Neural Networks & Deep Learning – ICP 8

Name: Vyshnavi Bussa

Student ID: 700757313

GitHub Link: https://github.com/Vyshnavi1157/NN-ICP8.git

Video Link: https://drive.google.com/file/d/1VZ-0oI43-Co-XsxZzGmezO4zoHhuIowC/view?usp=sharing

In [8]:
```
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
```

In [10]:
```
#generate cifar10 data
(x_train,y_train),(x_test,y_test) = cifar10.load_data()
```

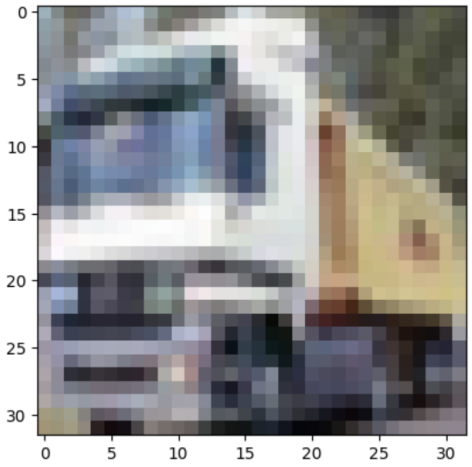
In [9]:
```
#config parameters
num_classes = 10
input_shape = x_train.shape[1:4]
optimizer = optimizers.Adam(lr=0.001)
```

In [11]:
```
#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)
```

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

```
(32, 32, 3)
```

```
(32, 32, 3)
```



In [13]:

```
# 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(BatchNormalization())
model.add(Conv2D(64,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))
```

```
model.add(Conv2D(128,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(256,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(512,(3,3),activation='relu',padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(4096,activation='relu'))
model.add(Dense(2048, activation='relu'))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

In [14]:
```
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

In [14]:
```
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

In [15]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 32, 32, 64)        1792

 batch_normalization (BatchN  (None, 32, 32, 64)       256
 ormalization)

 conv2d_2 (Conv2D)           (None, 32, 32, 64)        36928

 batch_normalization_1 (Batc  (None, 32, 32, 64)       256
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 64)       0
 )

 dropout (Dropout)           (None, 16, 16, 64)        0
```

In [ ]:
```
history = model.fit(x=x_train, y=one_hot_y_train, batch_size=128, epochs=30, validation_split=0.1)
```

```
Epoch 1/30
352/352 [==============================] - 29s 81ms/step - loss: 1.7083 - accuracy: 0.3346 - val_loss: 2.6400 - val_accuracy:
0.3686
Epoch 2/30
352/352 [==============================] - 27s 76ms/step - loss: 1.3159 - accuracy: 0.5254 - val_loss: 1.6935 - val_accuracy:
0.5590
Epoch 3/30
352/352 [==============================] - 27s 77ms/step - loss: 1.0110 - accuracy: 0.6538 - val_loss: 1.0628 - val_accuracy:
0.6556
Epoch 4/30
352/352 [==============================] - 27s 78ms/step - loss: 0.8493 - accuracy: 0.7144 - val_loss: 0.9995 - val_accuracy:
0.6856
Epoch 5/30
352/352 [==============================] - 28s 79ms/step - loss: 0.7343 - accuracy: 0.7566 - val_loss: 0.8316 - val_accuracy:
0.7326
Epoch 6/30
352/352 [==============================] - 28s 79ms/step - loss: 0.6515 - accuracy: 0.7907 - val_loss: 0.8145 - val_accuracy:
```

```
0.8430
Epoch 22/30
352/352 [==============================] - 28s 78ms/step - loss: 0.1242 - accuracy: 0.9620 - val_loss: 0.6557 - val_accuracy:
0.8544
Epoch 23/30
352/352 [==============================] - 28s 80ms/step - loss: 0.1144 - accuracy: 0.9639 - val_loss: 0.7322 - val_accuracy:
0.8400
Epoch 24/30
352/352 [==============================] - 28s 80ms/step - loss: 0.1086 - accuracy: 0.9653 - val_loss: 0.6893 - val_accuracy:
0.8548
Epoch 25/30
352/352 [==============================] - 28s 78ms/step - loss: 0.1313 - accuracy: 0.9619 - val_loss: 0.6570 - val_accuracy:
0.8412
Epoch 26/30
352/352 [==============================] - 28s 80ms/step - loss: 0.1069 - accuracy: 0.9673 - val_loss: 0.6935 - val_accuracy:
0.8454
Epoch 27/30
352/352 [==============================] - 28s 78ms/step - loss: 0.0962 - accuracy: 0.9703 - val_loss: 0.7319 - val_accuracy:
0.8376
Epoch 28/30
352/352 [==============================] - 28s 78ms/step - loss: 0.0871 - accuracy: 0.9728 - val_loss: 0.7403 - val_accuracy:
0.8480
Epoch 29/30
352/352 [==============================] - 28s 79ms/step - loss: 0.0811 - accuracy: 0.9749 - val_loss: 0.6592 - val_accuracy:
0.8510
Epoch 30/30
352/352 [==============================] - 28s 78ms/step - loss: 0.0753 - accuracy: 0.9770 - val_loss: 0.6631 - val_accuracy:
0.8642
```

