In [4]:

```
#Importing library

import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
```

In [5]:

```
np.random.seed(1000)
```

In [6]:

```
#Instantiation
AlexNet = Sequential()

#1st Convolutional Layer
AlexNet.add(Conv2D(filters=96, input_shape=(32,32,3), kernel_size=(11,11), strides=(4,4), p
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))
```

In [7]:

```
#2nd Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(5, 5), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))
```

In [8]:

```
#3rd Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
```

In [9]:

```
#4th Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
```

In [10]:

```
#5th Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))
```

In [11]:

```
#Passing it to a Fully Connected layer
AlexNet.add(Flatten())
# 1st Fully Connected Layer
AlexNet.add(Dense(4096, input_shape=(32,32,3,)))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
# Add Dropout to prevent overfitting
AlexNet.add(Dropout(0.4))
```

In [12]:

```
#2nd Fully Connected Layer
AlexNet.add(Dense(4096))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
#Add Dropout
AlexNet.add(Dropout(0.4))
```

In [13]:

```
#3rd Fully Connected Layer
AlexNet.add(Dense(1000))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
#Add Dropout
AlexNet.add(Dropout(0.4))
```

In [14]:

```
#Output Layer
AlexNet.add(Dense(10))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('softmax'))
```

In [15]:

```
#Model Summary
AlexNet.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 8, 8, 96)          34944
_________________________________________________________________
batch_normalization (BatchNo (None, 8, 8, 96)          384
_________________________________________________________________
activation (Activation)      (None, 8, 8, 96)          0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 4, 4, 96)          0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 4, 4, 256)         614656
_________________________________________________________________
batch_normalization_1 (Batch (None, 4, 4, 256)         1024
_________________________________________________________________
activation_1 (Activation)    (None, 4, 4, 256)         0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 2, 2, 384)         885120
_________________________________________________________________
batch_normalization_2 (Batch (None, 2, 2, 384)         1536
_________________________________________________________________
activation_2 (Activation)    (None, 2, 2, 384)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 2, 2, 384)         1327488
_________________________________________________________________
batch_normalization_3 (Batch (None, 2, 2, 384)         1536
_________________________________________________________________
activation_3 (Activation)    (None, 2, 2, 384)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 2, 2, 256)         884992
_________________________________________________________________
batch_normalization_4 (Batch (None, 2, 2, 256)         1024
_________________________________________________________________
activation_4 (Activation)    (None, 2, 2, 256)         0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 4096)              1052672
_________________________________________________________________
batch_normalization_5 (Batch (None, 4096)              16384
_________________________________________________________________
activation_5 (Activation)    (None, 4096)              0
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
batch_normalization_6 (Batch (None, 4096)              16384
_________________________________________________________________
activation_6 (Activation)    (None, 4096)              0
```

```
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
batch_normalization_7 (Batch (None, 1000)              4000
_________________________________________________________________
activation_7 (Activation)    (None, 1000)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                10010
_________________________________________________________________
batch_normalization_8 (Batch (None, 10)                40
_________________________________________________________________
activation_8 (Activation)    (None, 10)                0
=================================================================
Total params: 25,730,506
Trainable params: 25,709,350
Non-trainable params: 21,156
_________________________________________________________________
```

In [16]:

```
# Compiling the model
AlexNet.compile(loss = keras.losses.categorical_crossentropy, optimizer= 'adam', metrics=['
```

In [17]:

```
#Keras library for CIFAR dataset
from keras.datasets import cifar10
(x_train, y_train),(x_test, y_test)=cifar10.load_data()

#Train-validation-test split
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=.3)
```

In [18]:

```
#Dimension of the CIFAR10 dataset
print((x_train.shape,y_train.shape))
print((x_val.shape,y_val.shape))
print((x_test.shape,y_test.shape))
```

```
((35000, 32, 32, 3), (35000, 1))
((15000, 32, 32, 3), (15000, 1))
((10000, 32, 32, 3), (10000, 1))
```

In [19]:

```
#Onehot Encoding the labels.
from sklearn.utils.multiclass import unique_labels
from keras.utils import to_categorical
```

