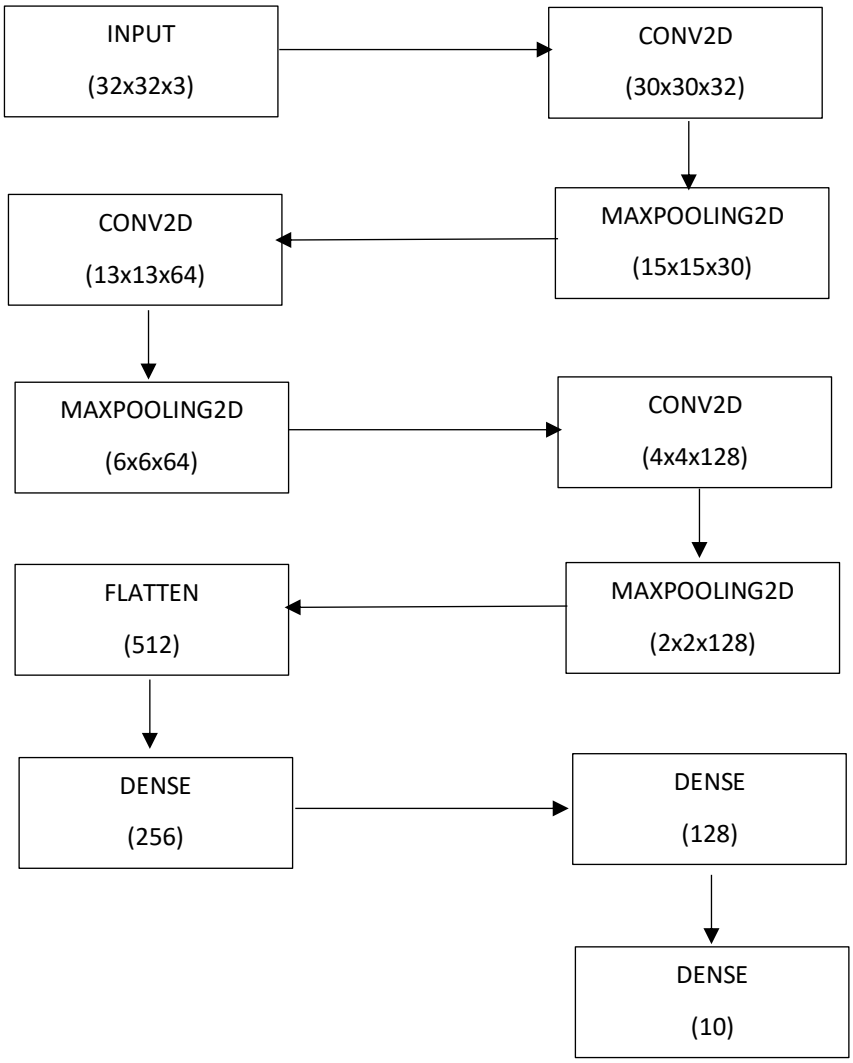
```
#!/usr/bin/env python
# coding: utf-8
# ### Import Libraries
# In[ ]:
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2
D
from tensorflow.keras.losses import sparse_categorical_crossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Model
import matplotlib.pyplot as plt
from keras import callbacks
import random
import matplotlib.pyplot as plt
import cv2
import numpy as np
# ### Model configuration
# In[ ]:
batch_size = 50
img_width, img_height, img_num_channels = 32, 32, 3
loss_function = sparse_categorical_crossentropy
no_classes = 10
no_epochs = 50
optimizer = Adam()
validation_split = 0.2
verbosity = 1
earlystopping = callbacks.EarlyStopping(monitor ="val_loss",mode ="min"
, patience = 40,restore_best_weights = True)
# ### Load CIFAR-10 data
# In[ ]:
(input_train, target_train), (input_test, target_test) = cifar10.load_d
ata()
# ### Display images with labels
# In[ ]:
```

```
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer','dog', '
frog', 'horse', 'ship', 'truck']
x = random.sample(range(1,600), 25)
plt.figure(figsize=(10,10))
for i in range(25):j=x[i]
plt.subplot(5,5,i+1)
plt.xticks([])
plt.yticks([])
plt.grid(False)
plt.imshow(input_train[j], cmap=plt.cm.binary)
# The CIFAR labels happen to be arrays,
# which is why you need the extra index
plt.xlabel(class_names[target_train[j][0]])
plt.show()
# ### Determine shape of the data
# In[ ]:
input_shape = (img_width, img_height, img_num_channels)
# ### Parse numbers as floats
# In[ ]:
input_train = input_train.astype('float32')
input_test = input_test.astype('float32')
# ### Normalize data
# In[ ]:
input_train = input_train / 255
input_test = input_test / 255
# ### Create the model
# In[ ]:
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape
=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(no_classes, activation='softmax'))
model.summary()
# ### Compile the model
# In[ ]:
model.compile(loss=loss_function,
optimizer=optimizer,
metrics=['accuracy'])
# ### Fit data to model
# In[ ]:
history = model.fit(input_train, target_train,
```

```
batch_size=batch_size,
epochs=no_epochs,
verbose=verbosity,
validation_split=validation_split,
callbacks =[earlystopping])
# ### Generate generalization metrics
# In[ ]:
score = model.evaluate(input_test, target_test, verbose=0)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')
