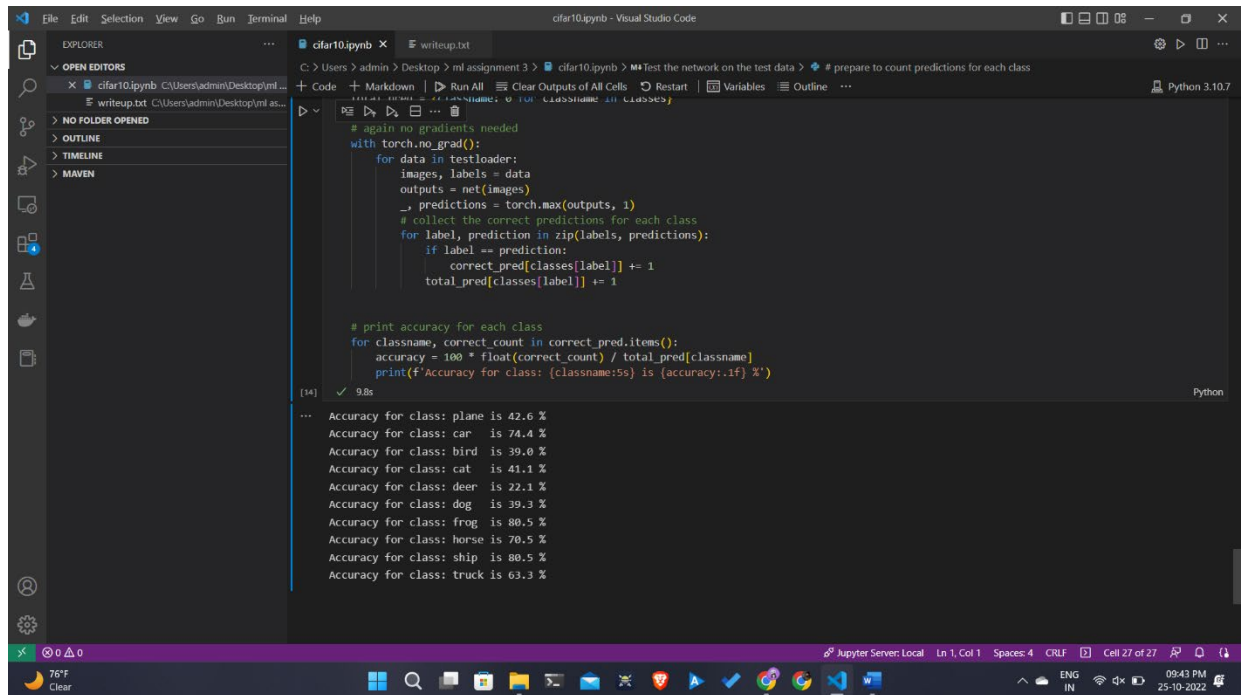


# ML ASSIGNMENT - 03

Name : Pathapati Venkata Sai Saravan  
Campus id : QZ79411

## Part 1

After running the code with out any changes we get the accuracies as



Initial values  
# lr = Learning Rate  
#initial

**lr = 0.001, momentum =0.9**  
**loop =2**

```
[1, 2000] loss: 2.211
[1, 4000] loss: 1.878
[1, 6000] loss: 1.686
[1, 8000] loss: 1.566
[1, 10000] loss: 1.515
[1, 12000] loss: 1.453
[2, 2000] loss: 1.387
[2, 4000] loss: 1.364
[2, 6000] loss: 1.334
[2, 8000] loss: 1.295
[2, 10000] loss: 1.284
[2, 12000] loss: 1.285
```

Finished Training

**Accuracy of the network on the 10000 test images: 55 %**

```
Accuracy for class: plane is 42.6 %
Accuracy for class: car   is 74.4 %
Accuracy for class: bird  is 39.0 %
Accuracy for class: cat   is 41.1 %
Accuracy for class: deer  is 22.1 %
Accuracy for class: dog   is 39.3 %
Accuracy for class: frog  is 80.5 %
Accuracy for class: horse is 70.5 %
Accuracy for class: ship  is 80.5 %
Accuracy for class: truck is 63.3 %
```