In [20]:

```
#Since we have 10 classes we should expect the shape[1] of y_train,y_val and y_test to chan
y_train=to_categorical(y_train)
y_val=to_categorical(y_val)
y_test=to_categorical(y_test)
```

In [21]:

```
#Verifying the dimension after one hot encoding
print((x_train.shape,y_train.shape))
print((x_val.shape,y_val.shape))
print((x_test.shape,y_test.shape))
```

```
((35000, 32, 32, 3), (35000, 10))
((15000, 32, 32, 3), (15000, 10))
((10000, 32, 32, 3), (10000, 10))
```

In [22]:

```
#Learning Rate Annealer
from keras.callbacks import ReduceLROnPlateau
lrr= ReduceLROnPlateau(   monitor='val_acc',   factor=.01,   patience=3,  min_lr=1e-5)
```

In [23]:

```
#Defining the parameters
batch_size= 100
epochs=100
learn_rate=.001
```

In [35]:

```
#Training the model
AlexNet.fit(x_train, y_train)
```

```
1094/1094 [==============================] - 1021s 808ms/step - loss: 1.8078
- accuracy: 0.3486
```

Out[35]:

```
<tensorflow.python.keras.callbacks.History at 0x1086a5784f0>
```

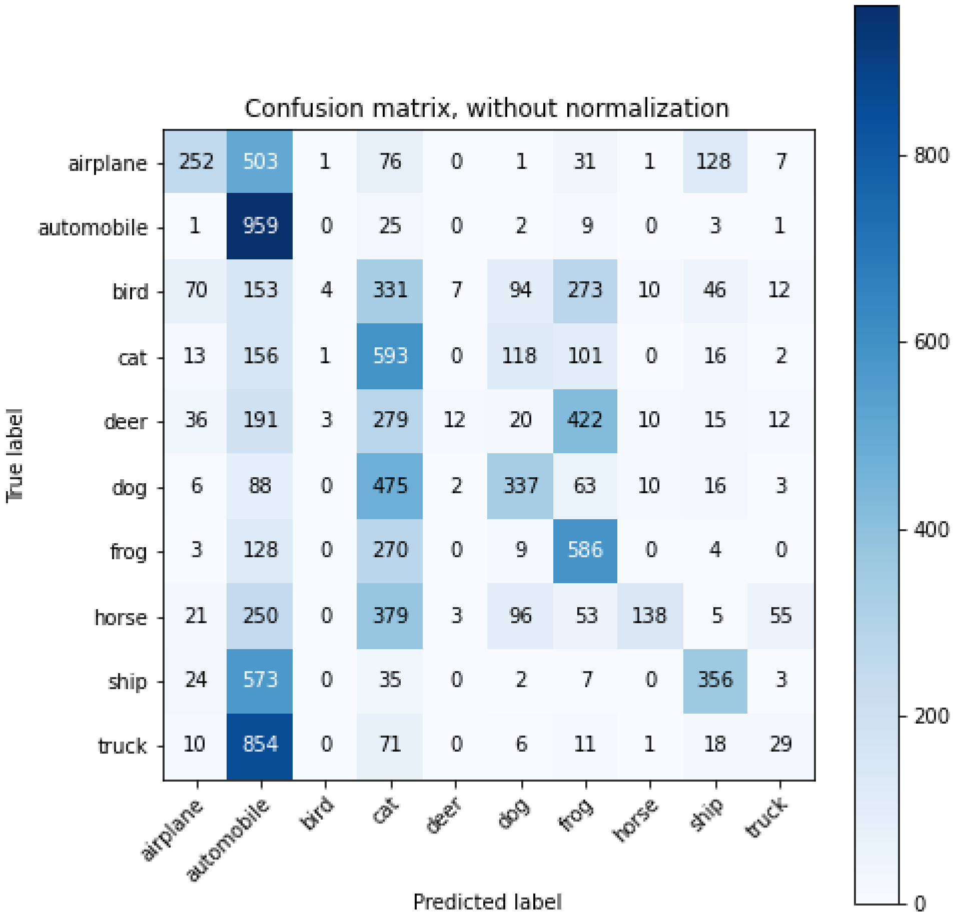
In [44]:

```
#Defining function for confusion matrix plot
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

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

#Print Confusion matrix
    fig, ax = plt.subplots(figsize=(7,7))
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

 # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")
    # Loop over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax

np.set_printoptions(precision=2)

#Making prediction
y_pred=AlexNet.predict_classes(x_test)
y_true=np.argmax(y_test,axis=1)

#Plotting the confusion matrix
from sklearn.metrics import confusion_matrix
confusion_mtx=confusion_matrix(y_true,y_pred)

class_names=['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship

# Plotting non-normalized confusion matrix
plot_confusion_matrix(y_true, y_pred, classes = class_names,title = 'Confusion matrix, with
```

```
Confusion matrix, without normalization
```

