SML PROJECT REPORT CIFAR-10 DATASET CLASSIFICATION

Farhan Rasheed Chughtai-002960063

Methods used for classification of the Dataset:

Logistic Regression:

A basic linear model used for classification problems. It uses the probability of one event taking place by having the log odds for the event be a linear combination of one or more variables.

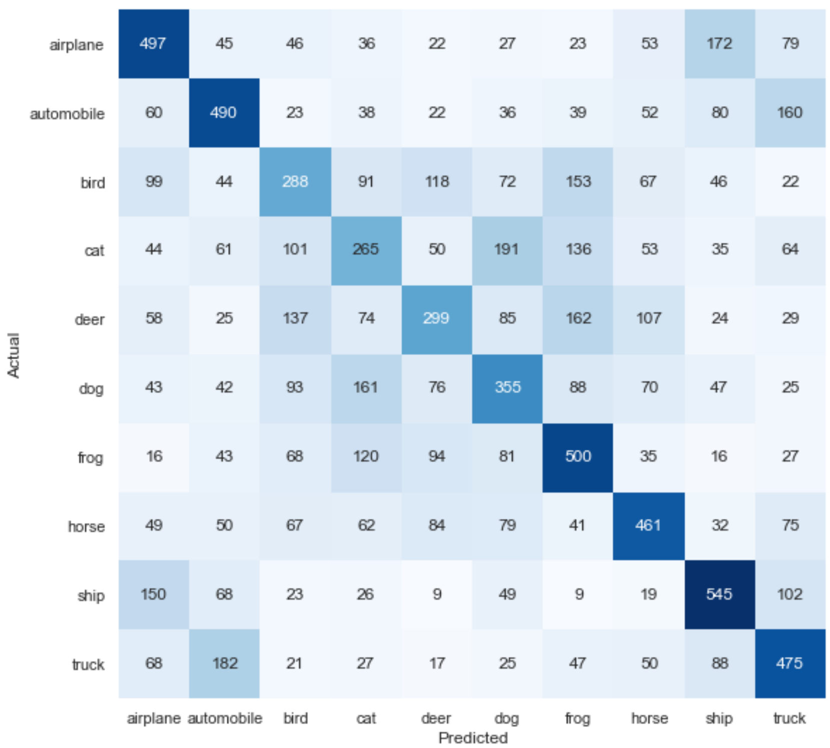
Deep Neural Networks with Convolution Layers:

Deep Neural Networks is a multi-layer model which comprises of layers and each layers have many neurons which have activation functions like sigmoid or relu. There is an initial input layer and an output layer and many in between hidden layers consisting of connected neurons. In addition to the neural network, I am going to be using convolution layers which are used to detect features in an image and after feature selection from convolution layers the output is flattened and passed onto a fully connected neural network with a SoftMax function at the output layer to classify each image into the 10 categories we have in the dataset.

Analysis of the Results:

Logistic Regression:

Confusion Matrix:



Classification Report:

	precision	recall	f1-score	support
airplane	0.46	0.50	0.48	1000
automobile	0.47	0.49	0.48	1000
bird	0.33	0.29	0.31	1000
cat	0.29	0.27	0.28	1000
deer	0.38	0.30	0.33	1000
dog	0.35	0.35	0.36	1000
frog	0.42	0.50	0.45	1000
horse	0.48	0.46	0.47	1000
ship	0.50	0.55	0.52	1000
truck	0.45	0.47	0.46	1000
accuracy			0.42	10000
macro avg	0.41	0.42	0.41	10000
weighted avg	0.41	0.42	0.41	10000

For Logistic Regression the highest f1-score was of identifying airplanes and automobiles so it is performing best at identifying these two objects.