# ## Visualize history
# ### Plot history: Loss
# In[ ]:
plt.plot(history.history['val_loss'], label="validation loss")
plt.plot(history.history['val_accuracy'], label ="validation accuracy")
plt.title('Validation loss history')
plt.ylabel('Accuracy and Loss')
plt.xlabel('Epochs')
plt.legend(loc='upper left')
plt.show()
# ### Plot Confusion Matrix
# In[ ]:
# Plot confusion matrix
from sklearn.metrics import confusion_matrix
import itertools
plt.rcParams['figure.figsize'] = [10,7]
def plot_confusion_matrix(cm, classes,normalize=True,title='Confusion m
atrix',cmap=plt.cm.Blues):
  if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    print("Normalized confusion matrix")
    fmt = '.2f'
  else:
    print('Confusion matrix, without normalization')
    fmt = 'd'
  print(cm)
  plt.imshow(cm, interpolation='nearest', cmap=cmap)
  plt.title(title)
  plt.colorbar()
  tick_marks = np.arange(len(classes))
  plt.xticks(tick_marks, classes, rotation=45)
  plt.yticks(tick_marks, classes)

  thresh = cm.max() / 2.
  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1]))
:
    plt.text(j, i, format(cm[i, j], fmt),
    horizontalalignment="center",
    color="white" if cm[i, j] > thresh else "black")
```

```
  plt.tight_layout()
  plt.ylabel('True label')
  plt.xlabel('Predicted label')
  plt.show()
p_test = model.predict(input_test).argmax(axis=1)
cm = confusion_matrix(target_test, p_test)
plot_confusion_matrix(cm, list(range(10)))
for e in range(10):
  print("{e} - {class_names[e]}\n")
# ### Show Correct Precition
# In[ ]:
misclassified_idx = np.where(p_test == target_test)[0]
i = np.random.choice(misclassified_idx)
imgnew=input_test[i] # Reading an Image
input_imgnew= np.expand_dims(imgnew, axis=0)
plt.style.use("dark_background")
plt.imshow(imgnew, cmap=plt.cm.binary)
plt.title("True label: %s Predicted: %s" %(class_names[target_test[i][0
]],
class_names[p_test[i]]));
# ### Show Incorrect Prediction
# In[ ]:
misclassified_idx = np.where(p_test != target_test)[0]
i = np.random.choice(misclassified_idx)
imgnew=input_test[i] # Reading an Image
input_imgnew= np.expand_dims(imgnew, axis=0)
plt.style.use("dark_background")
plt.imshow(imgnew, cmap=plt.cm.binary)
plt.title("True label: %s Predicted: %s" %(class_names[target_test[i][0
]],
class_names[p_test[i]]));
```