Out[44]:

```
<AxesSubplot:title={'center':'Confusion matrix, without normalization'}, xla
bel='Predicted label', ylabel='True label'>
```



In [45]:

```
# Task 2



import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.optimizers import Adam
from keras.layers.normalization import BatchNormalization
from keras.utils import np_utils
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D, GlobalAveragePooling2D
from keras.layers.advanced_activations import LeakyReLU
from keras.preprocessing.image import ImageDataGenerator
```

In [47]:

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dat
asets/mnist.npz (https://storage.googleapis.com/tensorflow/tf-keras-dataset
s/mnist.npz)
11493376/11490434 [==============================] - 1s 0us/step
```

In [48]:

```
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

X_train/=255
X_test/=255
```

In [49]:

```
number_of_classes = 10

Y_train = np_utils.to_categorical(y_train, number_of_classes)
Y_test = np_utils.to_categorical(y_test, number_of_classes)
```

In [50]:

```
model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape=(28,28,1)))
model.add(BatchNormalization(axis=-1))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(BatchNormalization(axis=-1))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64,(3, 3)))
model.add(BatchNormalization(axis=-1))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(BatchNormalization(axis=-1))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
# Fully connected layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(10))

model.add(Activation('softmax'))
```

In [51]:

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

In [52]:

```
gen = ImageDataGenerator(rotation_range=8, width_shift_range=0.08, shear_range=0.3,
                         height_shift_range=0.08, zoom_range=0.08)

test_gen = ImageDataGenerator()
```

In [53]:

```
train_generator = gen.flow(X_train, Y_train, batch_size=64)
test_generator = test_gen.flow(X_test, Y_test, batch_size=64)
```

In [54]:

```
model.fit_generator(train_generator, steps_per_epoch=60000//64, epochs=5,
                    validation_data=test_generator, validation_steps=10000//64)
```

```
C:\Users\VRINDA\anaconda3\lib\site-packages\tensorflow\python\keras\engine\t
raining.py:1844: UserWarning: `Model.fit_generator` is deprecated and will b
e removed in a future version. Please use `Model.fit`, which supports genera
tors.
  warnings.warn('`Model.fit_generator` is deprecated and '

Epoch 1/5
937/937 [==============================] - 328s 328ms/step - loss: 0.2618 -
accuracy: 0.9186 - val_loss: 0.0348 - val_accuracy: 0.9884
Epoch 2/5
937/937 [==============================] - 279s 298ms/step - loss: 0.0586 -
accuracy: 0.9815 - val_loss: 0.0342 - val_accuracy: 0.9881
Epoch 3/5
937/937 [==============================] - 285s 304ms/step - loss: 0.0446 -
accuracy: 0.9865 - val_loss: 0.0339 - val_accuracy: 0.9892
Epoch 4/5
937/937 [==============================] - 307s 328ms/step - loss: 0.0392 -
accuracy: 0.9885 - val_loss: 0.0377 - val_accuracy: 0.9874
Epoch 5/5
937/937 [==============================] - 284s 303ms/step - loss: 0.0303 -
accuracy: 0.9907 - val_loss: 0.0248 - val_accuracy: 0.9924
```

Out[54]:

```
<tensorflow.python.keras.callbacks.History at 0x108161f96d0>
```

In [ ]:

```
```