Accuracy of Model:

41.75%

Deep Neural Networks with Convolution Layers:

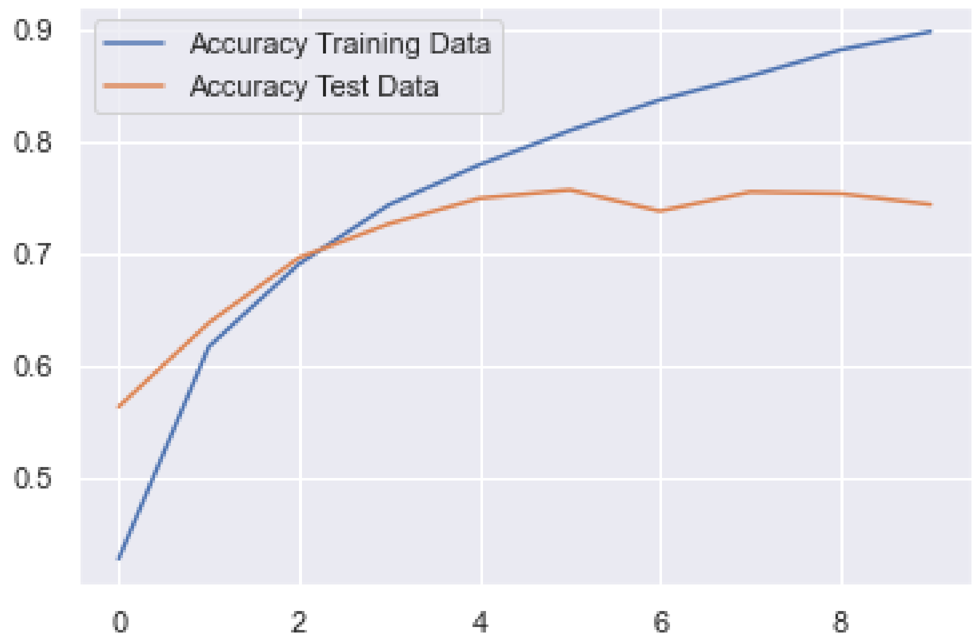
Confusion Matrix:

Classification Report:

	precision	recall	f1-score	support
airplane	0.76	0.78	0.77	1000
automobile	0.91	0.86	0.88	1000
bird	0.66	0.57	0.61	1000
cat	0.50	0.64	0.56	1000
deer	0.76	0.63	0.69	1000
dog	0.69	0.62	0.65	1000
frog	0.78	0.83	0.80	1000
horse	0.83	0.75	0.79	1000
ship	0.78	0.89	0.83	1000
truck	0.83	0.88	0.85	1000
accuracy			0.74	10000
macro avg	0.75	0.74	0.74	10000
weighted avg	0.75	0.74	0.74	10000

For Deep Neural Network the highest f1-score was of automobile so it was able to detect automobiles with a very high accuracy.

Accuracy trend throughout the Epochs with respect to Training and Testing Data:



We can see at the end that our deep learning model is overfitting at the end as accuracy on training data increases but accuracy on test data does not increase.

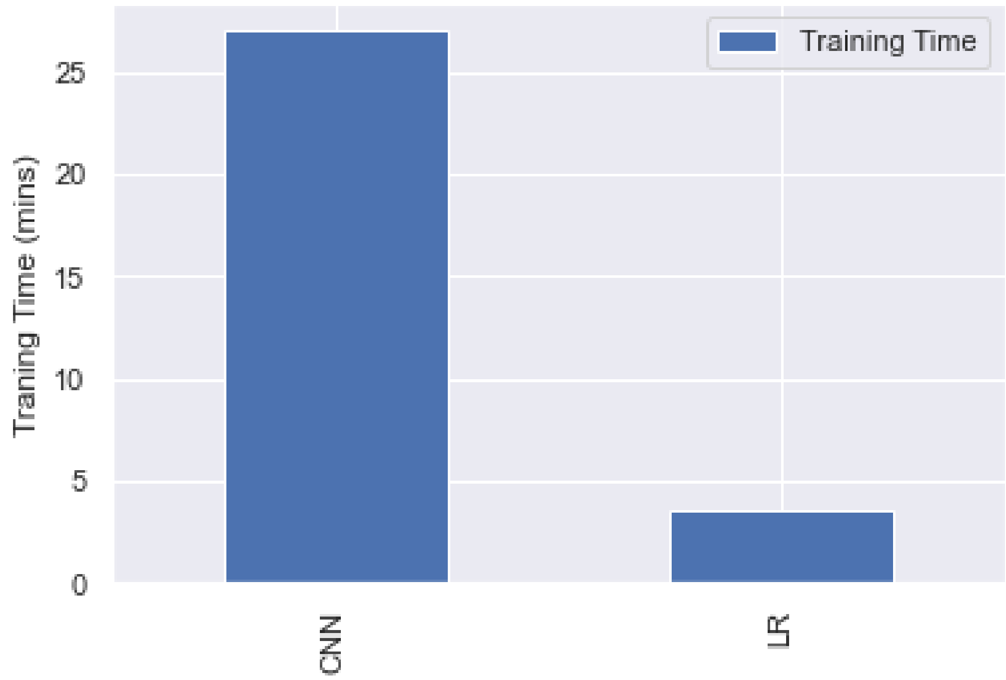
Accuracy of Model:

74.38%

Comparison of Results:

Convergence Speed and Overall Training Time:

For logistic regression the overall training time and convergence speed was very fast as logistic regression is a simple model which calculates the probability of one event taking place and it does not have any layers. The Number of computations was also very less. Compared to deep neural network with convolution layers. I ran the deep neural network for 10 epochs and the model had 9 hidden layers each epochs took about 160 seconds to complete so in total the whole training took about 1600 seconds which is about 27 minutes. Which is very high in comparison to logistic regression which was only around 3.5 minutes.

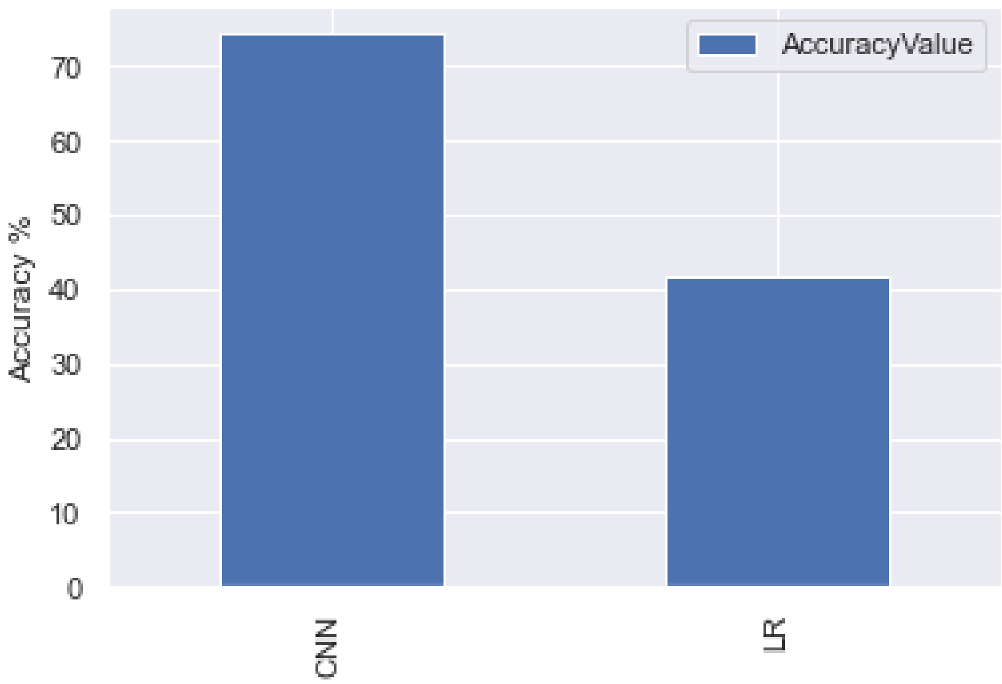


Resources Used:

Logistic Regression used a bare minimum of my computer resources to train and a lot less computations were involved compared to the deep neural network with convolution layers which was using around 95 – 99 percent of CPU usage for about 27 minutes so deep learning networks are very resource hungry to train.

Accuracy:

This is where logistic regression was very terrible as it was only about to get an accuracy of 41.75 % which is below the 50% threshold as compared to the deep neural network which had an accuracy of 74.38% about 80 percent better than the logistic regression model which is huge and should be the case because deep learning model is many times more complicated than a simple model than logistics regression and it uses the convolution layers to detect the features instead of just looking at each pixel value.



Conclusion:

Deep neural network with Convolution layers was considerably better than a simple logistic regression model for image classification of this dataset. Which is primary because of the convolution layers that are present in the deep neural network.