-----

## Running Training longer

# looping over the dataset 3 times (initial looping was done 2 times)

**Lr = 0.001 , momentum = 0.9**

**loop =3**

```
[1, 2000] loss: 1.213
[1, 4000] loss: 1.200
[1, 6000] loss: 1.211
[1, 8000] loss: 1.212
[1, 10000] loss: 1.205
[1, 12000] loss: 1.211
[2, 2000] loss: 1.210
[2, 4000] loss: 1.199
[2, 6000] loss: 1.212
[2, 8000] loss: 1.192
[2, 10000] loss: 1.209
[2, 12000] loss: 1.219
[3, 2000] loss: 1.203
[3, 4000] loss: 1.197
[3, 6000] loss: 1.206
[3, 8000] loss: 1.212
[3, 10000] loss: 1.200
[3, 12000] loss: 1.230
Finished Training
```

**Accuracy of the network on the 10000 test images: 55 %**

```
Accuracy for class: plane is 42.6 %
Accuracy for class: car   is 74.4 %
Accuracy for class: bird  is 39.0 %
Accuracy for class: cat   is 41.1 %
Accuracy for class: deer  is 22.1 %
Accuracy for class: dog   is 39.3 %
Accuracy for class: frog  is 80.5 %
Accuracy for class: horse is 70.5 %
Accuracy for class: ship  is 80.5 %
Accuracy for class: truck is 63.3 %
```

-----

## changing the Learning rate and momentum

```
lr= 0.0001 , momentum = 0.95  
loop =2
```

```
[1, 2000] loss: 1.106  
[1, 4000] loss: 1.075  
[1, 6000] loss: 1.076  
[1, 8000] loss: 1.034  
[1, 10000] loss: 1.073  
[1, 12000] loss: 1.075  
[2, 2000] loss: 1.022  
[2, 4000] loss: 1.013  
[2, 6000] loss: 1.004  
[2, 8000] loss: 1.018  
[2, 10000] loss: 1.012  
[2, 12000] loss: 1.014
```

Finished Training

**Accuracy of the network on the 10000 test images: 61 %**

```
Accuracy for class: plane is 42.6 %  
Accuracy for class: car is 74.4 %  
Accuracy for class: bird is 39.0 %  
Accuracy for class: cat is 41.1 %  
Accuracy for class: deer is 22.1 %  
Accuracy for class: dog is 39.3 %  
Accuracy for class: frog is 80.5 %  
Accuracy for class: horse is 70.5 %  
Accuracy for class: ship is 80.5 %  
Accuracy for class: truck is 63.3 %
```

---

## Increase or decrease the number of convolutional filters

```
self.conv1 = nn.Conv2d(3, 6, 5) -> self.conv1 = nn.Conv2d(3, 10, 5)  
self.conv2 = nn.Conv2d(6, 16, 5) -> self.conv2 = nn.Conv2d(6, 20, 5)  
self.fc1 = nn.Linear(16 * 5 * 5, 120) -> self.fc1 = nn.Linear(20 * 5 * 5,  
120)
```

```
lr=0.001, momentum=0.9  
loop =2
```

```
[1, 2000] loss: 2.205  
[1, 4000] loss: 1.806  
[1, 6000] loss: 1.616  
[1, 8000] loss: 1.536  
[1, 10000] loss: 1.473  
[1, 12000] loss: 1.415  
[2, 2000] loss: 1.323  
[2, 4000] loss: 1.301  
[2, 6000] loss: 1.280  
[2, 8000] loss: 1.259  
[2, 10000] loss: 1.217  
[2, 12000] loss: 1.176
```

Finished Training

**Accuracy of the network on the 10000 test images: 58 %**

Accuracy for class: plane is 71.7 %  
Accuracy for class: car is 83.5 %  
Accuracy for class: bird is 40.7 %  
Accuracy for class: cat is 23.1 %  
Accuracy for class: deer is 61.0 %  
Accuracy for class: dog is 60.7 %  
Accuracy for class: frog is 76.6 %  
Accuracy for class: horse is 52.1 %  
Accuracy for class: ship is 63.5 %  
Accuracy for class: truck is 56.0 %

