Neural Networks Deep Learning – Icp6

Name: Deekshitha, Gaddameedhi **ID: 700755765**

GitHub link: https://github.com/deekshitha430/icp7_neural

Video Link: https://drive.google.com/file/d/1I7IP-0CO_VdnmXTnYWPMlnPcxutW2769/view?usp=sharing

Use Case Description:

LeNet5, AlexNet, Vgg16, Vgg19

1. Training the model

2. Evaluating the model

LeNEt5&AlexNet:

Run Code

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.optimizers import RMSprop, Adam
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import classification_report, confusion_matrix
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
(x_train, y_train), (x_test, y_test)  = keras.datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 6s 0us/step
```

In [4]:
```
classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

In [5]:
```
y_train = y_train.reshape(-1,)
```

In [6]:
```
# Reshape converting 2D to 1D
y_test = y_test.reshape(-1,)
y_train = y_train.reshape(-1,)
```

In [7]:
```
# This code normalazation
x_train = x_train / 255.0
```

Neural Networks Deep Learning – Icp6

```
x_train = x_train / 255.0
x_test = x_test / 255.0
```

```
[8]: x_train.shape
```

```
Out[8]: (50000, 32, 32, 3)
```

```
[9]: from tensorflow.keras import layers, models
lenet = keras.models.Sequential([
    keras.layers.Conv2D(6, kernel_size=5, strides=1,  activation='relu', input_shape=(32,32,3), padding='same'), #C1
    keras.layers.AveragePooling2D(), #S1
    keras.layers.Conv2D(16, kernel_size=5, strides=1, activation='relu', padding='valid'), #C2
    keras.layers.AveragePooling2D(), #S2
    keras.layers.Conv2D(120, kernel_size=5, strides=1, activation='relu', padding='valid'), #C3
    keras.layers.Flatten(), #Flatten
    keras.layers.Dense(84, activation='relu'), #F1
    keras.layers.Dense(10, activation='softmax') #Output layer
])
```

```
[10]: lenet.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 6)         456

 average_pooling2d (AverageP  (None, 16, 16, 6)        0
 ooling2D)

 conv2d_1 (Conv2D)           (None, 12, 12, 16)        2416

 average_pooling2d_1 (Averag  (None, 6, 6, 16)         0
 ePooling2D)

 conv2d_2 (Conv2D)           (None, 2, 2, 120)         48120

 flatten (Flatten)           (None, 480)               0
```


```
In [10]: lenet.summary()
```


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 6)         456

 average_pooling2d (AverageP  (None, 16, 16, 6)        0
 ooling2D)

 conv2d_1 (Conv2D)           (None, 12, 12, 16)        2416

 average_pooling2d_1 (Averag  (None, 6, 6, 16)         0
 ePooling2D)

 conv2d_2 (Conv2D)           (None, 2, 2, 120)         48120

 flatten (Flatten)           (None, 480)               0

 dense (Dense)               (None, 84)                40404

 dense_1 (Dense)             (None, 10)                850

=================================================================
Total params: 92,246
Trainable params: 92,246
Non-trainable params: 0
_________________________________________________________________
```

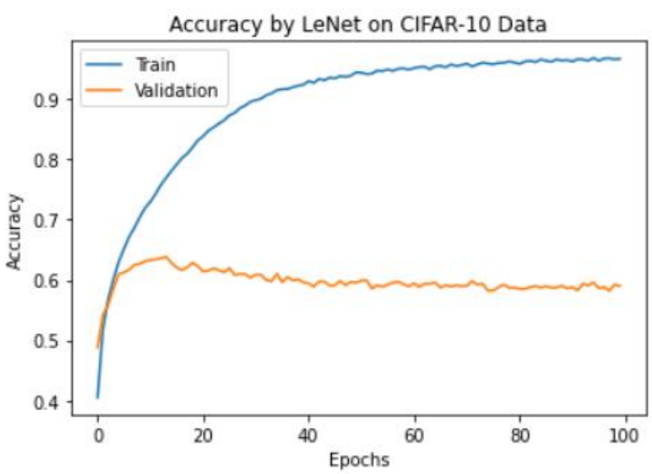
```
In [11]: lenet.compile(optimizer='adam', loss=keras.losses.sparse_categorical_crossentropy, metrics=['accuracy'])
```

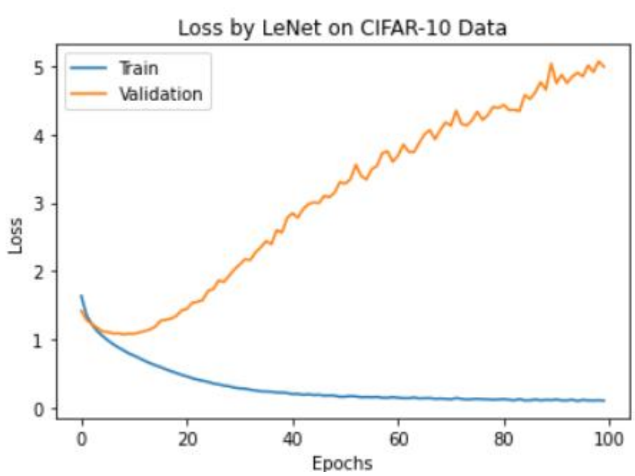
Neural Networks Deep Learning – Icp6

```
In [12]:  hist = lenet.fit(x_train, y_train, epochs=100, validation_data=(x_test, y_test),verbose=1)
acy: 0.5907
Epoch 95/100
1563/1563 [==============================] - 9s 6ms/step - loss: 0.0966 - accuracy: 0.9683 - val_loss: 4.9078 - val_accu
acy: 0.5959
Epoch 96/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.1112 - accuracy: 0.9635 - val_loss: 4.8539 - val_accu
acy: 0.5863
Epoch 97/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.1035 - accuracy: 0.9671 - val_loss: 5.0162 - val_accu
acy: 0.5881
Epoch 98/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.1029 - accuracy: 0.9678 - val_loss: 4.9166 - val_accu
acy: 0.5819
Epoch 99/100
1563/1563 [==============================] - 10s 6ms/step - loss: 0.1059 - accuracy: 0.9658 - val_loss: 5.0741 - val_acc
racy: 0.5924
Epoch 100/100
1563/1563 [==============================] - 8s 5ms/step - loss: 0.1006 - accuracy: 0.9670 - val_loss: 4.9955 - val_accu
acy: 0.5902
```