In [ ]:
```python
# evaluate
print(model.metrics_names)
model.evaluate(x=x_test,y=one_hot_y_test,batch_size=512)
```

```
['loss', 'accuracy']
20/20 [==============================] - 5s 132ms/step - loss: 0.6627 - accuracy: 0.8592
```

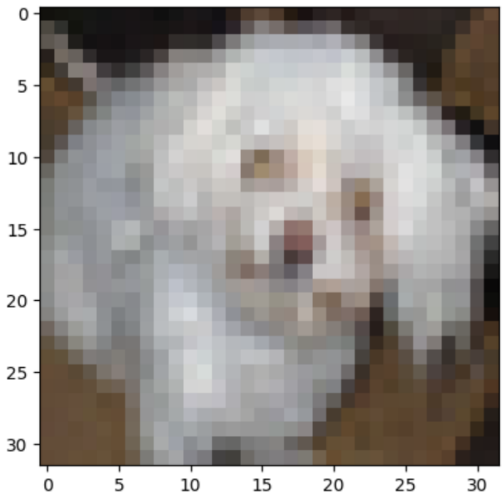
Out[21]:
```
[0.6626988053321838, 0.8592000007629395]
```

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

result = model.predict(x_test[1000:1001]).tolist()
predict = 0
```

Out[21]: [0.6626988053321838, 0.8592000007629395]

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)
```

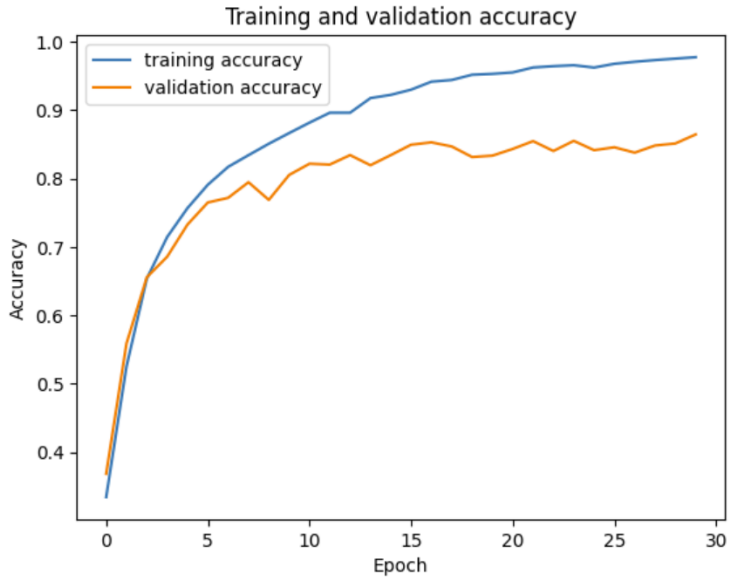
```
1/1 [==============================] - 1s 740ms/step
predict class: 5
expected class: 5
```