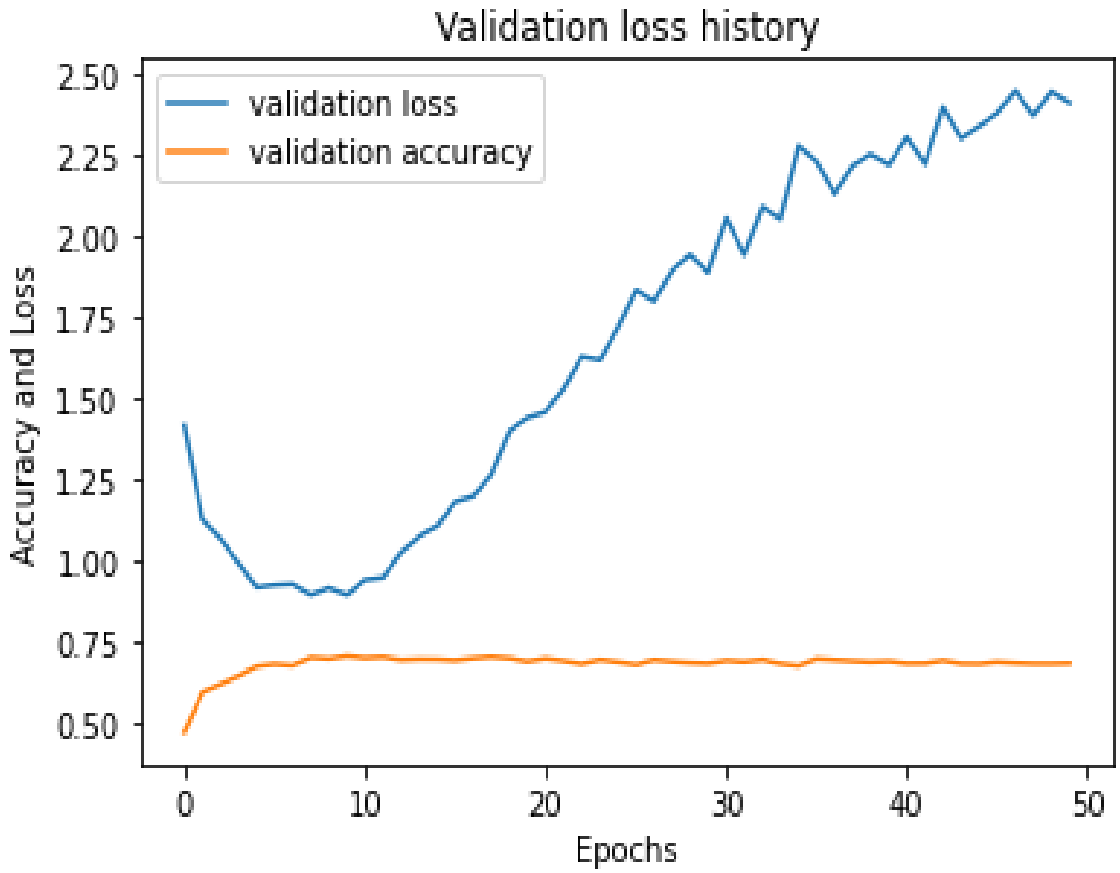



**FLOW CHART**

Our model is made of several layer which include: -

- 3 2DConvolution Layers with ReLU as activation function.
- 3 2DMaxPooling Layers with both kernel size and stride as (2,2).
- 1 Flatten Layer to flatten the Input.
- 2 Dense Layers with ReLU as activation function.
- 1 Dense Layer with softmax as activation which is our final layer as well as Output Layer.

