Assignment-8 DEEKSHITH ATHMAKUR 700743388

Neural Networks and Deep learning

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

```
In [11]: M 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

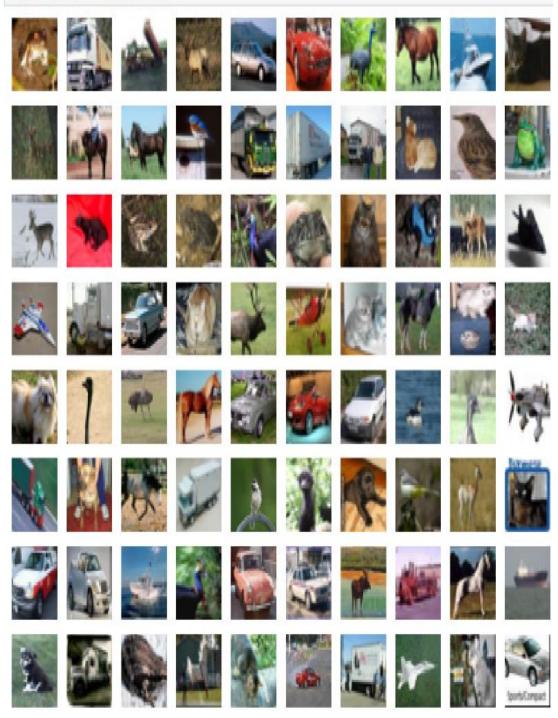
**Mmatplotlib inline**
```

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()
```

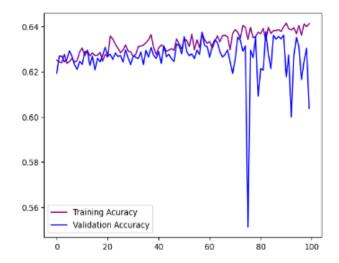
```
In [13]: M classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

```
In [14]: M 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')
```



```
In [15]: M from sklearn.model_selection import train_test_split
            x_train, x_val, y_train, y_val = train_test_split(X_train,Y_train,test_size=0.2)
y_train = to_categorical(y_train, num_classes = 10)
            y_val = to_categorical(y_val, num_classes = 10)
In [17]: M print(x_train.shape)
           print(y_train.shape)
            print(x_val.shape)
            print(y_val.shape)
            print(X_test.shape)
            print(Y_test.shape)
            (40000, 32, 32, 3)
            (40000, 10)
            (10000, 32, 32, 3)
            (10000, 10)
            (10000, 32, 32, 3)
            (10000, 1)
In [18]: | train_datagen = ImageDataGenerator(
               preprocessing_function = tf.keras.applications.vgg19.preprocess_input,
               rotation_range=10,
               zoom_range = 0.1,
               width_shift_range = 0.1,
               height_shift_range = 0.1,
               shear_range = 0.1,
               horizontal_flip = True
            train_datagen.fit(x_train)
            val_datagen = ImageDataGenerator(preprocessing_function = tf.keras.applications.vgg19.preprocess_input)
            val_datagen.fit(x_val)
In [19]: ▶ from keras.callbacks import ReduceLROnPlateau
            learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                                     patience=3,
                                                     verbose=1,
                                                     factor=0.5,
                                                     min_lr=0.00001)
```

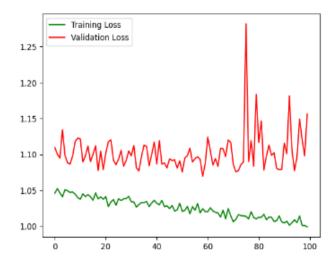
Out[76]: <matplotlib.legend.Legend at 0x7f75101e8160>



```
In [77]: M 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>

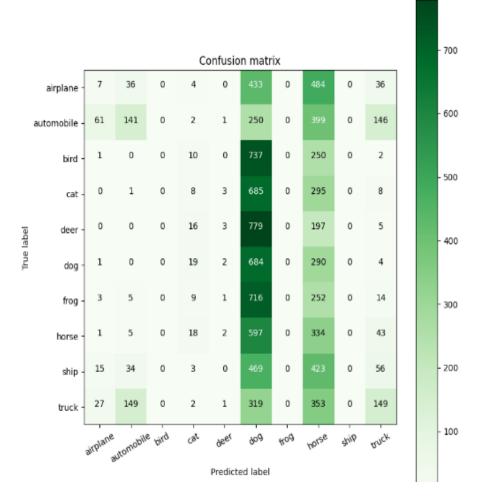


```
In [81]: M import itertools
              def plot_confusion_matrix(cm, classes,
                                            normalize=False,
                                            title='Confusion matrix',
                                           cmap=plt.cm.Greens):
                   This 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)
                       cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
print("Normalized confusion matrix")
                       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')
```

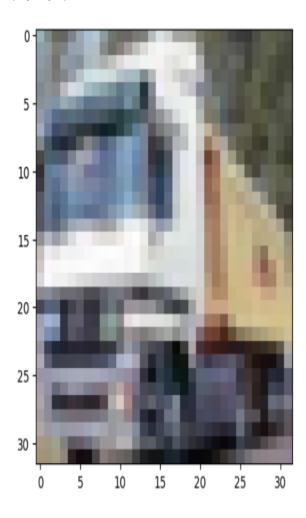
plot_confusion_matrix(cm,classes)

Confusion matrix, without normalization

In [82]: M plt.figure(figsize=(8,8))



(32, 32, 3)



```
In [ ]: M 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