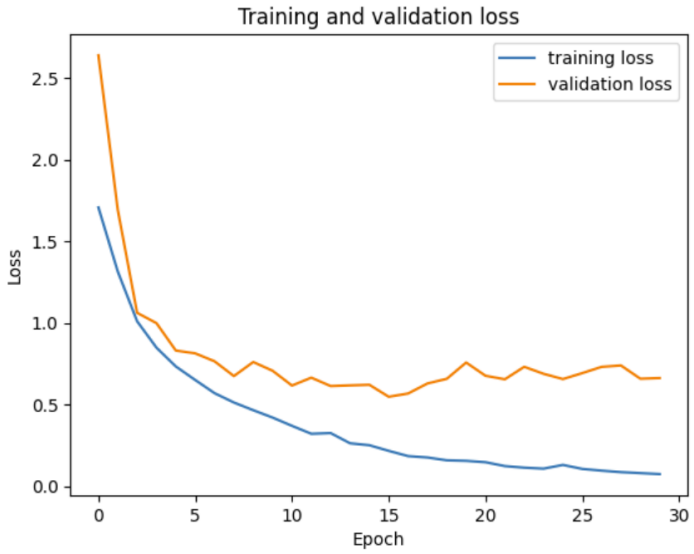
0 5 10 15 20 25 30

In [16]:
```
# save model
model.save("keras-VGG16-cifar10.h5")
```

In [ ]:
```
#plot the training and validation accuracy
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()
```



In [ ]:
```
#plot the training and validation loss
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()
```

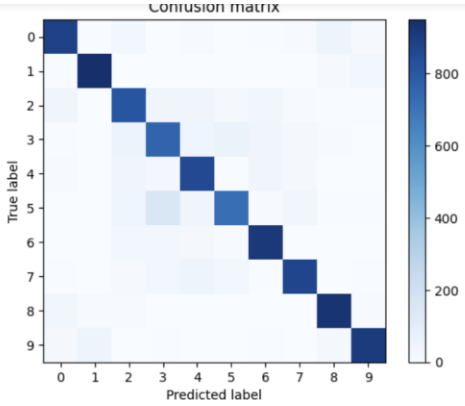


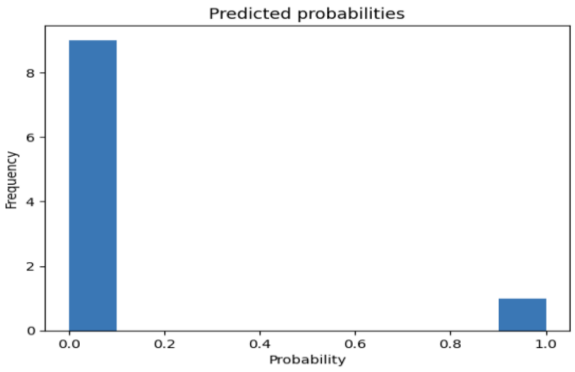
In [ ]:
```
import numpy as np
from sklearn.metrics import confusion_matrix

# calculate the confusion matrix
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
```

Epoch

In [ ]:
```
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 [==============================] - 3s 8ms/step
```