## RESULTS

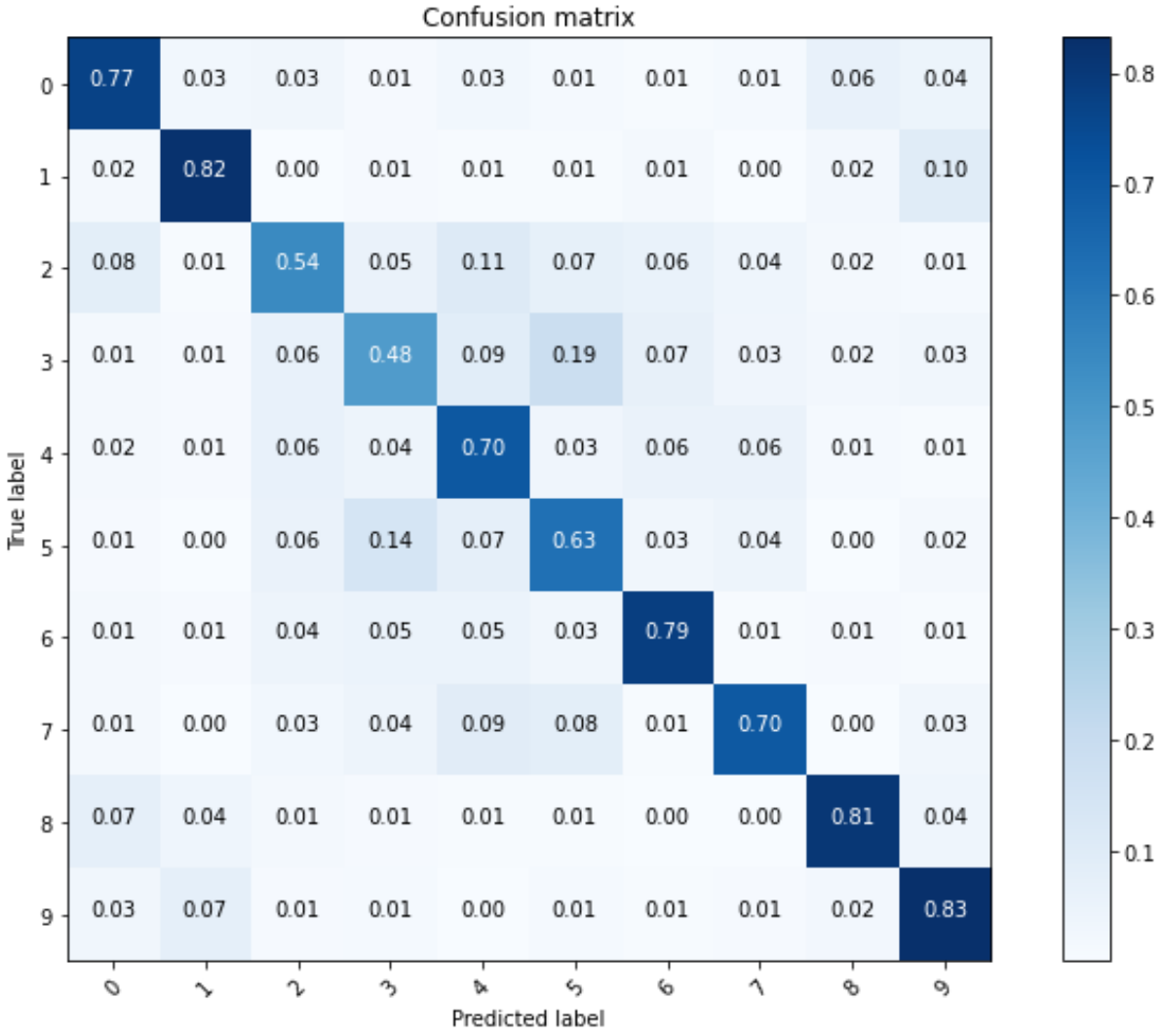



**Validation loss** is the same metric as training **loss**, but it is not used to update the weights. If **validation loss** > **training loss** you can call it some overfitting.

If **validation loss** < **training loss** you can call it some underfitting. If **validation loss** << **training loss** you can call it underfitting. Your aim is to make the **validation loss** as low as possible. Some overfitting is nearly always a good thing.

It also seems that the validation loss will keep going up if I train the model for more epochs.

In other words, the test (or testing) **accuracy** often refers to the **validation accuracy**, that is, the **accuracy** you calculate on the data set you do not use for training, but you use (during the training process) for validating (or "testing") the generalisation ability of your model or for "early stopping".




A **Confusion matrix** is an N x N **matrix** also known as an error matrix used for evaluating the performance of a classification model, where N is the number of target classes. The **matrix** compares the actual target values with those predicted by the **machine learning** model.

The confusion matrices discussed above have only two conditions: positive and negative.

Each row in a confusion matrix represents an actual class, while each column represents a predicted class.

- It evaluates the performance of the classification models, when they make predictions on test data, and tells how good our classification model is.
- It not only tells the error made by the classifiers but also the type of errors such as it is either type-I or type-II error.