SML Project

May 5, 2022

1 Project Code

[18]:
```
import tensorflow
import keras
from keras.datasets import cifar10
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
import sklearn.metrics as metrics
from sklearn.metrics import classification_report, confusion_matrix,␣
↪f1_score,accuracy_score, precision_score, recall_score, roc_auc_score
from sklearn.model_selection import RepeatedStratifiedKFold, StratifiedKFold
from sklearn.preprocessing import OneHotEncoder
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
import seaborn as sns; sns.set()
from tensorflow.keras import datasets, layers, models
import time
from datetime import datetime

import warnings
warnings.filterwarnings(action="ignore")
```

1.1 Loading and Exploring the Data Set

[2]:
```
# define num_class
num_classes = 10

# load dataset keras will download cifar-10 datset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

Looking and the Shape and see the first images in the data set

[3]:
```
print("x_train shape : ",x_train.shape)
print("Y_train sahpe : ",y_train.shape)
```

```
x_train shape :  (50000, 32, 32, 3)
Y_train sahpe :  (50000, 1)
```

Chaging shape of Y values to single dimension number array

[4]:
```python
y_train = y_train.reshape(-1,)
y_test = y_test.reshape(-1,)
print("y_train",y_train)
print("y_train shape",y_train.shape)
```

```
y_train [6 9 9 … 9 1 1]
y_train shape (50000,)
```

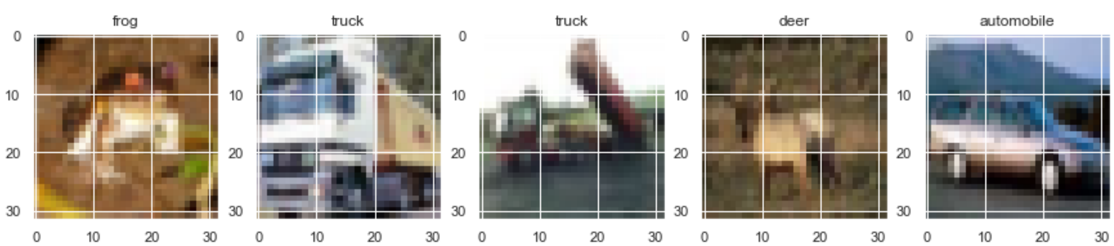
Setting up the labels for the data

[5]:
```python
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',␣
 ↪'horse', 'ship', 'truck']
```

Printing out the first few pictures with labels

[7]:
```python
fig, axes = plt.subplots(ncols=5,figsize=(15, 15))

for i in range(5):
    axes[i].set_title(labels[y_train[i]])
    axes[i].imshow(x_train[i])
```



2 Staring Machine Learning with First Model Linear Regression

2.0.1 Preprocessing the data to be passed into the Model

[6]:
```python
x_train_reshapedto2d = x_train.reshape(50000,32*32*3)
x_test_reshapedto2d = x_test.reshape(10000 ,32*32*3)
```

Printing out new shape of data to be passed to Logistic Regression Modelb

[7]:
```python
print("X_train shape",x_train_reshapedto2d.shape)
print("X_test shape",x_test_reshapedto2d.shape)
```

```
X_train shape (50000, 3072)
X_test shape (10000, 3072)
```

Normalizing the dataset by dividing each value by 255 beacause each value is RGB value which ranges from 0 -255 so diving it by 255 will give us a number between 0 - 1.

[8]:
```
x_train_reshapedto2d_normalized = x_train_reshapedto2d/255
x_test_reshapedto2d_normalized = x_test_reshapedto2d/255
```

Printing out the first few lines of data to be passed to the model

[9]:
```
x_train_reshapedto2d_normalized
```

[9]:
```
array([[0.23137255, 0.24313725, 0.24705882, …, 0.48235294, 0.36078431,
        0.28235294],
       [0.60392157, 0.69411765, 0.73333333, …, 0.56078431, 0.52156863,
        0.56470588],
       [1.        , 1.        , 1.        , …, 0.31372549, 0.3372549 ,
        0.32941176],
       …,
       [0.1372549 , 0.69803922, 0.92156863, …, 0.04705882, 0.12156863,
        0.19607843],
       [0.74117647, 0.82745098, 0.94117647, …, 0.76470588, 0.74509804,
        0.67058824],
       [0.89803922, 0.89803922, 0.9372549 , …, 0.63921569, 0.63921569,
        0.63137255]])
```

2.0.2 Training the Logistic Regression Model

Setting the parameters of Logistic Regression Model. We are going to use sparse regression with l2 penality.

[21]:
```
logisticRegStartTime = time.time()
logregmodel = LogisticRegression(C=0.01, multi_class='multinomial',␣
 ↪solver='sag',penalty='l2')
logregmodel.fit(x_train_reshapedto2d_normalized,y_train)
logisticRegEndTime = time.time()
```

[24]:
```
time_intervalLG = logisticRegEndTime - logisticRegStartTime
print("Time Taken To Train The data ",time_intervalLG/60)
```

