

## Assignment-8 DEEKSHITH ATHMAKUR 700743388

### Neural Networks and Deep learning

**GitHub link:** (<https://github.com/DEEKSHITH-ATHMAKUR/ICP8>)

```
In [11]: import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.datasets import mnist

from tensorflow.keras.optimizers import RMSprop
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout, BatchNormalization

%matplotlib inline
```

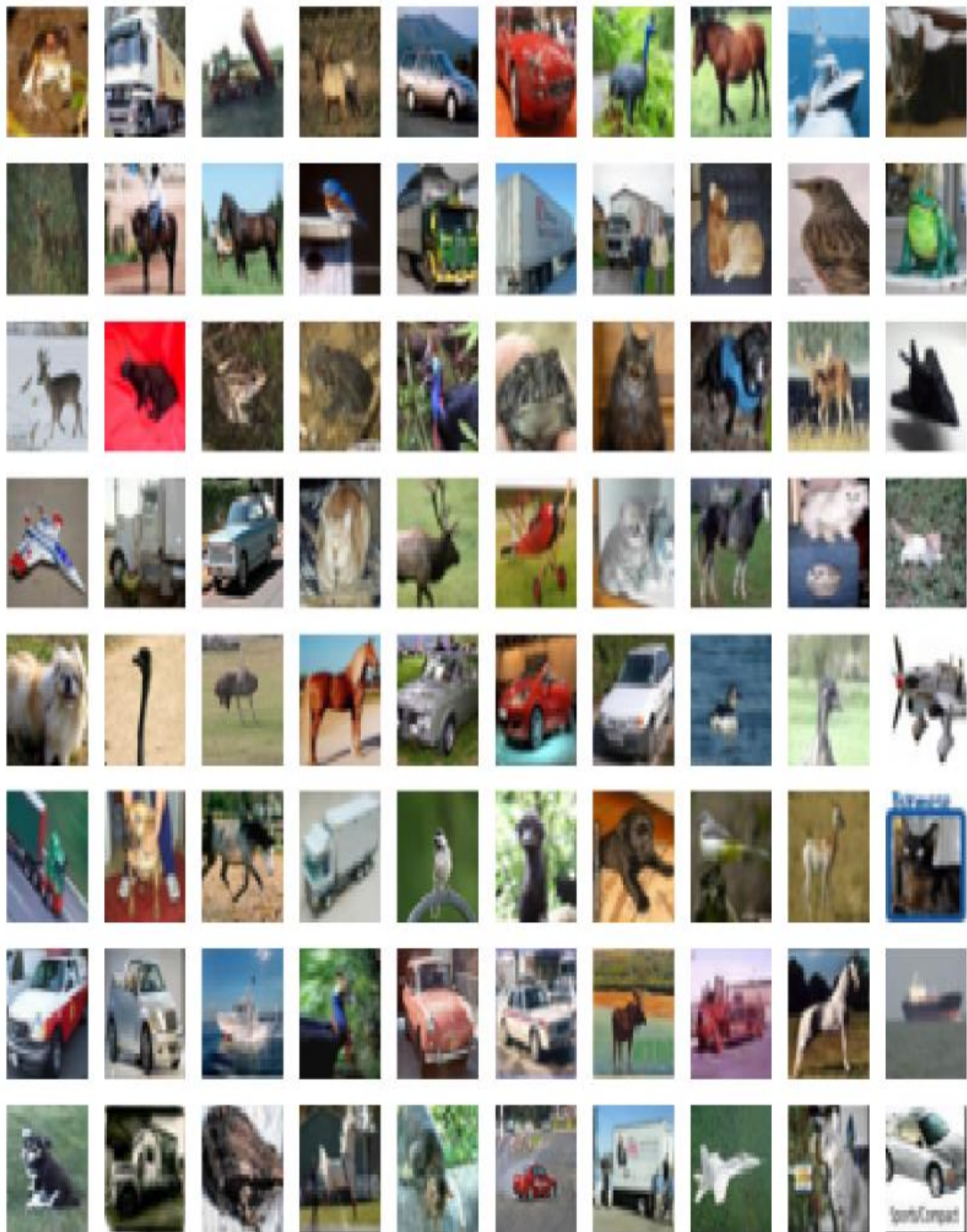
Extract data and train and test dataset

```
In [12]: #cifar100 = tf.keras.datasets.cifar100
(X_train,Y_train) , (X_test,Y_test) = cifar10.load_data()
```

```
In [13]: classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

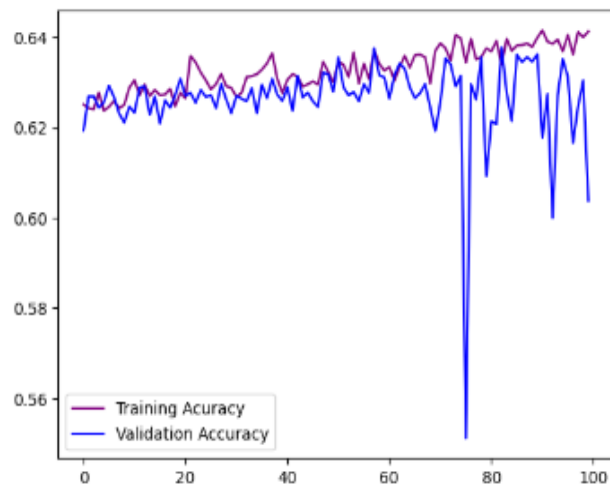
Let's look into the dataset images

