Problem 1: Construct a deep neural network with at least 2 convolutional and 2 fully connected layers. Do analysis on various parameters like accuracy, time taken and compare the performance with various activation functions, with and without momentum, adaptive learning rate, with and without Batch normalization.

Approach:

To start with, I had just added 2 conv layers and 3 fc layers.
To calculate the output size for convolutional layer, I had used this formula: **[(W−K+2P)/S]+1.**

- W is the input volume.
- K is the Kernel size.
- P is the padding.
- S is the stride.

Network description (2 Conv layer followed by 3 FC layers):

Conv1 layer is of shape (3, 6, 3) i.e input channel as 3 because image is of shape 32*32*3, output channel as 6 and kernel size as 3. And stride and padding takes default value 1 and 0 respectively. Followed by **Max pooling layer** with kernel size as 2*2.

Conv2 layer is of shape (6, 16, 3) i.e input channel as 6 because first layer gives output of shape 15*15*6, output channel as 16 and kernel size as 3. And stride and padding takes default value 1 and 0 respectively.

Full connected layer 1 (FC1) has input of shape (16*14*14, 1) and output of shape (520, 1). Followed by **FC2** which has input of shape (520, 1) and output of shape (84, 1). And then the last **FC3** of shape (84,1) and output of shape (10,1) (Because CIFAR-10 has 10 classes.)

To train this network various hyper parameter values are as follows:

- Number of epochs: 5
- Learning rate: 0.001
- SGD with Momentum = 0.9
- And activation function as ReLU
- Batch size =4

With this network design and hyper-parameters I had got accuracy of **65.67%.** Now, let's see if going more deep will help in increasing accuracy of the model or not.

Network description (3 Conv layer followed by 3 FC layers):

In this architecture I had added one more Conv layer. But as image is of size 32*32 so **hold on the size of image till we reach the end**, for the first layer I had done **padding = 1** (Same size output) hence for 1st layer I will get the same height and weight but increased depth.

Conv1: (input_channel=3, output_channel=6, kernel=3, stride=1, padding=1)
maxpool(2, 2)
Conv2: (input_channel=6, output_channel=16, kernel=3, stride=1, padding=0)
maxpool(2, 2)
Conv3: (input_channel=16,output_channel=16, kernel=3, stride=1, padding=0)
maxpool(2, 2)
Fc1: (input=16*6*6, output= 520)
Fc2: (input=520, output= 84)
Fc3: (input=84, output= 10)

Hyper parameter values are as follows:

- Number of epochs: 10
- Learning rate: 0.001
- SGD with Momentum= 0.9
- Activation function: ReLU
- Batch size = 4

Accuracy for this network was 69.16% and time required to train and test(at each epoch) was 10.86 minutes (Using GPU on COLAB).

As this network performed better, I tried different activation functions on it and trained it without momentum. Result obtained for it are as follows:

- With **tanh: 69.16%** acc and time taken is 10.86 mins.
- With **sigmoid: 10%** acc (at 4th epoch so stopped early).
- Without momentum: **48.49%** acc and time taken is 12 mins.

As the fully connected layers are not that much dense, hence I tried making those layers dense.

Keeping conv layers same as above, the new Fc layers are as follows:

- **Fc1:** (input=16*6*6, output= 1024)
- **Fc2:** (input=1024, output= 512)
- **Fc3:** (input=512, output= 10)

With same hyper parameters configuration, it gave an accuracy of 68.2% and took 10.79 mins.

Adaptive learning rate performance:

On the same network as described above I had tried training it with adaptive learning rate using scheduler with step size as 2, starting rate as 0.1 and gamma as 0.1 . Hence after every 2 epochs, the learning rate decreases by factor of 10. Keeping rest hyper parameters as same, the accuracy obtained was **55.32%** and time required was 12 mins.

So far the best performance (69.16%) is obtained for 2nd architecture, hence next we will do further experiments on this architecture.

Next I had tried applying Batch normalization after every conv layer and sgd without momentum. It gave accuracy of **71.2%** and took 11.26 mins.

Hyper parameter values are as follows:

- Number of epochs: 10
- Learning rate: 0.01
- SGD without Momentum
- Activation function: ReLU
- Batch size =4

With same settings as above but **with momentum =0.9** the result obtained are : **52.6% accuracy** and took 12.13 mins.