```
Time Taken To Train The data  3.5329426407814024
```

2.0.3 Predicting and Collecting the Results

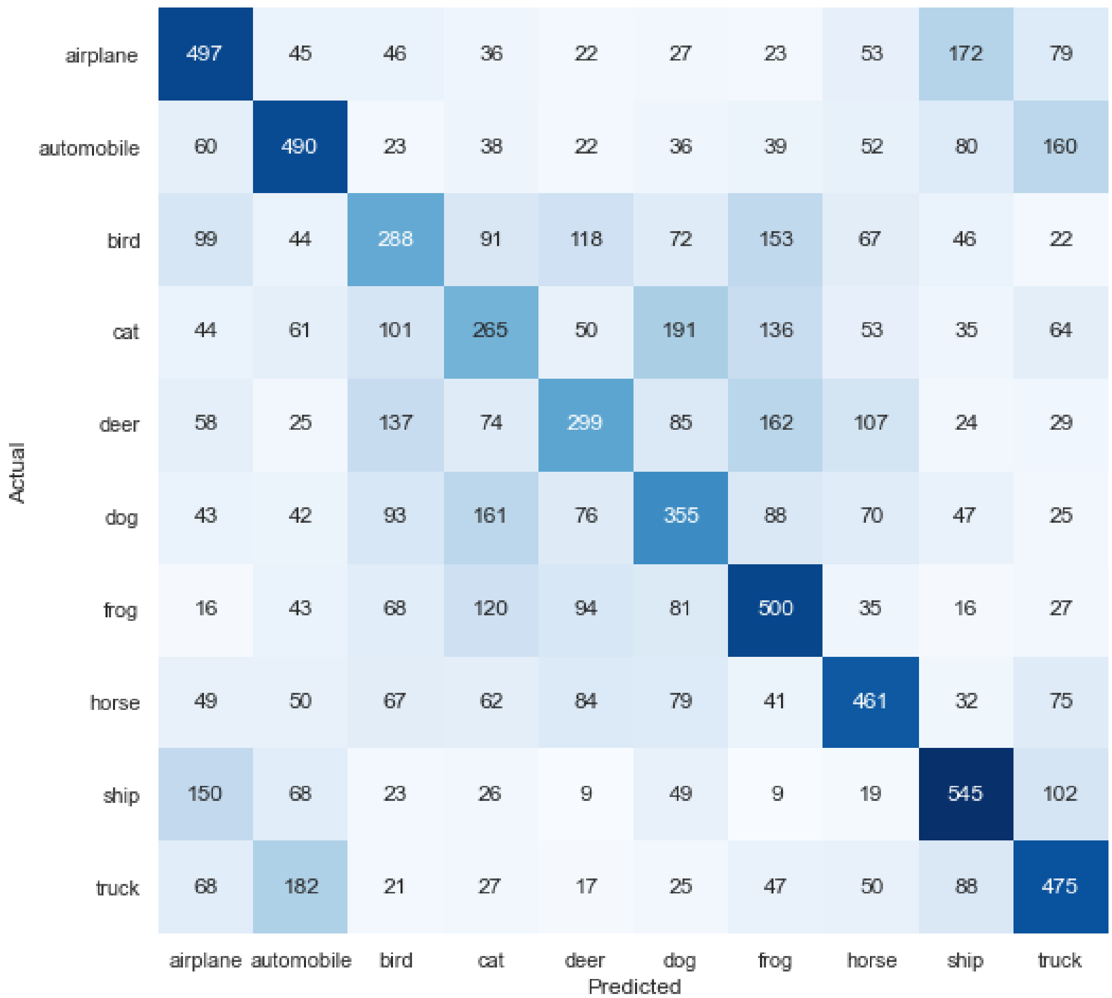
[26]:
```
predictionsLogReg = logregmodel.predict(x_test_reshapedto2d_normalized)
```

[27]:
```
conmaxReg = confusion_matrix(y_test, predictionsLogReg)
```

Confusion Matrix of the Result

[28]:
```
plt.figure(figsize=(9,9))
```