```
In [13]:  # summarize history for accuracy
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title("Accuracy by LeNet on CIFAR-10 Data")
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Loss by LeNet on CIFAR-10 Data')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['Train', 'Validation'])
plt.show()
```


```
plt.show()
```

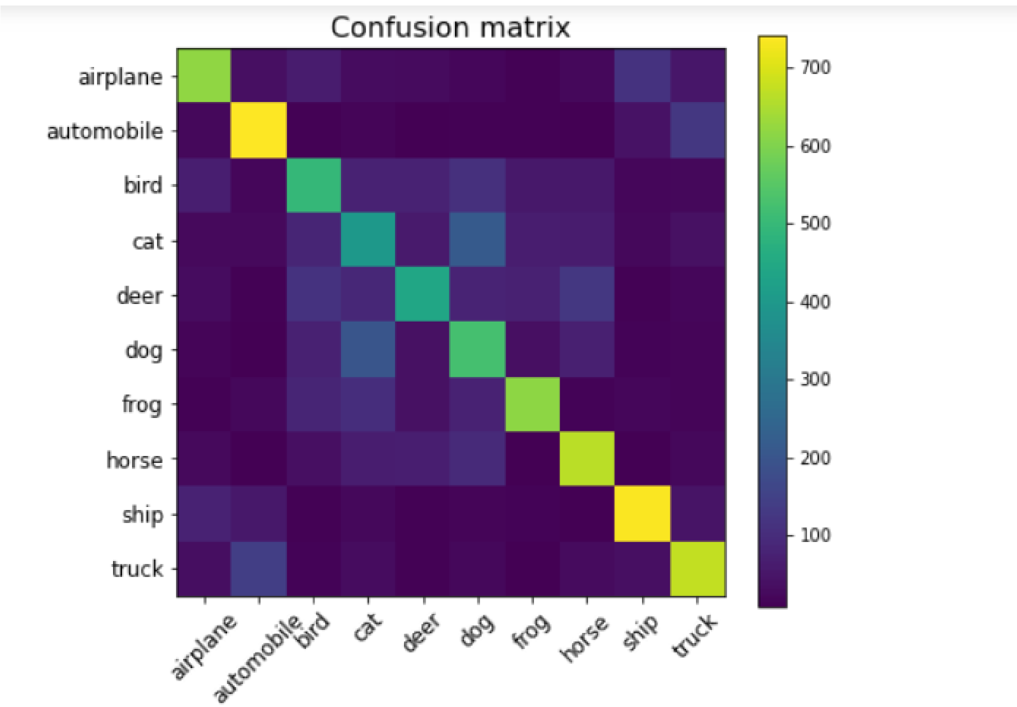





Neural Networks Deep Learning – Icp6

```
In [14]:  from sklearn.metrics import confusion_matrix
          from sklearn.metrics import ConfusionMatrixDisplay
          y_predictions= lenet.predict(x_test)
          y_predictions.reshape(-1,)
          y_predictions= np.argmax(y_predictions, axis=1)

          confusion_matrix(y_test, y_predictions)

          313/313 [==============================] - 1s 2ms/step

Out[14]:  array([[618,  36,  64,  32,  28,  21,  11,  24, 115,  51],
                 [ 24, 740,  12,  17,   7,  10,  12,   8,  43, 127],
                 [ 69,  21, 495,  76,  77, 110,  53,  55,  21,  23],
                 [ 25,  27,  83, 397,  61, 215,  67,  63,  23,  39],
                 [ 32,  10, 114,  90, 441,  80,  75, 125,  12,  21],
                 [ 18,   8,  75, 200,  39, 523,  38,  70,  13,  16],
                 [ 12,  23,  82, 102,  40,  76, 615,  13,  21,  16],
                 [ 25,   7,  37,  65,  68,  95,   8, 664,   8,  23],
                 [ 77,  55,  11,  22,  12,  16,  14,   9, 737,  47],
                 [ 33, 142,  13,  30,  11,  23,   9,  29,  38, 672]])
```

```
In [16]:  # confusion matrix and accuracy
          from sklearn.metrics import confusion_matrix, accuracy_score
          plt.figure(figsize=(7, 6))
          plt.title('Confusion matrix', fontsize=16)
          plt.imshow(confusion_matrix(y_test, y_predictions))
          plt.xticks(np.arange(10), classes, rotation=45, fontsize=12)
          plt.yticks(np.arange(10), classes, fontsize=12)
          plt.colorbar()
          plt.show()
```




```
print("Test accuracy:", accuracy_score(y_test, y_predictions))

Test accuracy: 0.5902
```

Neural Networks Deep Learning – Icp6

```python
L = 8
W = 8
fig, axes = plt.subplots(L, W, figsize = (20,20))
axes = axes.ravel() #

for i in np.arange(0, L * W):
    axes[i].imshow(x_test[i])
    axes[i].set_title("Predicted = {}\n Actual  = {}".format(classes[y_predictions[i]], classes[y_test[i]]))
    axes[i].axis('off')

plt.subplots_adjust(wspace=1)
```


```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import SGD, Adam
from keras.layers.convolutional import Convolution2D
from keras.layers.convolutional import MaxPooling2D
```

```python
#Define Alexnet Model
AlexNet = Sequential()
AlexNet.add(Conv2D(filters=16,kernel_size=(3,3),strides=(4,4),input_shape=(32,32,3), activation='relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
AlexNet.add(Conv2D(60,(5,5),padding='same',activation='relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
AlexNet.add(Conv2D(60,(3,3),padding='same',activation='relu'))
AlexNet.add(Conv2D(30,(3,3),padding='same',activation='relu'))
AlexNet.add(Conv2D(20,(3,3),padding='same',activation='relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
AlexNet.add(Flatten())
AlexNet.add(Dense(200, activation='relu'))
AlexNet.add(Dropout(0.1))
AlexNet.add(Dense(200, activation='relu'))
AlexNet.add(Dropout(0.1))
AlexNet.add(Dense(10,activation='softmax'))