Recommended architecture:

Conv1: (input_channel=3, output_channel=6, kernel=3, stride=1, padding=1)

BatchNormalization(6), ReLU, maxpool(2, 2)

Conv2: (input_channel=6, output_channel=16, kernel=3, stride=1, padding=0)

BatchNormalization(16), ReLU, maxpool(2, 2)

Conv3: (input_channel=16,output_channel=16, kernel=3, stride=1, padding=0)

BatchNormalization(16), ReLU, maxpool(2, 2)

Fc1: (input=16*6*6, output= 520)

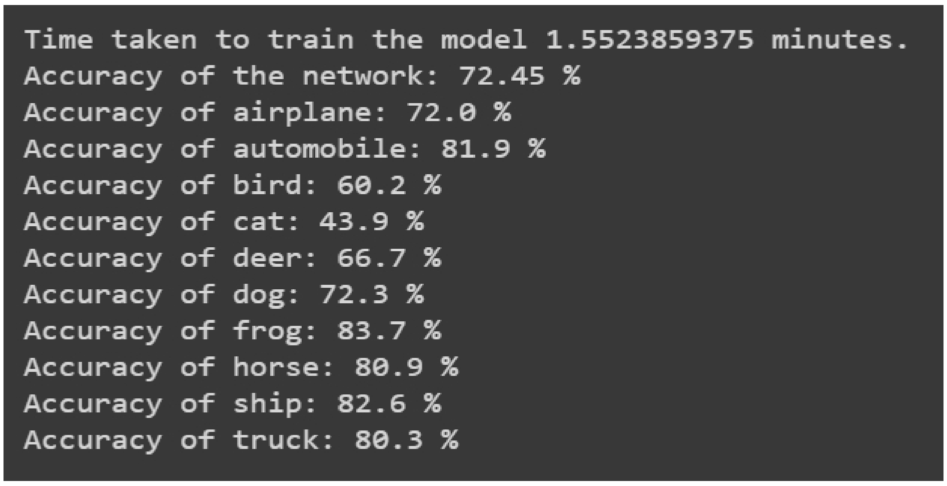
Fc2: (input=520, output= 84)

Fc3: (input=84, output= 10)

Hyper-parameters:

- Number of epochs: 10
- Learning rate: 0.01
- SGD without Momentum
- Activation function: ReLU
- **Batch size =100 (changed from 4 to 100)**

Accuracy: 72.45%



```
Time taken to train the model 1.5523859375 minutes.
Accuracy of the network: 72.45 %
Accuracy of airplane: 72.0 %
Accuracy of automobile: 81.9 %
Accuracy of bird: 60.2 %
Accuracy of cat: 43.9 %
Accuracy of deer: 66.7 %
Accuracy of dog: 72.3 %
Accuracy of frog: 83.7 %
Accuracy of horse: 80.9 %
Accuracy of ship: 82.6 %
Accuracy of truck: 80.3 %
```

As it can be seen, model is performing equally good on all the classes that is the accuracy is not only because of one class performance which is a good sign.

Problem 2: Using Alexnet as a feature extracter (extract last layer as features), use any model on top and report the classification accuracy. Fine tune using pre-trained alexnet.

Dataset:

For this problem, I have used STL10 data set.

- 10 classes: airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck.
- Images are 96x96 pixels, color.
- 500 training images, 800 test images per class.

Approach:

Step1: First of all make an object of pre trained AlexNet model.

```
new_model = models.alexnet(pretrained=True)
```

Step2: Remove last layer from pre trained AlexNet.

```
new_classifier = nn.Sequential(*list(new_model.classifier.children())[:-1])

new_model.classifier = new_classifier
```

So our new_model will not have that last Fully connected layer of shape (4096, 1000).

Step3: Create SVM class. And it's object which we will add it at the end of new_model.

```
final_model = nn.Sequential(new_model, model2)
```

This will add new layer as last layer. It will be SVM with shape of (4096, 10).

Step4: Finally train the model for 5 epochs.