```
sns.heatmap(conmaxReg, cbar=False, xticklabels=labels, yticklabels=labels,fmt='d',annot=True, cmap=plt.cm.Blues)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Classificaiton Report of the Classes

[29]:
```
print(classification_report(y_test,predictionsLogReg,target_names=labels))
```

```
              precision    recall  f1-score   support

    airplane       0.46      0.50      0.48      1000
  automobile       0.47      0.49      0.48      1000
        bird       0.33      0.29      0.31      1000
         cat       0.29      0.27      0.28      1000
        deer       0.38      0.30      0.33      1000
```

```
         dog       0.35      0.35      0.36      1000
        frog       0.42      0.50      0.45      1000
       horse       0.48      0.46      0.47      1000
        ship       0.50      0.55      0.52      1000
       truck       0.45      0.47      0.46      1000

    accuracy                           0.42     10000
   macro avg       0.41      0.42      0.41     10000
weighted avg       0.41      0.42      0.41     10000
```

Overall Accuracy Of the Mode

[30]:
```
print("Accuray of Logistic Regression Model is :␣
↪",accuracy_score(predictionsLogReg,y_test))
```

```
Accuray of Logistic Regression Model is :  0.4175
```

3 Machine Learning with Deep Neural Network with Convolution Layer

Traning a simple neural network

[32]:
```
model2 = keras.Sequential([
    keras.layers.Dense(3072, input_shape=(3072,), activation='relu'),
    keras.layers.Dense(1536, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

model2.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model2.fit(x_train_reshapedto2d_normalized, y_train, epochs=10)
```

```
Epoch 1/10
1563/1563 [==============================] - 65s 42ms/step - loss: 1.8888 -
accuracy: 0.3256
Epoch 2/10
1563/1563 [==============================] - 66s 42ms/step - loss: 1.6716 -
accuracy: 0.3983
Epoch 3/10
1563/1563 [==============================] - 60s 38ms/step - loss: 1.6011 -
accuracy: 0.4251
Epoch 4/10
1563/1563 [==============================] - 60s 39ms/step - loss: 1.5537 -
accuracy: 0.4427
Epoch 5/10
1563/1563 [==============================] - 62s 40ms/step - loss: 1.5189 -
```

```
accuracy: 0.4531
Epoch 6/10
1563/1563 [==============================] - 62s 40ms/step - loss: 1.4923 -
accuracy: 0.4663
Epoch 7/10
1563/1563 [==============================] - 62s 40ms/step - loss: 1.4706 -
accuracy: 0.4747
Epoch 8/10
1563/1563 [==============================] - 63s 40ms/step - loss: 1.4542 -
accuracy: 0.4806
Epoch 9/10
1563/1563 [==============================] - 61s 39ms/step - loss: 1.4372 -
accuracy: 0.4852
Epoch 10/10
1563/1563 [==============================] - 63s 41ms/step - loss: 1.4199 -
accuracy: 0.4913
```

[32]:
```
<keras.callbacks.History at 0x290414f1640>
```

Checking Accuracy of the Deep Neural Network

[33]:
```
model2.evaluate(x_test_reshapedto2d_normalized,y_test)
```

```
313/313 [==============================] - 2s 6ms/step - loss: 1.4737 -
accuracy: 0.4732
```