```
In [14]: plt.figure(figsize = (16,16))  
for i in range(100):  
    plt.subplot(10,10,1+i)  
    plt.axis('off')  
    plt.imshow(X_train[i], cmap = 'gray')
```





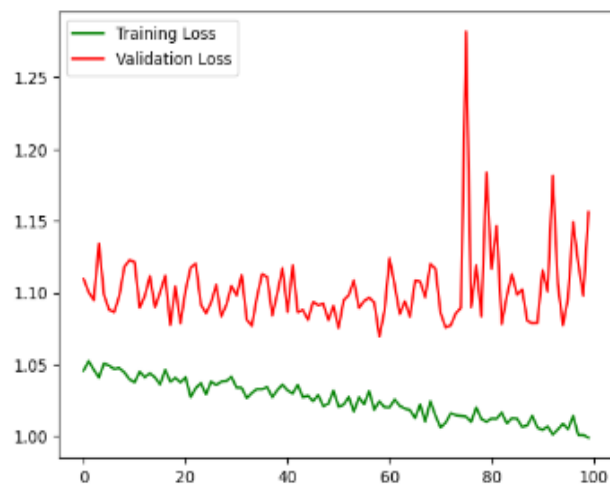
Out[76]: <matplotlib.legend.Legend at 0x7f75101e8160>



```
In [77]: loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure()
plt.plot(loss,color = 'green',label = 'Training Loss')
plt.plot(val_loss,color = 'red',label = 'Validation Loss')
plt.legend()
```

Out[77]: <matplotlib.legend.Legend at 0x7f75101e8d30>



```
In [81]: import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Greens):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting 'normalize=True'.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=30)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