Import libraries

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
```

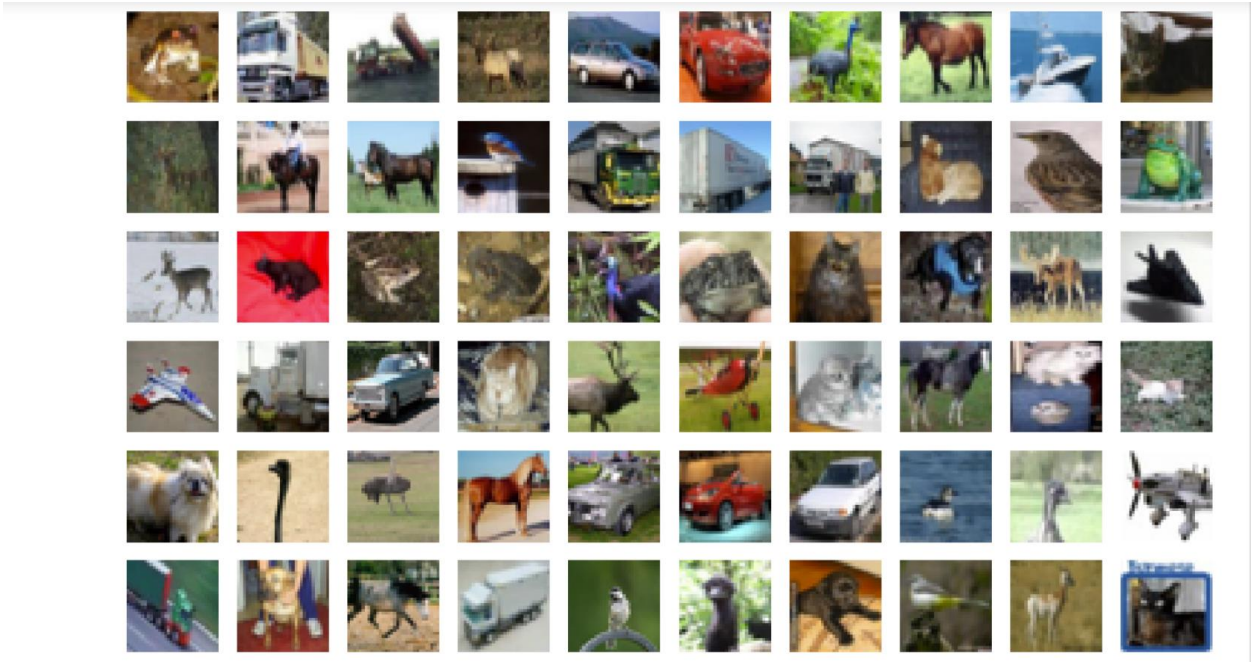
Extract data and train and test dataset

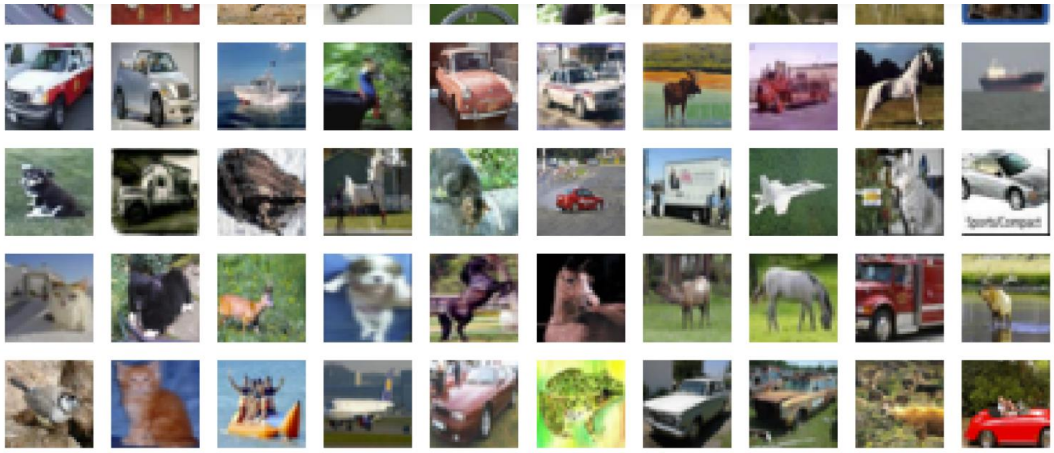
In [12]:
```
#cifar100 = tf.keras.datasets.cifar100
(X_train,Y_train) , (X_test,Y_test) = cifar10.load_data()
```

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')
```





Training , Validating and Splitting trained and tested data

In [15]:
```
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 [16]:
```
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 [17]:
```
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)
```