AlexNet.compile(optimizer='SGD', loss=keras.losses.sparse_categorical_crossentropy, metrics=['accuracy'])
AlexNet.summary()
```

Neural Networks Deep Learning – Icp6

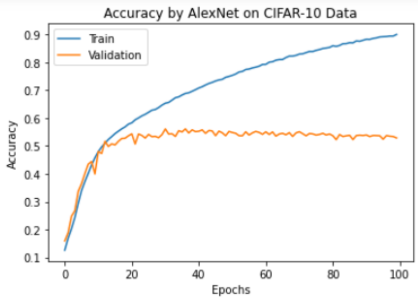
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_3 (Conv2D)           (None, 8, 8, 16)          448       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 4, 4, 16)         0         
 )                                                               
                                                                 
 conv2d_4 (Conv2D)           (None, 4, 4, 60)          24060     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 2, 2, 60)         0         
 2D)                                                             
                                                                 
 conv2d_5 (Conv2D)           (None, 2, 2, 60)          32460     
                                                                 
 conv2d_6 (Conv2D)           (None, 2, 2, 30)          16230     
                                                                 
 conv2d_7 (Conv2D)           (None, 2, 2, 20)          5420      
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 1, 1, 20)         0         
 2D)                                                             
                                                                 
 flatten_1 (Flatten)         (None, 20)                0         
                                                                 
 dense_2 (Dense)             (None, 200)               4200      
                                                                 
 dropout (Dropout)           (None, 200)               0         
                                                                 
 dense_3 (Dense)             (None, 200)               40200     
                                                                 
 dropout_1 (Dropout)         (None, 200)               0         
                                                                 
 dense_4 (Dense)             (None, 10)                2010      
                                                                 
=================================================================
Total params: 125,028
Trainable params: 125,028
Non-trainable params: 0
_________________________________________________________________
```

[22]:
```
history1 = AlexNet.fit(x_train, y_train, epochs=100, validation_data=(x_test, y_test),verbose=1)
```

```
racy: 0.5372
Epoch 95/100
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3035 - accuracy: 0.8912 - val_loss: 2.3625 - val_accu
racy: 0.5365
Epoch 96/100
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3017 - accuracy: 0.8921 - val_loss: 2.4652 - val_accu
racy: 0.5241
Epoch 97/100
1563/1563 [==============================] - 11s 7ms/step - loss: 0.2984 - accuracy: 0.8927 - val_loss: 2.4627 - val_accu
racy: 0.5373
Epoch 98/100
1563/1563 [==============================] - 11s 7ms/step - loss: 0.2925 - accuracy: 0.8944 - val_loss: 2.4082 - val_accu
racy: 0.5349
Epoch 99/100
1563/1563 [==============================] - 10s 7ms/step - loss: 0.2918 - accuracy: 0.8943 - val_loss: 2.4604 - val_accu
racy: 0.5338
Epoch 100/100
1563/1563 [==============================] - 11s 7ms/step - loss: 0.2835 - accuracy: 0.9001 - val_loss: 2.6043 - val_accu
racy: 0.5287
```

[23]:
```
# summarize history for accuracy
plt.plot(history1.history['accuracy'])
plt.plot(history1.history['val_accuracy'])
plt.title("Accuracy by AlexNet on CIFAR-10 Data")
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('Loss by AlexNet on CIFAR-10 Data')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['Train', 'Validation'])
plt.show()
```

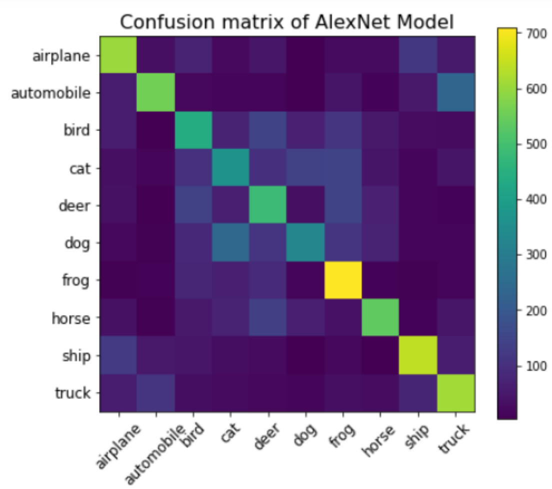
Neural Networks Deep Learning – Icp6



24]:

```
y_predictions1 = AlexNet.predict(x_test)
y_predictions1.reshape(-1,)
y_predictions1= np.argmax(y_predictions1, axis=1)

confusion_matrix(y_test, y_predictions1)
```

```
313/313 [==============================] - 1s 2ms/step
```



[26]:

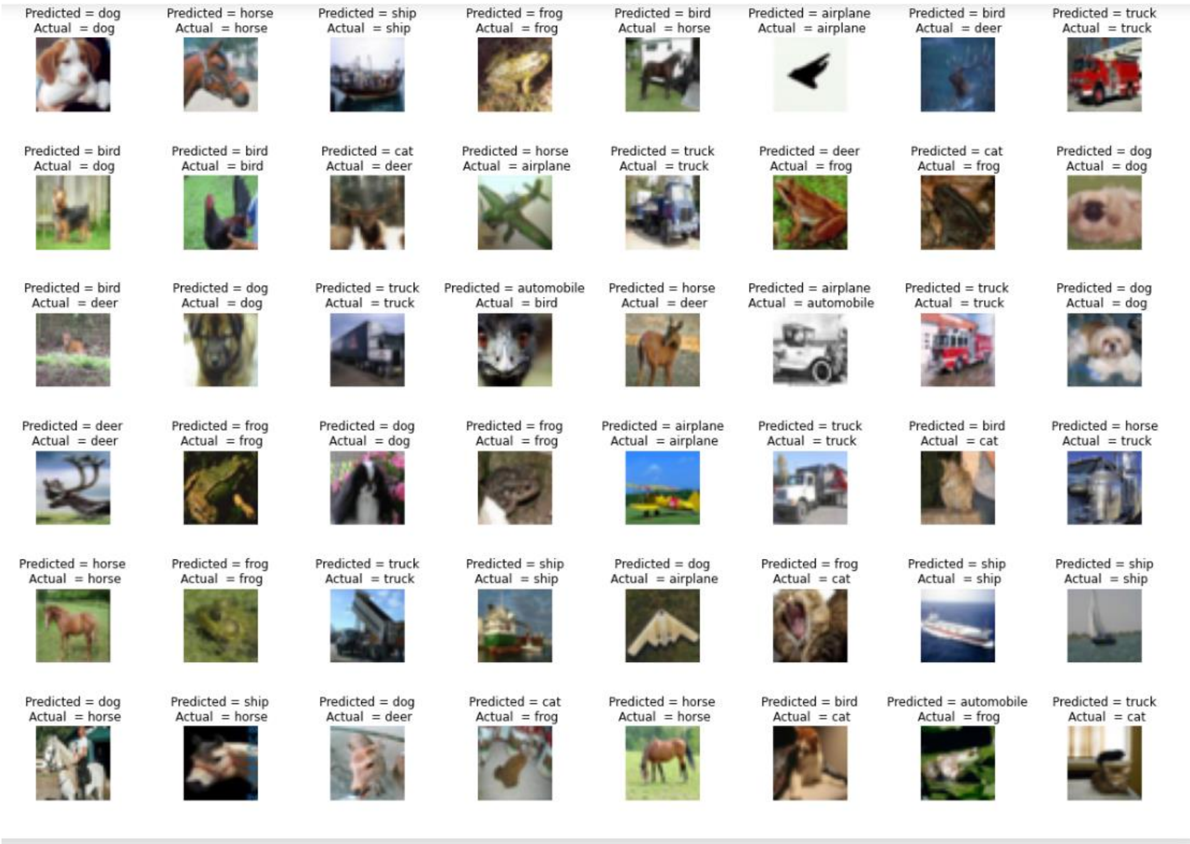
```
print("Test accuracy by AlexNet:", accuracy_score(y_test, y_predictions))
```

```
Test accuracy by AlexNet: 0.5902
```

[27]:

```
L = 8
W = 8
fig, axes = plt.subplots(L, W, figsize = (20,20))
axes = axes.ravel() #

for i in np.arange(0, L * W):
    axes[i].imshow(x_test[i])
    axes[i].set_title("Predicted = {}\n Actual  = {}".format(classes[y_predictions[i]], classes[y_test[i]]))
    axes[i].axis('off')

plt.subplots_adjust(wspace=1)
```

Neural Networks Deep Learning – Icp6



VGG16

```
import keras
from keras.models import Sequential
from keras.preprocessing import image
from keras.layers import Activation,Dense,Dropout,Conv2D,Flatten,MaxPooling2D,BatchNormalization
from keras.datasets import cifar10
from keras import optimizers
from matplotlib import pyplot as plt
```

```
# generate cifar10 data
(x_train,y_train),(x_test,y_test) = cifar10.load_data()
```

```
# config parameters
num_classes = 10
input_shape = x_train.shape[1:4]
optimizer = optimizers.Adam(lr=0.0003)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.Adam.
```

```
# convert label to one-hot
one_hot_y_train = keras.utils.to_categorical(y_train,num_classes=num_classes)
one_hot_y_test = keras.utils.to_categorical(y_test,num_classes=num_classes)
```

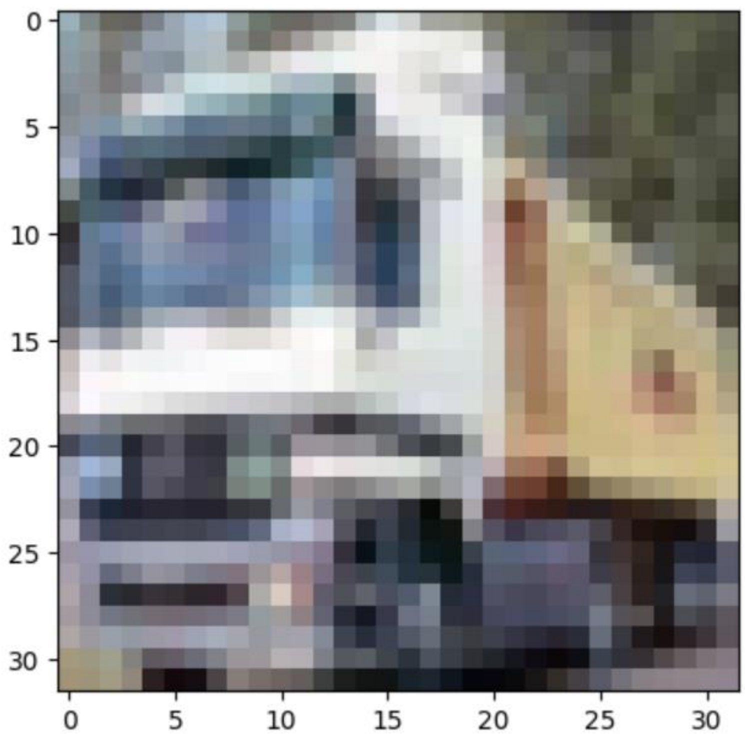
```
# check data
plt.imshow(x_train[1])
print(x_train[1].shape)
```

Neural Networks Deep Learning – Icp6

```
one_hot_y_test = keras.utils.to_categorical(y_test,num_classes=num_classes)
```

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

(32, 32, 3)




```
# build model(similar to VGG16, only change the input and output shape)
model = Sequential()
model.add(Conv2D(64,(3,3),activation='relu',input_shape=input_shape,padding='same'))
model.add(Conv2D(64,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Flatten())
model.add(Dense(4096,activation='relu'))
model.add(Dense(4096,activation='relu'))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