[33]:
```
[1.4736998081207275, 0.4731999933719635]
```

Testing and training with convolution layers added in the Neural Network

[31]:
```
x_train_covnn = x_train/255
x_test_covnn = x_test/255
```

[39]:
```
model4 = keras.Sequential([
    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu',␣
 ↪input_shape=(32,32,3),strides=1,padding="same"),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3),␣
 ↪activation='relu',strides=1,padding="same"),
    keras.layers.Conv2D(filters=128, kernel_size=(3, 3),␣
 ↪activation='relu',strides=1,padding="same"),
    keras.layers.MaxPooling2D((2, 2)),
    #keras.layers.AveragePooling2D((2, 2)),

    keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu'),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    keras.layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    #keras.layers.AveragePooling2D((2, 2)),
```

```
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

model4.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

DNNCNNstarttime = time.time()
ModelTrainingData = model4.fit(x_train_covnn, y_train,␣
↪epochs=10,batch_size=50,validation_data=(x_test_covnn,y_test))
DNNCNNEndtime = time.time()
```

```
Epoch 1/10
1000/1000 [==============================] - 161s 161ms/step - loss: 1.5576 -
accuracy: 0.4268 - val_loss: 1.2132 - val_accuracy: 0.5629
Epoch 2/10
1000/1000 [==============================] - 157s 157ms/step - loss: 1.0785 -
accuracy: 0.6167 - val_loss: 1.0396 - val_accuracy: 0.6380
Epoch 3/10
1000/1000 [==============================] - 159s 159ms/step - loss: 0.8698 -
accuracy: 0.6906 - val_loss: 0.8521 - val_accuracy: 0.6962
Epoch 4/10
1000/1000 [==============================] - 161s 161ms/step - loss: 0.7293 -
accuracy: 0.7436 - val_loss: 0.7974 - val_accuracy: 0.7266
Epoch 5/10
1000/1000 [==============================] - 163s 163ms/step - loss: 0.6254 -
accuracy: 0.7793 - val_loss: 0.7352 - val_accuracy: 0.7493
Epoch 6/10
1000/1000 [==============================] - 163s 163ms/step - loss: 0.5362 -
accuracy: 0.8099 - val_loss: 0.7215 - val_accuracy: 0.7569
Epoch 7/10
1000/1000 [==============================] - 163s 163ms/step - loss: 0.4597 -
accuracy: 0.8374 - val_loss: 0.8014 - val_accuracy: 0.7379
Epoch 8/10
1000/1000 [==============================] - 158s 158ms/step - loss: 0.3964 -
accuracy: 0.8586 - val_loss: 0.7822 - val_accuracy: 0.7550
Epoch 9/10
1000/1000 [==============================] - 170s 170ms/step - loss: 0.3314 -
accuracy: 0.8822 - val_loss: 0.8295 - val_accuracy: 0.7536
Epoch 10/10
1000/1000 [==============================] - 165s 165ms/step - loss: 0.2826 -
accuracy: 0.8984 - val_loss: 0.9567 - val_accuracy: 0.7438
```

[40]:
```
model4.summary()
```

```
Model: "sequential_7"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_50 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 32, 32, 64)        18496
_________________________________________________________________
conv2d_52 (Conv2D)           (None, 32, 32, 128)       73856
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
conv2d_53 (Conv2D)           (None, 14, 14, 32)        36896
_________________________________________________________________
conv2d_54 (Conv2D)           (None, 12, 12, 64)        18496
_________________________________________________________________
conv2d_55 (Conv2D)           (None, 10, 10, 128)       73856
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 5, 5, 128)         0
_________________________________________________________________
flatten_7 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_21 (Dense)             (None, 128)               409728
_________________________________________________________________
dense_22 (Dense)             (None, 10)                1290
=================================================================
Total params: 633,514
Trainable params: 633,514
Non-trainable params: 0
_________________________________________________________________
```

[42]:
```
time_intervalDNNCNN = DNNCNNEndtime - DNNCNNstarttime
print("Time Taken To Train The data ",time_intervalDNNCNN/60)
```

```
Time Taken To Train The data  27.020609664916993
```

Accuracy Graphs

[43]:
```
plt.plot(ModelTrainingData.history['accuracy'],label='Accuracy Training Data')
plt.plot(ModelTrainingData.history['val_accuracy'],label='Accuracy Test Data')
plt.legend()
plt.show()
```

Checking the accuracy of the CNN Model

[44]:
```
model4.evaluate(x_test_covnn,y_test)
```

```
313/313 [==============================] - 9s 27ms/step - loss: 0.9567 -
accuracy: 0.7438
```