-----

## Change the number or sizes of the fully connected layers

**lr=0.001, momentum=0.9**

**loop =2**

```
self.fc1 = nn.Linear(16 * 5 * 5, 240)
self.fc2 = nn.Linear(240, 120)
self.fc3 = nn.Linear(120, 60)
self.fc4 = nn.Linear(60, 10)
```

```
x = F.relu(self.fc1(x))
x = F.relu(self.fc2(x))
x = F.relu(self.fc3(x))
x = self.fc4(x)
return x
```

```
[1, 2000] loss: 1.209
[1, 4000] loss: 1.193
[1, 6000] loss: 1.197
[1, 8000] loss: 1.180
[1, 10000] loss: 1.179
[1, 12000] loss: 1.165
[2, 2000] loss: 1.087
[2, 4000] loss: 1.109
[2, 6000] loss: 1.074
[2, 8000] loss: 1.083
[2, 10000] loss: 1.050
[2, 12000] loss: 1.053
```

Finished Training

**Accuracy of the network on the 10000 test images: 59 %**

Accuracy for class: plane is 44.5 %  
Accuracy for class: car is 68.4 %  
Accuracy for class: bird is 47.7 %  
Accuracy for class: cat is 42.4 %  
Accuracy for class: deer is 61.8 %  
Accuracy for class: dog is 62.1 %  
Accuracy for class: frog is 62.6 %  
Accuracy for class: horse is 63.3 %

```
Accuracy for class: ship is 81.0 %
Accuracy for class: truck is 62.1 %
```

---

## Change the size/stride of the filters

```
self.pool = nn.MaxPool2d(2, 2) -> self.pool = nn.MaxPool2d(2, 1)
```

```
lr=0.001, momentum=0.9
loop =2
```

```
self.pool = nn.MaxPool2d(2,2) -> self.pool = nn.AdaptiveMaxPool2d((5,5))
```

```
""" You need to modify this linear layer to match the incoming tensor
flattened spatial shape. But notice how this value will depend on the
image that is fed to the CNN, ultimately this will dictate the size of
the tensor fed to the classifier. The general way to solve this is to use
adaptive layers: such as nn.AdaptiveMaxPool2d which will always provide
the same output shape regardless of the input dimension size. """
```

```
[1, 2000] loss: 1.359
[1, 4000] loss: 1.358
[1, 6000] loss: 1.382
[1, 8000] loss: 1.337
[1, 10000] loss: 1.363
[1, 12000] loss: 1.366
[2, 2000] loss: 1.307
[2, 4000] loss: 1.330
[2, 6000] loss: 1.337
[2, 8000] loss: 1.334
[2, 10000] loss: 1.325
[2, 12000] loss: 1.310
Finished Training
```

**Accuracy of the network on the 10000 test images: 52 %**

```
Accuracy for class: plane is 65.7 %
Accuracy for class: car is 46.0 %
Accuracy for class: bird is 32.8 %
Accuracy for class: cat is 58.4 %
Accuracy for class: deer is 57.6 %
Accuracy for class: dog is 33.8 %
Accuracy for class: frog is 47.6 %
Accuracy for class: horse is 53.8 %
Accuracy for class: ship is 64.4 %
Accuracy for class: truck is 67.0 %
```

---

In the above cases you can observe that changing the Running Training to Longer doesn't have much effect on the accuracies but you can see the change in the loss if the loop is done at multiple time, because the more the training done on the data set more the accurate prediction or loss in the prediction is possible.

Changing learning rate and momentum increases the accuracy, because lower the learning rate more nearest possible values are found. If you combine a higher learning rate with a big momentum, you will rush past the minimum with big steps. When the gradient keeps changing direction, momentum will smooth out the variations. Hence taking the min small learning rates and lower momentum will give more accurate values.

Increasing the number of convolution layers increases the accuracy because, adding more convolution decomposes the input image to more layers which will help us to identify the minor details in the image for the more accurate training.

“A fully connected layer multiplies the input by a weight matrix and then adds a bias vector.” With more full connected layers more hidden layers are taken into consideration, since the more hidden layers gives more accuracy the change in the number of fully connected layers will change the accuracy.

Adding more strides decreases the accuracy because, increase in the stride increases the movement in the image which will lead to more disturbances.

**So, to improve the accuracy my ideal solution is to take the lower learning rate and momentum, more loops , convolution layers and less strides.**