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

In [20]:
```
vgg_model = tf.keras.applications.VGG19(
    include_top=False,
    weights=None,
    input_shape=(32,32,3),
)

vgg_model.summary()
```

```
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

 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
```

```
 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 [24]:
```
model = tf.keras.Sequential()
model.add(vgg_model)
model.add(Flatten())
model.add(Dense(1024, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dense(1024, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dense(256, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(10, activation = 'softmax'))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 1, 1, 512)         20024384
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 1, 1, 512)         20024384

 flatten_1 (Flatten)         (None, 512)               0

 dense_4 (Dense)             (None, 1024)              525312

 batch_normalization_3 (Batc  (None, 1024)             4096
 hNormalization)

 dense_5 (Dense)             (None, 1024)              1049600

 batch_normalization_4 (Batc  (None, 1024)             4096
 hNormalization)

 dense_6 (Dense)             (None, 256)               262400

 batch_normalization_5 (Batc  (None, 256)              1024
 hNormalization)

 dropout_1 (Dropout)         (None, 256)               0

 dense_7 (Dense)             (None, 10)                2570

=================================================================
Total params: 21,873,482
Trainable params: 21,868,874
Non-trainable params: 4,608
_________________________________________________________________
```

In [25]:
```
optimizer = tf.keras.optimizers.SGD(learning_rate = 0.001, momentum = 0.9)
model.compile(optimizer= optimizer,
            loss='categorical_crossentropy',
            metrics=['accuracy'])
```


```
Trainable params: 21,868,874
Non-trainable params: 4,608
_________________________________________________________________
```

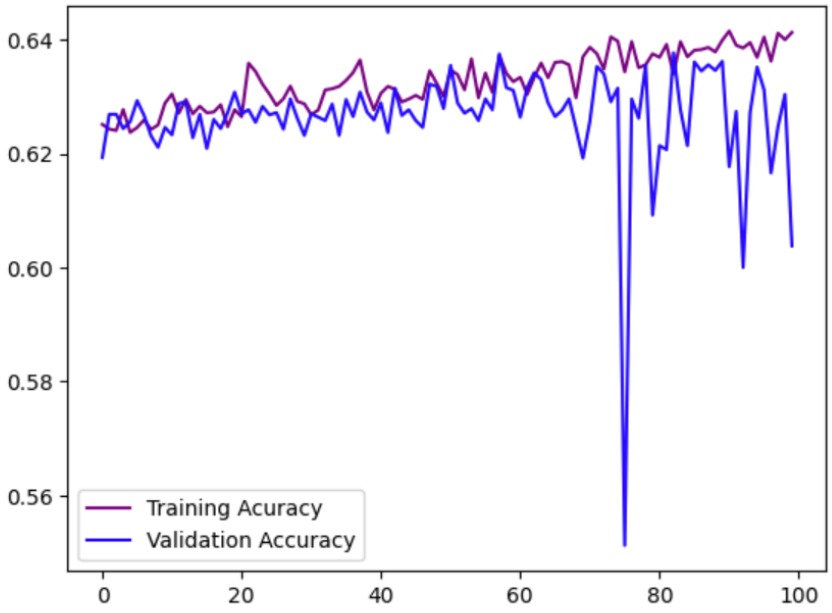
In [25]:
```
optimizer = tf.keras.optimizers.SGD(learning_rate = 0.001, momentum = 0.9)
model.compile(optimizer= optimizer,
            loss='categorical_crossentropy',
            metrics=['accuracy'])
```


In [75]:
```
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 = 100,
    verbose = 1,
    callbacks = [learning_rate_reduction]
)
```