```
# config optimizer,loss,metrics
model.compile(optimizer=optimizer,loss='categorical_crossentropy',metrics=['accuracy'])
```

Neural Networks Deep Learning – Icp6

27]:
```
# train
history=model.fit(x=x_train,y=one_hot_y_train,batch_size=128,epochs=10,validation_split=0.1)
```

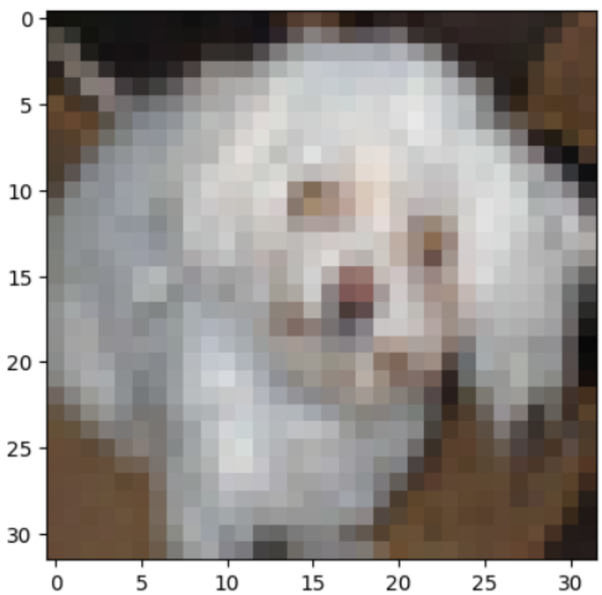
```
Epoch 1/10
352/352 [==============================] - 25s 70ms/step - loss: 2.3027 - accuracy: 0.0997 - val_loss: 2.3027 - val_accuracy:
0.0950
Epoch 2/10
352/352 [==============================] - 22s 61ms/step - loss: 2.3027 - accuracy: 0.0993 - val_loss: 2.3027 - val_accuracy:
0.0986
Epoch 3/10
352/352 [==============================] - 21s 61ms/step - loss: 2.3027 - accuracy: 0.0998 - val_loss: 2.3028 - val_accuracy:
0.0950
Epoch 4/10
352/352 [==============================] - 22s 62ms/step - loss: 2.3027 - accuracy: 0.0993 - val_loss: 2.3029 - val_accuracy:
0.0950
Epoch 5/10
352/352 [==============================] - 22s 63ms/step - loss: 2.3027 - accuracy: 0.0965 - val_loss: 2.3029 - val_accuracy:
0.0958
Epoch 6/10
352/352 [==============================] - 22s 62ms/step - loss: 2.3027 - accuracy: 0.0955 - val_loss: 2.3028 - val_accuracy:
0.0976
Epoch 7/10
352/352 [==============================] - 22s 61ms/step - loss: 2.3027 - accuracy: 0.0990 - val_loss: 2.3029 - val_accuracy:
0.0958
Epoch 8/10
352/352 [==============================] - 21s 61ms/step - loss: 2.3027 - accuracy: 0.0983 - val_loss: 2.3028 - val_accuracy:
0.1024
Epoch 9/10
352/352 [==============================] - 21s 61ms/step - loss: 2.3027 - accuracy: 0.0990 - val_loss: 2.3027 - val_accuracy:
0.0976
Epoch 10/10
352/352 [==============================] - 21s 61ms/step - loss: 2.3027 - accuracy: 0.0990 - val_loss: 2.3028 - val_accuracy:
0.0976
```

28]:

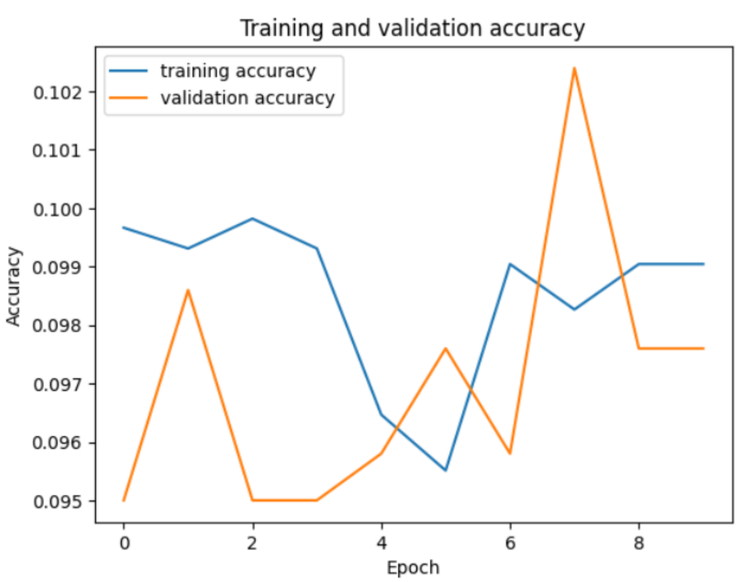
Neural Networks Deep Learning – Icp6

```
# predict
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 [==============================] - 0s 143ms/step
predict class: 6
expected class: 5
```




```
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



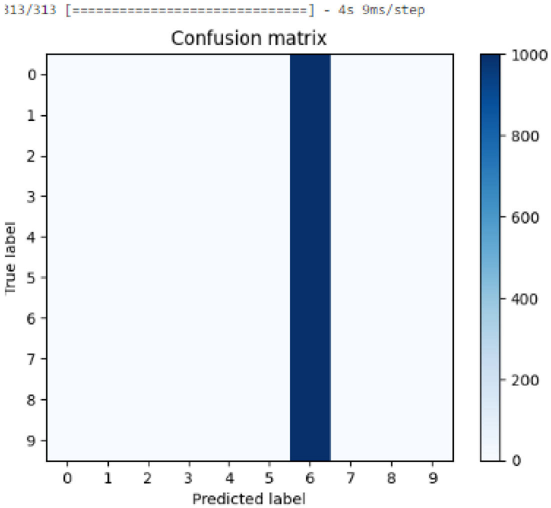
Neural Networks Deep Learning – Icp6

[32]:

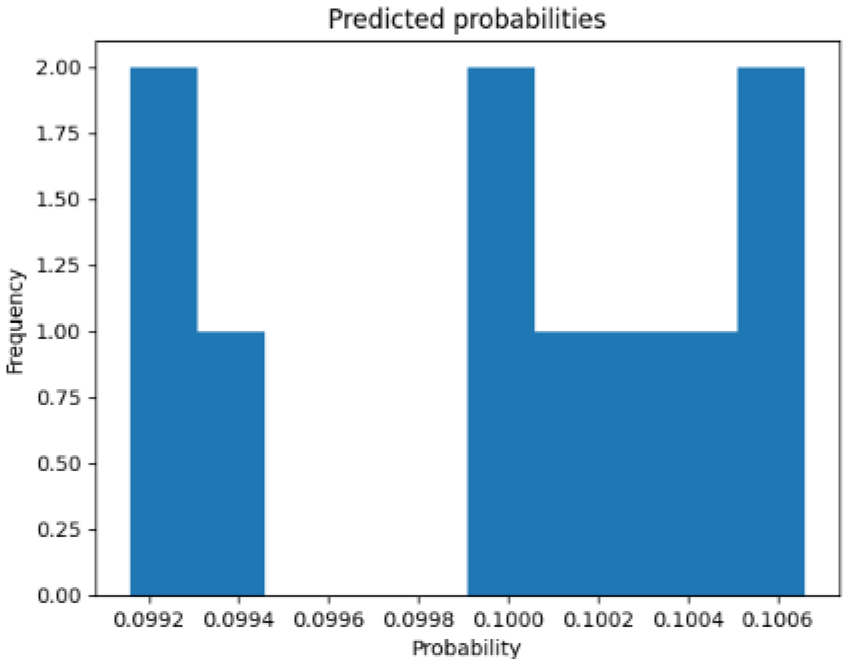
```
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```


```
import numpy as np
from sklearn.metrics import confusion_matrix

# calculate the confusion matrix
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = y_test.ravel()
cm = confusion_matrix(y_true, y_pred_classes)

# plot the confusion matrix
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion matrix')
plt.colorbar()
tick_marks = np.arange(num_classes)
plt.xticks(tick_marks, range(num_classes))
plt.yticks(tick_marks, range(num_classes))
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

# plot a histogram of the predicted probabilities for a sample image
plt.hist(y_pred[1000])
plt.title('Predicted probabilities')
plt.xlabel('Probability')
plt.ylabel('Frequency')
plt.show()
```

```
313/313 [==============================] - 4s 9ms/step
```



Neural Networks Deep Learning – Icp6



VGG19MODE_CIFAR100:

Import libraries

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf

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 [ ]:

```
cifar100 = tf.keras.datasets.cifar100
(X_train,Y_train) , (X_test,Y_test) = cifar10.load_data()
```

In [ ]:

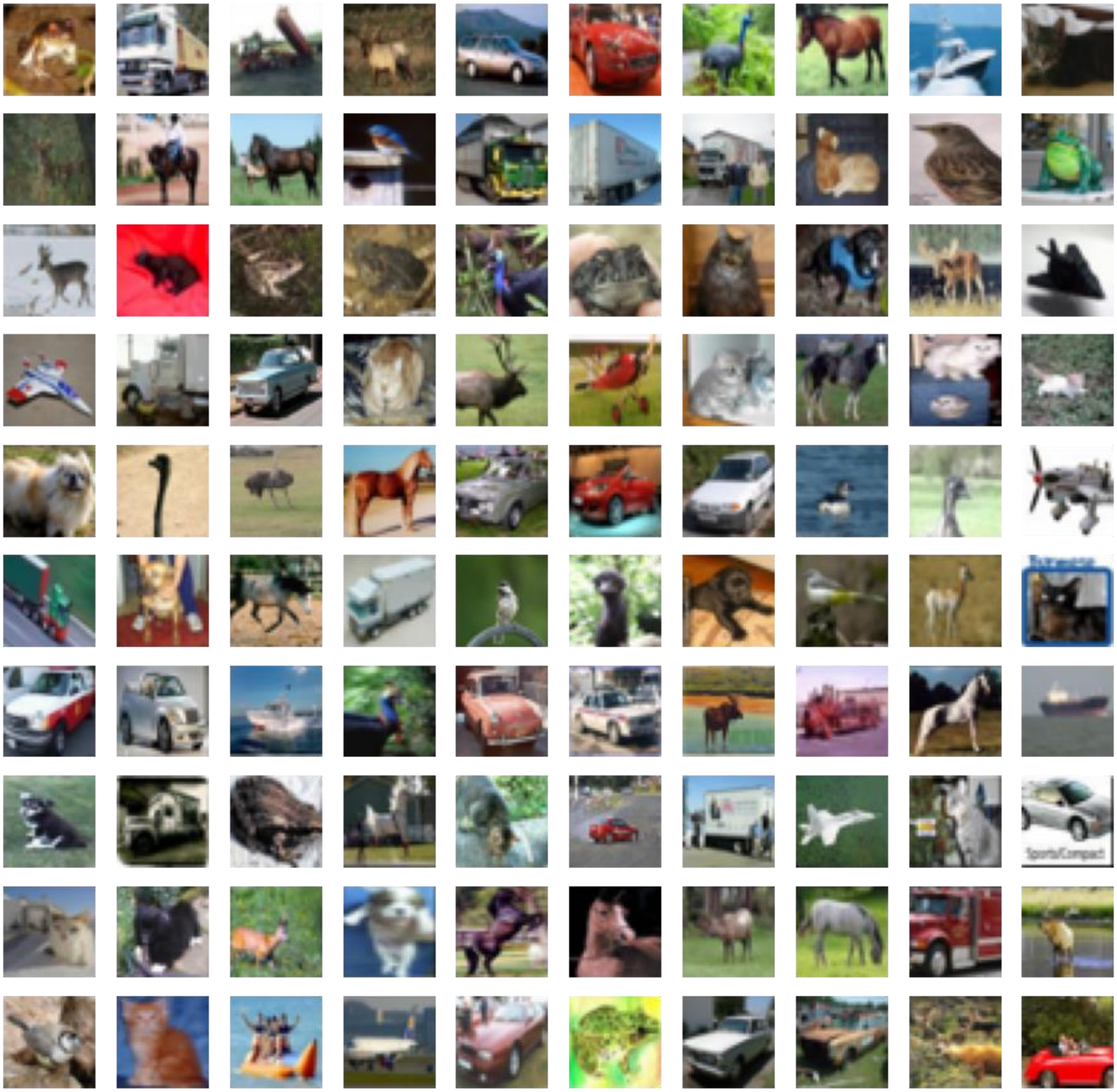
```
classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