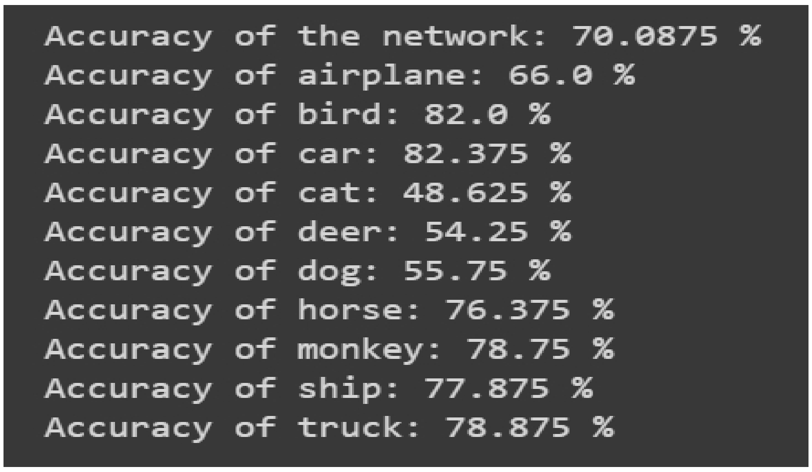
Note: As pre trained alexnet is used just as feature extractor and we are not training this pre trained model, so we have to freeze all the layers of this model.

Code to do so:

```
for param in model.parameters():

        param.requires_grad = False
```

Output:

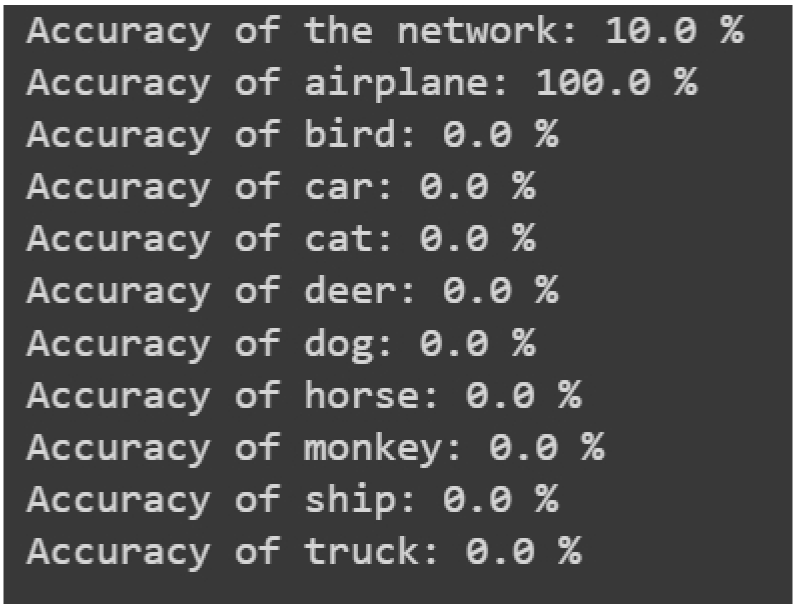


```
Accuracy of the network: 70.0875 %
Accuracy of airplane: 66.0 %
Accuracy of bird: 82.0 %
Accuracy of car: 82.375 %
Accuracy of cat: 48.625 %
Accuracy of deer: 54.25 %
Accuracy of dog: 55.75 %
Accuracy of horse: 76.375 %
Accuracy of monkey: 78.75 %
Accuracy of ship: 77.875 %
Accuracy of truck: 78.875 %
```

Fine tuning pre-trained AlexNet model on STL10 Dataset:

Fine tuning for 20 epochs, with SGD optimizer with momentum = 0.9 and learning rate as 0.01 . The result obtained are as follows,

Output:



```
Accuracy of the network: 10.0 %
Accuracy of airplane: 100.0 %
Accuracy of bird: 0.0 %
Accuracy of car: 0.0 %
Accuracy of cat: 0.0 %
Accuracy of deer: 0.0 %
Accuracy of dog: 0.0 %
Accuracy of horse: 0.0 %
Accuracy of monkey: 0.0 %
Accuracy of ship: 0.0 %
Accuracy of truck: 0.0 %
```

Accuracy of this model is just 10% and even bad thing is, it is only identifying Airplane and nothing else. This performance might be because we are just fine tuning alexnet for only 20 epochs. And alexnet weights are learned on completely different dataset so we might need to train this for 100 or 200 epochs for better tuning.