```
Epoch 1/100
313/313 [==============================] - 33s 106ms/step - loss: 1.0459 - accuracy: 0.6251 - val_loss: 1.1096 - val_accurac
y: 0.6193 - lr: 1.0000e-05
Epoch 2/100
313/313 [==============================] - 32s 102ms/step - loss: 1.0524 - accuracy: 0.6243 - val_loss: 1.1004 - val_accurac
y: 0.6269 - lr: 1.0000e-05
Epoch 3/100
313/313 [==============================] - 32s 101ms/step - loss: 1.0464 - accuracy: 0.6241 - val_loss: 1.0950 - val_accurac
y: 0.6269 - lr: 1.0000e-05
Epoch 4/100
313/313 [==============================] - 33s 105ms/step - loss: 1.0410 - accuracy: 0.6277 - val_loss: 1.1344 - val_accurac
y: 0.6244 - lr: 1.0000e-05
Epoch 5/100
313/313 [==============================] - 32s 102ms/step - loss: 1.0508 - accuracy: 0.6237 - val_loss: 1.0986 - val_accurac
y: 0.6256 - lr: 1.0000e-05
Epoch 6/100
313/313 [==============================] - 33s 104ms/step - loss: 1.0493 - accuracy: 0.6246 - val_loss: 1.0882 - val_accurac
y: 0.6293 - lr: 1.0000e-05
```

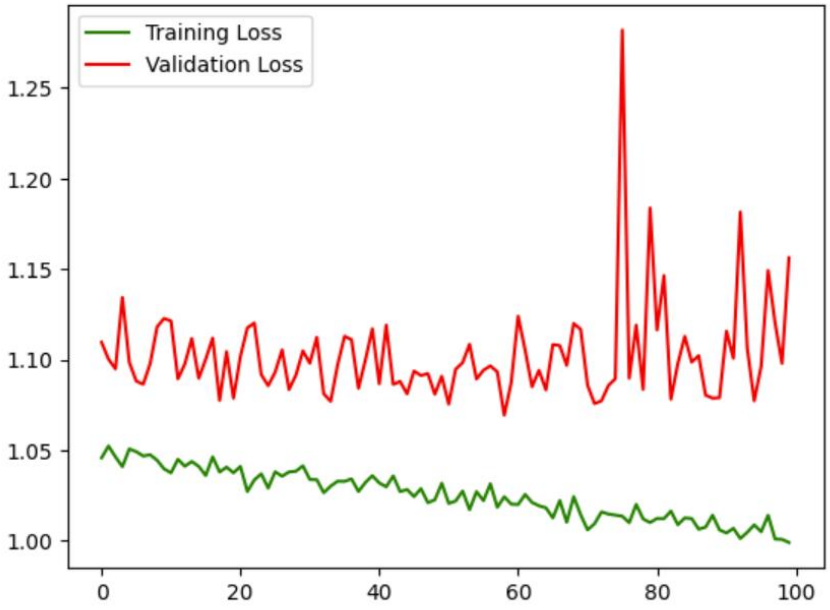
In [76]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

Out[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 [78]:
```
X_test = tf.keras.applications.vgg19.preprocess_input(X_test)
y_pred = np.argmax(model.predict(X_test), axis=-1)

y_pred[:10]
```

```
313/313 [==============================] - 3s 9ms/step
```

Out[78]:
```
array([5, 1, 5, 5, 5, 5, 7, 5, 5, 7])
```

In [79]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
print('Testing Accuarcy : ', accuracy_score(Y_test, y_pred))
```

```
Testing Accuarcy :  0.1326
```