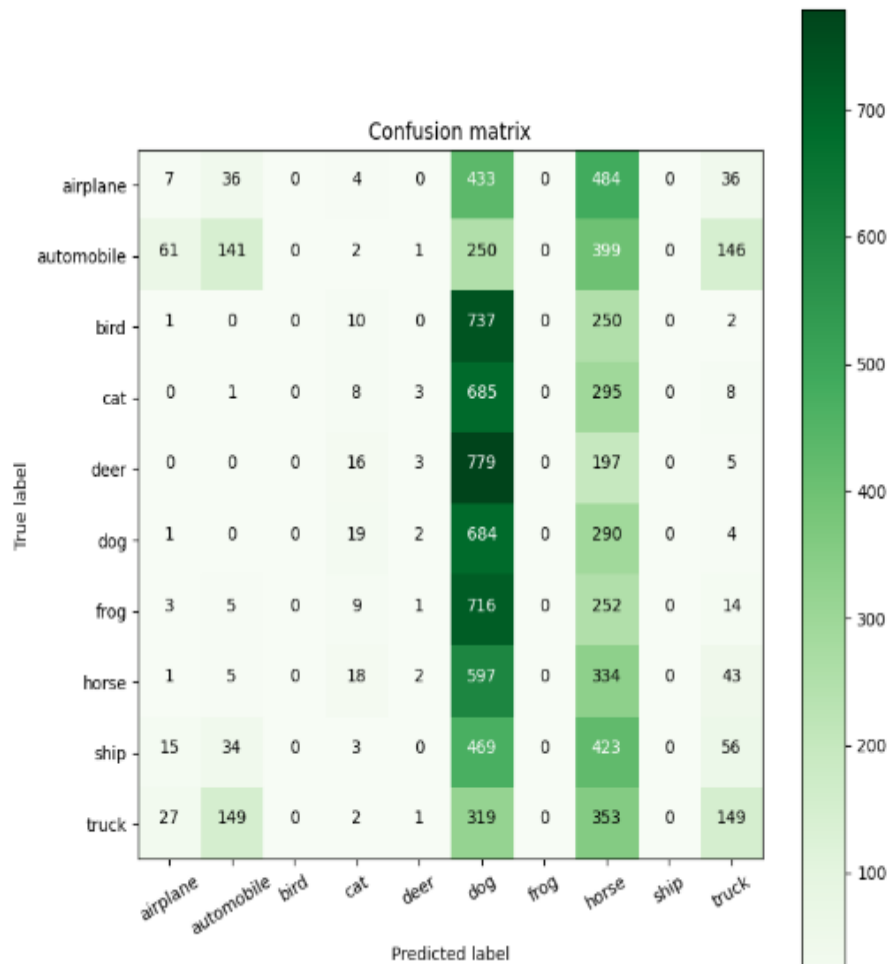
    #print(cm)

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

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

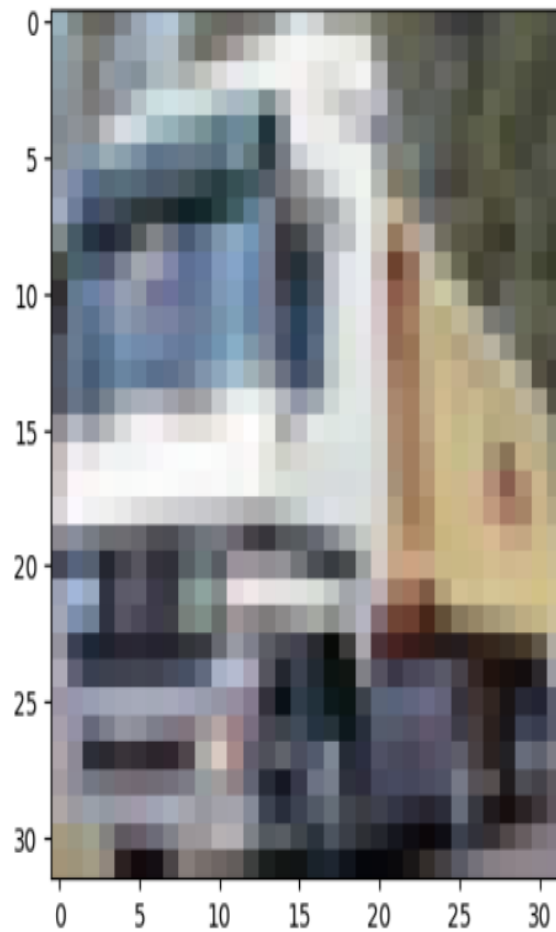
```
In [82]: plt.figure(figsize=(8,8))
plot_confusion_matrix(cm,classes)
```

Confusion matrix, without normalization



```
In [12]: # check data  
plt.imshow(x_train[1])  
print(x_train[1].shape)
```

(32, 32, 3)



```
In [ ]: ▶ model.save("keras-VGG16-cifar10.h5")
plt.imshow(x_test[1000])

result = model.predict(x_test[1000:1001]).tolist()
predict = 0
expect = y_test[1000][0]
for i, _ in enumerate(result[0]):
    if result[0][i] > result[0][predict]:
        predict = i
print("predict class:", predict)
print("expected class:", expect)

1/1 [=====] - 1s 740ms/step
predict class: 5
expected class: 5
```