Let's look into the dataset images

In [ ]:

```
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')
```

Neural Networks Deep Learning – Icp6



Training , Validating and Splitting trained and tested data

In [ ]:

```
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)
```

In [ ]:

```
from keras.utils.np_utils import to_categorical
y_train = to_categorical(y_train, num_classes = 10)
y_val = to_categorical(y_val, num_classes = 10)
```

In [ ]:

```
print(x_train.shape)
print(y_train.shape)
```

Neural Networks Deep Learning – Icp6

```
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 [ ]:

```
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 [ ]:

```
from keras.callbacks import ReduceLROnPlateau
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

We have used only 16 layers out of 19 layers in the CNN

In [ ]:

```
vgg_model = tf.keras.applications.VGG19(
    include_top=False,
    weights="imagenet",
    input_shape=(32,32,3),
)

vgg_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kern
els_notop.h5
80142336/80134624 [==============================] - 1s 0us/step
Model: "vgg19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928
_________________________________________________________________
```

Neural Networks Deep Learning – Icp6

```
block1_pool (MaxPooling2D)   (None, 16, 16, 64)       0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)      73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)      147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)        0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)        295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)        590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)        590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 8, 8, 256)        590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)        0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 4, 4, 512)        1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 4, 512)        2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 4, 512)        2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 4, 4, 512)        2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 2, 512)        0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 2, 512)        2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 2, 512)        2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 2, 512)        2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 2, 2, 512)        2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)        0
=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model = tf.keras.Sequential()
model.add(vgg_model)
model.add(Flatten())
model.add(Dense(1024, activation = 'relu'))
model.add(Dense(1024, activation = 'relu'))
model.add(Dense(256, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

Neural Networks Deep Learning – Icp6

```
=================================================================
vgg19 (Functional)         (None, 1, 1, 512)        20024384
_________________________________________________________________
flatten (Flatten)          (None, 512)              0
_________________________________________________________________
dense (Dense)              (None, 1024)             525312
_________________________________________________________________
dense_1 (Dense)            (None, 1024)             1049600
_________________________________________________________________
dense_2 (Dense)            (None, 256)              262400
_________________________________________________________________
dense_3 (Dense)            (None, 10)               2570
=================================================================
Total params: 21,864,266
Trainable params: 21,864,266
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
optimizer = tf.keras.optimizers.SGD(lr = 0.001, momentum = 0.9)
model.compile(optimizer= optimizer,
          loss='categorical_crossentropy',
          metrics=['accuracy'])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/optimizer_v2.py:375: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

In [ ]:

```
history = model.fit(
    train_datagen.flow(x_train, y_train, batch_size = 128),
    validation_data = val_datagen.flow(x_val,y_val, batch_size = 128),
    epochs = 25,
    verbose = 1,
    callbacks = [learning_rate_reduction]
)
```

```
Epoch 1/25
313/313 [==============================] - 84s 165ms/step - loss: 1.6671 - accuracy: 0.3875 - val_loss: 1.1241 - val_accur
acy: 0.6056
Epoch 2/25
313/313 [==============================] - 50s 159ms/step - loss: 0.9631 - accuracy: 0.6640 - val_loss: 0.7466 - val_accur
acy: 0.7404
Epoch 3/25
313/313 [==============================] - 50s 160ms/step - loss: 0.7464 - accuracy: 0.7430 - val_loss: 0.6792 - val_accur
acy: 0.7716
Epoch 4/25
313/313 [==============================] - 50s 160ms/step - loss: 0.6533 - accuracy: 0.7782 - val_loss: 0.6814 - val_accur
acy: 0.7829
Epoch 5/25
313/313 [==============================] - 50s 159ms/step - loss: 0.5779 - accuracy: 0.8013 - val_loss: 0.5932 - val_accur
acy: 0.8058
Epoch 6/25
313/313 [==============================] - 50s 160ms/step - loss: 0.5369 - accuracy: 0.8136 - val_loss: 0.5455 - val_accur
acy: 0.8157
Epoch 7/25
313/313 [==============================] - 50s 160ms/step - loss: 0.4925 - accuracy: 0.8299 - val_loss: 0.5119 - val_accur
acy: 0.8269
Epoch 8/25
```