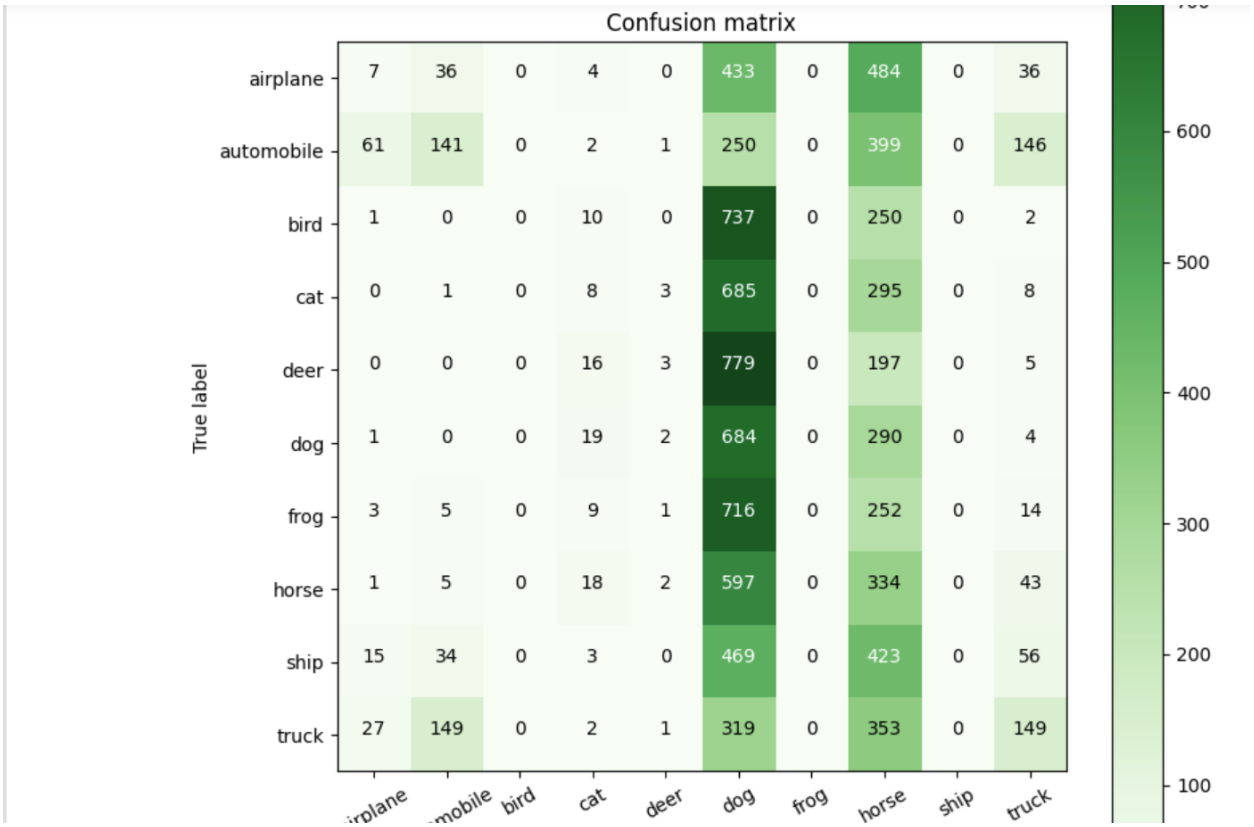
In [80]:
```
cm = confusion_matrix(Y_test, y_pred)
cm
```

Out[80]:
```
array([[  7,  36,   0,   4,   0, 433,   0, 484,   0,  36],
       [ 61, 141,   0,   2,   1, 250,   0, 399,   0, 146],
       [  1,   0,   0,  10,   0, 737,   0, 250,   0,   2],
       [  0,   1,   0,   8,   3, 685,   0, 295,   0,   8],
       [  0,   0,   0,  16,   3, 779,   0, 197,   0,   5],
       [  1,   0,   0,  19,   2, 684,   0, 290,   0,   4],
       [  3,   5,   0,   9,   1, 716,   0, 252,   0,  14],
       [  1,   5,   0,  18,   2, 597,   0, 334,   0,  43],
       [ 15,  34,   0,   3,   0, 469,   0, 423,   0,  56],
       [ 27, 149,   0,   2,   1, 319,   0, 353,   0, 149]])
```

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`.
```

```
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
True label
airplane 7 36 0 4 0 433 0 484 0 36
automobile 61 141 0 2 1 250 0 399 0 146
bird 1 0 0 10 0 737 0 250 0 2
cat 0 1 0 8 3 685 0 295 0 8
deer 0 0 0 16 3 779 0 197 0 5
dog 1 0 0 19 2 684 0 290 0 4
frog 3 5 0 9 1 716 0 252 0 14
horse 1 5 0 18 2 597 0 334 0 43
ship 15 34 0 3 0 469 0 423 0 56
truck 27 149 0 2 1 319 0 353 0 149
bird cat deer dog frog horse ship truck
600
500
400
300
200
100