1. To Proceed with the LDF, I first load the images in the dataset with a Sobel Mask, create a Harris feature detector and then a Fisher Classifier. We then use it to check the accuracy as follows.

```
#Loading images with Sobel mask
train,test,count,label = load_cfar10_dataset()

#Creating Feature Detector
feature_detector = harris_detector()
train_features = feature_detector.create_features(train)
test_features = feature_detector.create_features(test)

#Creating Feature Classifier
feature_classifier = fisher()
cm = feature_classifier.Fisher_classifier(train=train_features,test=test_features)
print('Accuracy:%d'%(np.sum(cm*np.eye(10))/np.sum(cm)*100))
#Checking Error Rates
error_rate = np.zeros(10)
for idx,i in enumerate(cm):
    error_rate[idx] = (np.sum(i)-i[idx])
for i,j in zip(error_rate,['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'sl
    print('Class-{}:{}%'.format(j,i))
#Plotting the Graph
plt.figure(figsize=(20,20))
plt.subplot(212)
ax1=plt.subplot(211)
ax2=plt.subplot(212)
for i in range(feature_classifier.no_classes):
    ax1.plot(feature_classifier.class_mean[i], alpha=0.5, label = 'class-{}'.format(i))
ax1.legend(loc=2)
ax1.set_title('Actual')
for i in range(feature_classifier.no_classes):
    ax2.plot(feature_classifier.fisher_class_mean[i], alpha=0.5, label = 'class-{}'.format(i))
ax2.legend(loc=2)
ax2.set_title('Fisher')
plt.savefig('../data/CIFAR_GRAPH.png')
plt.show()
```

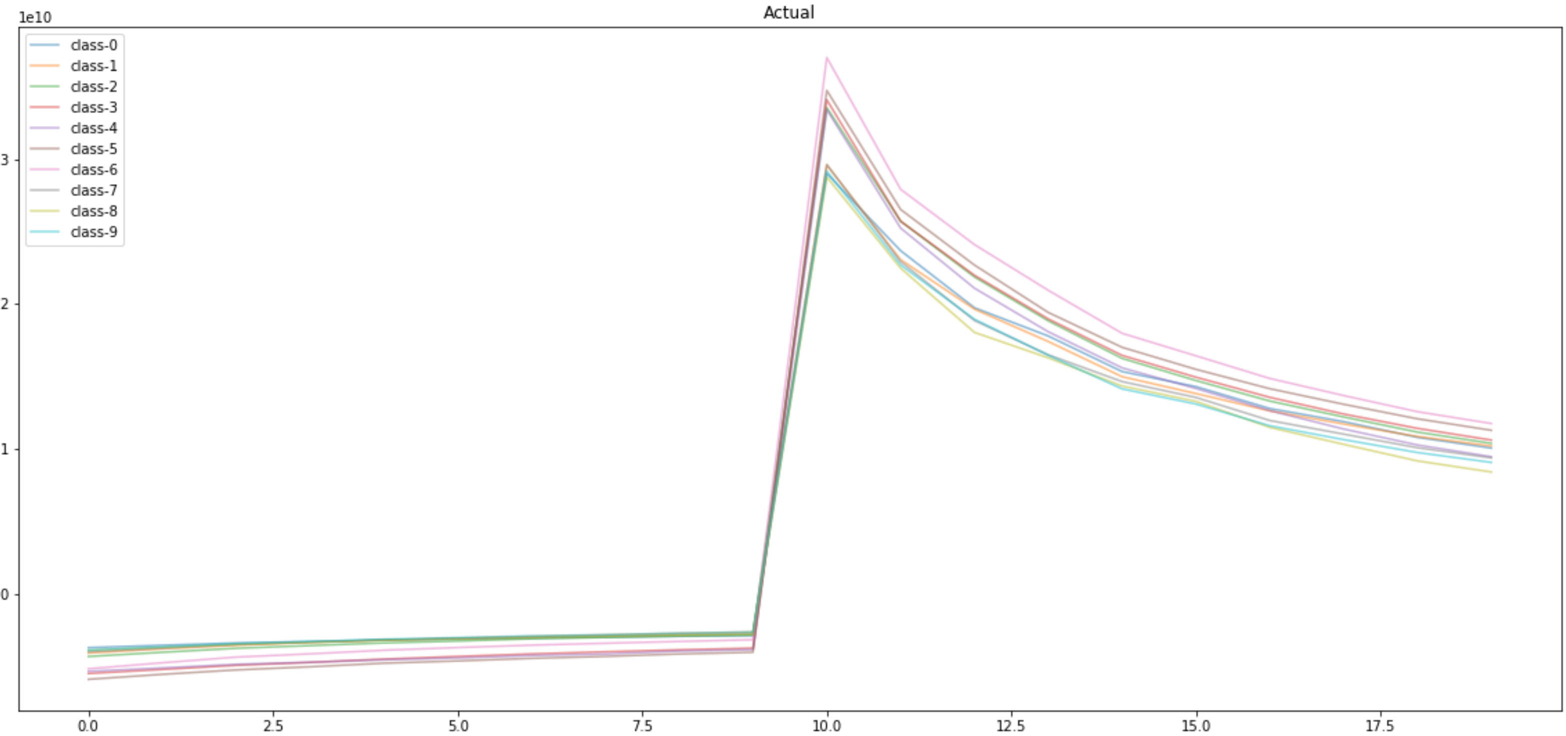
The dataset split for the considered batches are:

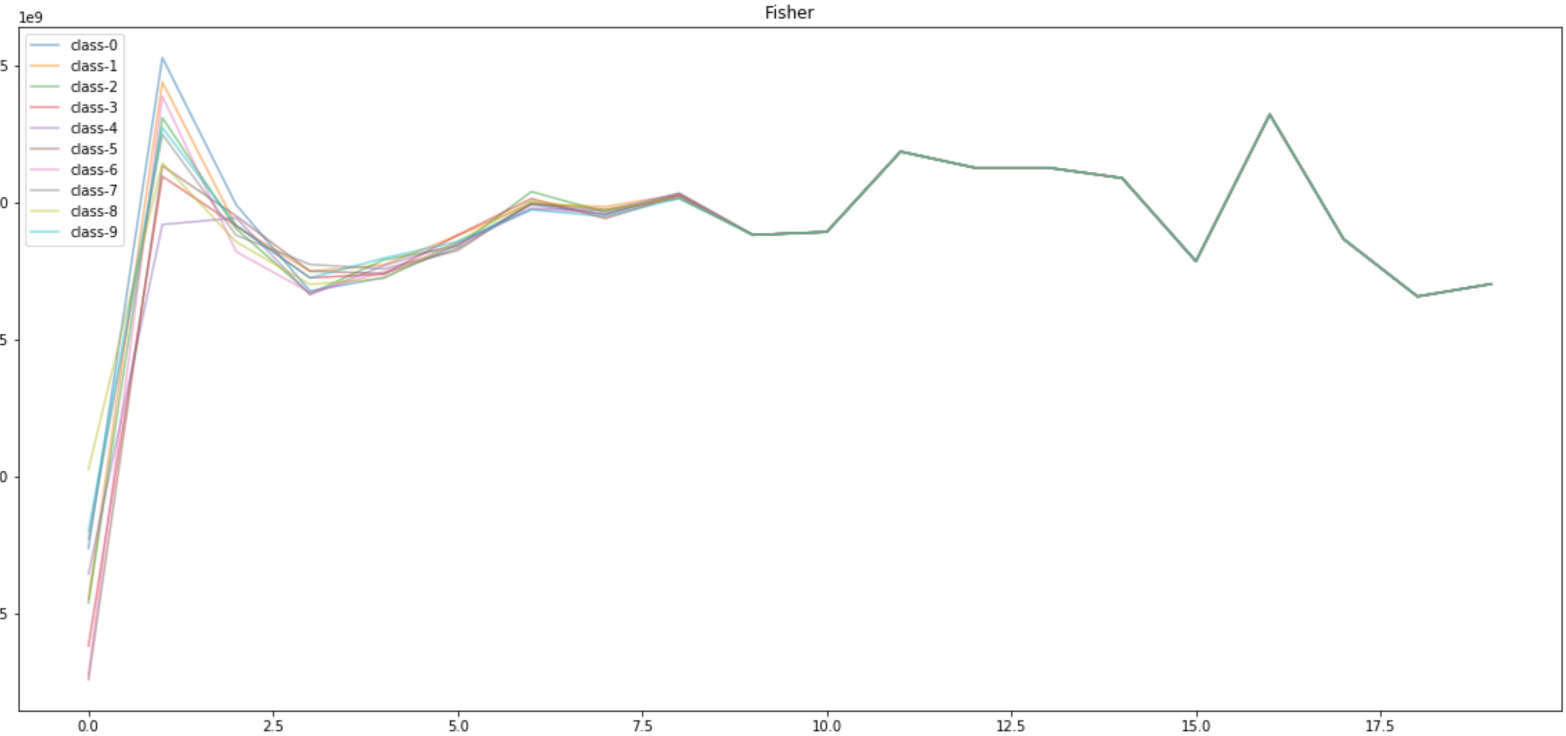
```
Dataset Statistics
Class-label :airplane          train-1005      test-100
Class-label :automobile        train-974       test-100
Class-label :bird              train-1032      test-100
Class-label :cat               train-1016      test-100
Class-label :deer              train-999       test-100
Class-label :dog               train-937       test-100
Class-label :frog              train-1030      test-100
Class-label :horse             train-1001      test-100
Class-label :ship              train-1025      test-100
Class-label :truck             train-981       test-100
```

The confusion matrix, results and graphs obtained on running are as follows:

```
Confusion Matrix is
[[10.  0. 14.  0.  9.  7. 27.  9. 22.  2.]
 [34.  4. 12.  3.  4.  4. 17.  5. 12.  5.]
 [15.  7. 23.  3.  5.  2. 23.  3. 15.  4.]
 [22.  9.  5.  1.  9.  5. 21.  6. 15.  7.]
 [23.  4.  5.  3.  3.  6. 27.  4. 20.  5.]
 [25.  6. 13.  1.  9.  8. 16.  1. 14.  7.]
 [26.  3. 14.  0. 12.  2. 11.  3. 22.  7.]
 [22.  3. 16.  2.  9.  3. 26.  3. 15.  1.]
 [14.  2. 15.  1.  4.  6. 30.  8. 16.  4.]
 [17.  6. 15.  1.  6.  6. 30.  4. 12.  3.]]
```

```
Accuracy:8
Class-airplane:90.0%
Class-automobile:96.0%
Class-bird:77.0%
Class-cat:99.0%
Class-deer:97.0%
Class-dog:92.0%
Class-frog:89.0%
Class-horse:97.0%
Class-ship:84.0%
Class-truck:97.0%
```


Actual
1e10
class-0
class-1
class-2
class-3
class-4
class-5
class-6
class-7
class-8
class-9
3
2
1
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Fisher
1e9
class-0
class-1
class-2
class-3
class-4
class-5
class-6
class-7
class-8
class-9
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

2.

```
#Defining CNN
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 5)
        self.conv3 = nn.Conv2d(64, 64, 5)
        self.fc1 = nn.Linear(64 * 5 * 5, 64)
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 64 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        return F.log_softmax(x)


net = Net()
net
```