- With the help of the confusion matrix, we can calculate the different parameters for the model, such as accuracy, precision, etc.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 256)               131328
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 258,762
Trainable params: 258,762
Non-trainable params: 0
```

```
_________________________________________________________________
Epoch 1/50
800/800 [==============================] - 63s 77ms/step - loss: 1.6331 - accuracy: 0.3918 - val_loss: 1.4192 - val_accuracy: 0.4758
Epoch 2/50
800/800 [==============================] - 62s 78ms/step - loss: 1.2322 - accuracy: 0.5555 - val_loss: 1.1314 - val_accuracy: 0.5966
Epoch 3/50
800/800 [==============================] - 62s 77ms/step - loss: 1.0746 - accuracy: 0.6162 - val_loss: 1.0702 - val_accuracy: 0.6228
Epoch 4/50
800/800 [==============================] - 61s 77ms/step - loss: 0.9533 - accuracy: 0.6629 - val_loss: 0.9959 - val_accuracy: 0.6475
Epoch 5/50
800/800 [==============================] - 61s 77ms/step - loss: 0.8631 - accuracy: 0.6933 - val_loss: 0.9202 - val_accuracy: 0.6781
Epoch 6/50
800/800 [==============================] - 62s 77ms/step - loss: 0.7906 - accuracy: 0.7208 - val_loss: 0.9274 - val_accuracy: 0.6846
Epoch 7/50
800/800 [==============================] - 62s 77ms/step - loss: 0.7244 - accuracy: 0.7425 - val_loss: 0.9307 - val_accuracy: 0.6801
Epoch 8/50
800/800 [==============================] - 62s 77ms/step - loss: 0.6621 - accuracy: 0.7655 - val_loss: 0.8973 - val_accuracy: 0.7049
Epoch 9/50
800/800 [==============================] - 62s 77ms/step - loss: 0.6024 - accuracy: 0.7869 - val_loss: 0.9186 - val_accuracy: 0.7011
Epoch 10/50
800/800 [==============================] - 61s 77ms/step - loss: 0.5570 - accuracy: 0.8010 - val_loss: 0.8971 - val_accuracy: 0.7095
Epoch 11/50
800/800 [==============================] - 61s 77ms/step - loss: 0.5045 - accuracy: 0.8210 - val_loss: 0.9436 - val_accuracy: 0.7037
Epoch 12/50
800/800 [==============================] - 61s 77ms/step - loss: 0.4565 - accuracy: 0.8370 - val_loss: 0.9511 - val_accuracy: 0.7064
Epoch 13/50
800/800 [==============================] - 61s 77ms/step - loss: 0.4175 - accuracy: 0.8509 - val_loss: 1.0280 - val_accuracy: 0.6974
Epoch 14/50
800/800 [==============================] - 62s 77ms/step - loss: 0.3789 - accuracy: 0.8643 - val_loss: 1.0775 - val_accuracy: 0.6997
Epoch 15/50
800/800 [==============================] - 62s 77ms/step - loss: 0.3430 - accuracy: 0.8776 - val_loss: 1.1098 - val_accuracy: 0.6992
Epoch 16/50
800/800 [==============================] - 62s 77ms/step - loss: 0.3114 - accuracy: 0.8898 - val_loss: 1.1833 - val_accuracy: 0.6959
Epoch 17/50
800/800 [==============================] - 62s 77ms/step - loss: 0.2791 - accuracy: 0.8993 - val_loss: 1.2015 - val_accuracy: 0.7028
```