```
313/313 [==============================] - 50s 160ms/step - loss: 0.4660 - accuracy: 0.8398 - val_loss: 0.5036 - val_accur
acy: 0.8342
Epoch 9/25
313/313 [==============================] - 50s 160ms/step - loss: 0.4315 - accuracy: 0.8523 - val_loss: 0.4470 - val_accur
acy: 0.8461
Epoch 10/25
313/313 [==============================] - 50s 160ms/step - loss: 0.4110 - accuracy: 0.8569 - val_loss: 0.4712 - val_accur
acy: 0.8440
Epoch 11/25
313/313 [==============================] - 50s 159ms/step - loss: 0.3797 - accuracy: 0.8673 - val_loss: 0.4721 - val_accur
acy: 0.8504
Epoch 12/25
313/313 [==============================] - 50s 160ms/step - loss: 0.3641 - accuracy: 0.8728 - val_loss: 0.4315 - val_accur
acy: 0.8541
Epoch 13/25
313/313 [==============================] - 50s 159ms/step - loss: 0.3399 - accuracy: 0.8826 - val_loss: 0.4618 - val_accur
acy: 0.8501
Epoch 14/25
313/313 [==============================] - 50s 159ms/step - loss: 0.3259 - accuracy: 0.8860 - val_loss: 0.4929 - val_accur
acy: 0.8448
Epoch 15/25
313/313 [==============================] - 50s 159ms/step - loss: 0.3130 - accuracy: 0.8925 - val_loss: 0.4325 - val_accur
acy: 0.8604
Epoch 16/25
313/313 [==============================] - 50s 159ms/step - loss: 0.2969 - accuracy: 0.8955 - val_loss: 0.4943 - val_accur
acy: 0.8477
Epoch 17/25
313/313 [==============================] - 50s 159ms/step - loss: 0.2805 - accuracy: 0.9008 - val_loss: 0.4315 - val_accur
acy: 0.8650
Epoch 18/25
313/313 [==============================] - 50s 159ms/step - loss: 0.2638 - accuracy: 0.9089 - val_loss: 0.4128 - val_accur
acy: 0.8674
Epoch 19/25
313/313 [==============================] - 50s 159ms/step - loss: 0.2528 - accuracy: 0.9108 - val_loss: 0.4517 - val_accur
acy: 0.8616
Epoch 20/25
313/313 [==============================] - 50s 159ms/step - loss: 0.2420 - accuracy: 0.9147 - val_loss: 0.4273 - val_accur
acy: 0.8658
Epoch 21/25
313/313 [==============================] - 50s 159ms/step - loss: 0.2311 - accuracy: 0.9158 - val_loss: 0.4323 - val_accur
acy: 0.8693
Epoch 22/25
313/313 [==============================] - 50s 159ms/step - loss: 0.2142 - accuracy: 0.9244 - val_loss: 0.4716 - val_accur
acy: 0.8679
Epoch 23/25
313/313 [==============================] - 50s 159ms/step - loss: 0.2091 - accuracy: 0.9272 - val_loss: 0.4171 - val_accur
acy: 0.8765
Epoch 24/25
313/313 [==============================] - 50s 159ms/step - loss: 0.1964 - accuracy: 0.9304 - val_loss: 0.4461 - val_accur
acy: 0.8724
Epoch 25/25
313/313 [==============================] - 50s 159ms/step - loss: 0.1879 - accuracy: 0.9337 - val_loss: 0.4827 - val_accur
acy: 0.8671
```

In [ ]:

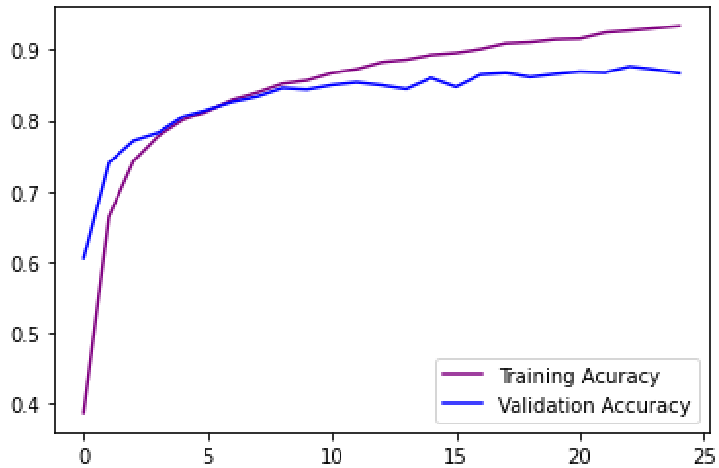
```
acc = history.history['accuracy']
```

Neural Networks Deep Learning – Icp6

```
val_acc = history.history['val_accuracy']

plt.figure()
plt.plot(acc,color = 'purple',label = 'Training Acuracy')
plt.plot(val_acc,color = 'blue',label = 'Validation Accuracy')
plt.legend()
```

Out[ ]:

<matplotlib.legend.Legend at 0x7fc5601fe610>

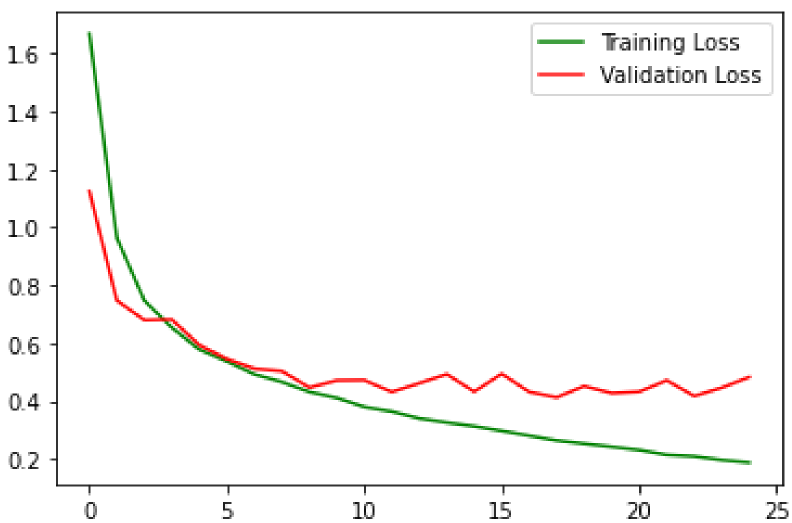


In [ ]:

```
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[ ]:

<matplotlib.legend.Legend at 0x7fc5bd878ad0>



In [ ]:

```
x_test = tf.keras.applications.vgg19.preprocess_input(X_test)
y_pred = model.predict_classes(x_test)
y_pred[:10]
```

Neural Networks Deep Learning – Icp6

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/sequential.py:455: UserWarning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

Out[ ]:

```
array([3, 8, 8, 0, 6, 6, 1, 6, 3, 1])
```

In [ ]:

```python
from sklearn.metrics import confusion_matrix, accuracy_score
print('Testing Accuarcy : ', accuracy_score(Y_test, y_pred))
```

```
Testing Accuarcy :  0.8591
```

In [ ]:

```python
cm = confusion_matrix(Y_test, y_pred)
cm
```

Out[ ]:

```
array([[882,  11,  20,   1,  12,   1,   5,   9,  43,  16],
       [  5, 944,   0,   2,   0,   1,   2,   0,   8,  38],
       [ 22,   3, 800,  11,  54,  30,  56,   9,  10,   5],
       [ 10,   7,  42, 536,  57, 228,  74,  26,   6,  14],
       [  6,   1,  22,   5, 884,  20,  35,  23,   2,   2],
       [  5,   4,   9,  54,  36, 845,  19,  26,   0,   2],
       [  5,   1,  11,   7,  16,  10, 941,   3,   3,   3],
       [  4,   1,  12,   7,  31,  28,   8, 902,   2,   5],
       [ 25,  16,   1,   1,   4,   0,   4,   1, 929,  19],
       [  6,  41,   1,   2,   4,   0,   6,   2,  10, 928]])
```

In [ ]:

```python
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")
```

Neural Networks Deep Learning – Icp6

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

In [ ]:

```
plt.figure(figsize=(8,8))
plot_confusion_matrix(cm,classes)
```

Confusion matrix, without normalization