[44]:
```
[0.956702470779419, 0.7437999844551086]
```

Making the confusion matrix

[45]:
```
predictions = model4.predict(x_test_covnn)
```

[47]:
```
predictionsCNN = [np.argmax(i) for i in predictions]
```

[48]:
```
conmaxCNN = confusion_matrix(y_test, predictionsCNN)
```

[49]:
```
plt.figure(figsize=(9,9))
sns.heatmap(conmaxCNN, cbar=False, xticklabels=labels,␣
↪yticklabels=labels,fmt='d',annot=True, cmap=plt.cm.Blues)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

[50]:
```
print(classification_report(y_test,predictionsCNN,target_names=labels))
```

```
              precision    recall  f1-score   support

    airplane       0.76      0.78      0.77      1000
  automobile       0.91      0.86      0.88      1000
        bird       0.66      0.57      0.61      1000
         cat       0.50      0.64      0.56      1000
        deer       0.76      0.63      0.69      1000
         dog       0.69      0.62      0.65      1000
        frog       0.78      0.83      0.80      1000
       horse       0.83      0.75      0.79      1000
        ship       0.78      0.89      0.83      1000
       truck       0.83      0.88      0.85      1000

    accuracy                           0.74     10000
```

```
   macro avg       0.75      0.74      0.74     10000
weighted avg       0.75      0.74      0.74     10000
```

3.0.1 Overall Accuray of the Model

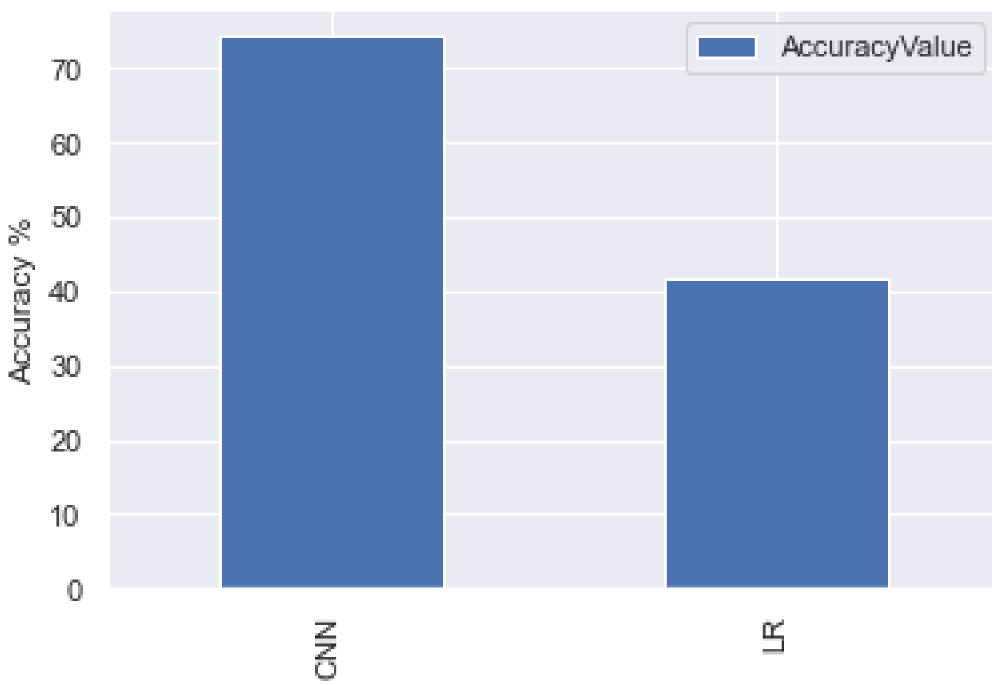
[51]:
```
print("Accuray of CNN Model is : ",accuracy_score(predictionsCNN,y_test))
```

```
Accuray of CNN Model is :  0.7438
```

Accuracy Comparison

[71]:
```
plotdataacc = pd.DataFrame(
    {"AccuracyValue": [74.38,41.75]},
    index=["CNN", "LR"])
plotdataacc.plot(kind="bar")
plt.ylabel("Accuracy %")
```

[71]:
```
Text(0, 0.5, 'Accuracy %')
```



[72]:
```
plotdatatimetaken = pd.DataFrame(
    {"Training Time": [27.02,3.53]},
    index=["CNN", "LR"])
plotdatatimetaken.plot(kind="bar")
plt.ylabel("Traning Time (mins)")
```

```
[72]: Text(0, 0.5, 'Traning Time (mins)')
```