```
Epoch 18/50
800/800 [==============================] - 62s 77ms/step - loss: 0.2580 - accuracy: 0.9071 - val_loss: 1.2668 - val_accuracy: 0.7069
Epoch 19/50
800/800 [==============================] - 62s 78ms/step - loss: 0.2252 - accuracy: 0.9179 - val_loss: 1.4020 - val_accuracy: 0.7025
Epoch 20/50
800/800 [==============================] - 62s 78ms/step - loss: 0.2186 - accuracy: 0.9215 - val_loss: 1.4427 - val_accuracy: 0.6926
Epoch 21/50
800/800 [==============================] - 62s 77ms/step - loss: 0.2027 - accuracy: 0.9261 - val_loss: 1.4614 - val_accuracy: 0.7020
Epoch 22/50
800/800 [==============================] - 62s 78ms/step - loss: 0.1882 - accuracy: 0.9323 - val_loss: 1.5311 - val_accuracy: 0.6942
Epoch 23/50
800/800 [==============================] - 62s 77ms/step - loss: 0.1665 - accuracy: 0.9413 - val_loss: 1.6308 - val_accuracy: 0.6847
Epoch 24/50
800/800 [==============================] - 62s 78ms/step - loss: 0.1655 - accuracy: 0.9409 - val_loss: 1.6198 - val_accuracy: 0.6978
Epoch 25/50
800/800 [==============================] - 62s 78ms/step - loss: 0.1546 - accuracy: 0.9445 - val_loss: 1.7179 - val_accuracy: 0.6910
Epoch 26/50
800/800 [==============================] - 63s 78ms/step - loss: 0.1512 - accuracy: 0.9463 - val_loss: 1.8350 - val_accuracy: 0.6826
Epoch 27/50
800/800 [==============================] - 63s 79ms/step - loss: 0.1473 - accuracy: 0.9477 - val_loss: 1.8006 - val_accuracy: 0.6974
Epoch 28/50
800/800 [==============================] - 63s 78ms/step - loss: 0.1269 - accuracy: 0.9550 - val_loss: 1.8962 - val_accuracy: 0.6924
Epoch 29/50
800/800 [==============================] - 62s 78ms/step - loss: 0.1301 - accuracy: 0.9543 - val_loss: 1.9438 - val_accuracy: 0.6884
Epoch 30/50
800/800 [==============================] - 63s 78ms/step - loss: 0.1271 - accuracy: 0.9553 - val_loss: 1.8887 - val_accuracy: 0.6861
Epoch 31/50
800/800 [==============================] - 62s 78ms/step - loss: 0.1131 - accuracy: 0.9606 - val_loss: 2.0569 - val_accuracy: 0.6954
Epoch 32/50
800/800 [==============================] - 63s 79ms/step - loss: 0.1276 - accuracy: 0.9560 - val_loss: 1.9434 - val_accuracy: 0.6915
Epoch 33/50
800/800 [==============================] - 63s 78ms/step - loss: 0.1047 - accuracy: 0.9631 - val_loss: 2.0915 - val_accuracy: 0.6979
Epoch 34/50
800/800 [==============================] - 63s 78ms/step - loss: 0.1183 - accuracy: 0.9586 - val_loss: 2.0539 - val_accuracy: 0.6857
Epoch 35/50
800/800 [==============================] - 62s 78ms/step - loss: 0.1024 - accuracy: 0.9643 - val_loss: 2.2801 - val_accuracy: 0.6790
Epoch 36/50
800/800 [==============================] - 63s 78ms/step - loss: 0.1057 - accuracy: 0.9634 - val_loss: 2.2297 - val_accuracy: 0.7008
Epoch 37/50
800/800 [==============================] - 63s 78ms/step - loss: 0.1013 - accuracy: 0.9652 - val_loss: 2.1303 - val_accuracy: 0.6972
Epoch 38/50
800/800 [==============================] - 62s 78ms/step - loss: 0.1006 - accuracy: 0.9664 - val_loss: 2.2178 - val_accuracy: 0.6939
Epoch 39/50
800/800 [==============================] - 63s 78ms/step - loss: 0.1049 - accuracy: 0.9635 - val_loss: 2.2527 - val_accuracy: 0.6909
Epoch 40/50
800/800 [==============================] - 63s 79ms/step - loss: 0.1023 - accuracy: 0.9660 - val_loss: 2.2196 - val_accuracy: 0.6934
Epoch 41/50
800/800 [==============================] - 63s 79ms/step - loss: 0.0969 - accuracy: 0.9674 - val_loss: 2.3057 - val_accuracy: 0.6861
Epoch 42/50
800/800 [==============================] - 63s 78ms/step - loss: 0.0980 - accuracy: 0.9662 - val_loss: 2.2219 - val_accuracy: 0.6865
Epoch 43/50
800/800 [==============================] - 63s 79ms/step - loss: 0.0833 - accuracy: 0.9719 - val_loss: 2.3974 - val_accuracy: 0.6956
Epoch 44/50
800/800 [==============================] - 63s 79ms/step - loss: 0.0855 - accuracy: 0.9715 - val_loss: 2.3003 - val_accuracy: 0.6856
Epoch 45/50
800/800 [==============================] - 63s 79ms/step - loss: 0.0981 - accuracy: 0.9670 - val_loss: 2.3393 - val_accuracy: 0.6848
Epoch 46/50
800/800 [==============================] - 63s 79ms/step - loss: 0.0909 - accuracy: 0.9698 - val_loss: 2.3790 - val_accuracy: 0.6911
Epoch 47/50
800/800 [==============================] - 63s 79ms/step - loss: 0.0931 - accuracy: 0.9682 - val_loss: 2.4484 - val_accuracy: 0.6876
Epoch 48/50
800/800 [==============================] - 63s 79ms/step - loss: 0.0796 - accuracy: 0.9729 - val_loss: 2.3726 - val_accuracy: 0.6855
Epoch 49/50
800/800 [==============================] - 63s 79ms/step - loss: 0.0841 - accuracy: 0.9723 - val_loss: 2.4463 - val_accuracy: 0.6853
Epoch 50/50
800/800 [==============================] - 63s 79ms/step - loss: 0.0935 - accuracy: 0.9699 - val_loss: 2.4108 - val_accuracy: 0.6866
Test loss: 0.9132522344589233 / Test accuracy: 0.7081000208854675
```


True label: deer Predicted: deer
0
5
10
15
20
25
30
0
5
10
15
20
25
30

**Output**


Actual:dog
Predicted:dog(0.94)


Actual:airplane
Predicted:ship(1.0)

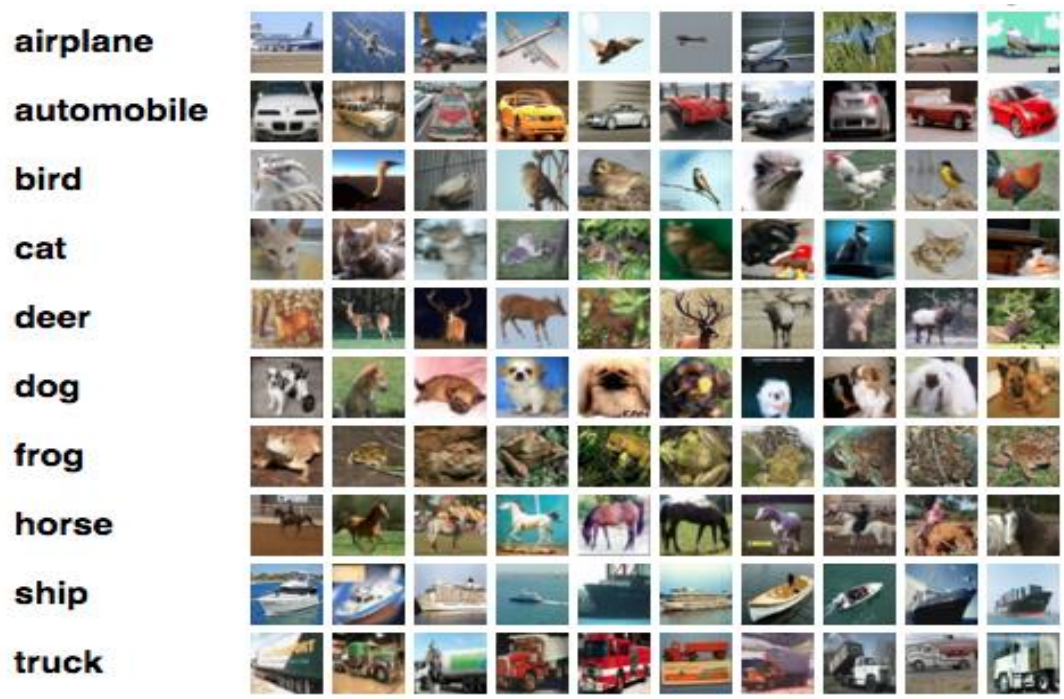

Actual:horse
Predicted:horse(1.0)


Actual:automobile
Predicted:automobile(1.0)





Few other outputs




**Cifar 10 segregates images like this**

**With a greater number of epochs, we get better accuracy as well as better image of our output.**

## CONCLUSION

The work analysed the prediction accuracy of the convolutional neural network (CNN) on the training and test dataset CIFAR10. The study was focused on 10 classes of each dataset only. The main purpose was to find out the accuracy of the network on the CIFAR10 dataset and evaluating the consistency of prediction by CNN.

It can further be summed up that the neural networks are new and the best emerging technique to make a machine intelligent for solving many real-life object categorization problems. Many types of research and works are being done on it.

It has wide applications and is easy and flexible to integrate into various platforms. The hardware requirements may not allow the network to be trained on normal desktop work but just with nominal requirements one can train the network and generate the desired model.

## REFERENCES


1. Akwasi Darkwah Akwaboah “Convolutional Neural Network for CIFAR-10 Dataset Image Classification”, https://www.researchgate.net/publication/337240963 November, 2019.
2. “CIFAR-10 and CIFAR-100 datasets.” [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html. [Accessed: 20-Oct-2019].
3. Neha Sharma,Vibhor Jain, Anju Mishra “An analysis of Convolution Neural Networks for Image Classification in International Conference on Computational Intelligence and Data Science (ICCIDS 2018).
4. Suyesh Pandit, Sushil Kumar “Improvement in Convolutional Neural Network for CIFAR-10 Dataset Image Classification” in International Journal of Computer Applications (0975 – 8887) Volume 176 – No. 37, July 2020.
5. Khoi Hoang, “Image Classification with Fashion – MNIST and CIFAR-10”.
6. Caner Calik, M. Fatih Demirci “Cifar-10 Image Classification with Convolution Neural Networks for Embedded Systems in IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA), October,2018.
7. Junho Yim, Jeongwoo Ju, Heechul Jung, and Junmo Kim “Image Classification Using Convolutional Neural Networks with Multi-stage Feature” Springer International Publishing Switzerland, 2015.
8. Travis Williams, Robert Li “Advanced Image Classification Using Wavelets and Convolutional Neural Networks” in 15th IEEE International Conference on Machine Learning and Applications (ICMLA), February, 2017.
9. https://www.cs.toronto.edu/~kriz/cifar.html
10. Shuying Li, Weihong Deng “Very deep convolution neural network-based image classification using small training sample size” in 3rd IAPR Asian Conference on Pattern Recognition (ACPR), June, 2016.
11. https://data-flair.training/blogs/deep-learning-project-ideas/
12. Yong Wang, Mingyuan Xin “Research on image classification model based on deep convolution neural network” in EURASIP Journal on Image and Video Processing, 2019.
13. Chen Wang, Yang Xi “Convolution Neural Network for Image Classification”.
14. Pankaj Garg, Tushar Jajodia “Image Classification – Cifar-10 Dataset” in International Research Journal of Modernization in Engineering Technology and Science, May, 2020.
15. Xinqi Zhu and Michael Bain (2017, October). B-CNN: Branch Convolutional Neural Network for Hierarchical Classification
16. Tien Ho-Phuoc, CIFAR10 to Compare Visual Recognition Performance between Deep Neural Networks and Humans.
17. https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/
18. https://keras.io/
19. Deepika Jaiswal, Soman Kp, Sowmya Vishvanathan “Image Classification using Convolution neural Networks” in International Journal of Scientific and Engineering Research, June 2014.

20. Md. Anwar Hossain & Md. Shahriar Alam Sajib “Classification of Image using Convolutional Neural Network (CNN)” in Global Journal of Computer Science and Technology, 2019.
21. Yifeng Zhao1, Weimin Lang1 ,Bin Li2 —Performence analysis of neural network with improved weight training process in IEEE 2019.
22. Raniah Zaheer, Humera Shaziya —A Study of the Optimization Algorithms in Deep Learning‖ in IEEE 2019.
23. Sara Mourad, Haris Vikalo and Ahmed Tewfik —ONLINE SELECTIVE TRAINING FOR FASTER NEURAL NETWORK LEARNING in IEEE 2019.
24. Y. Chen, Y. Yang, W. Wang, and C. C. Jay Kuo, —Ensembles of feedforward-designed convolutional neural networks, Jan. 2019.
25. Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, —GPipe: Efficient training of giant neural networks using pipeline parallelism, Nov. 2018.
26. M. Tan and Q. V. Le, —EfficientNet: Rethinking model scaling for convolutional neural networks, May 2019.
27. E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, —AutoAugment: Learning augmentation policies from data, May 2018.
28. N. Nayman, A. Noy, T. Ridnik, I. Friedman, R. Jin, and L. Zelnik-Manor, —XNAS: Neural architecture search with expert advice,June 2019.

## Plagiarism check (open source)