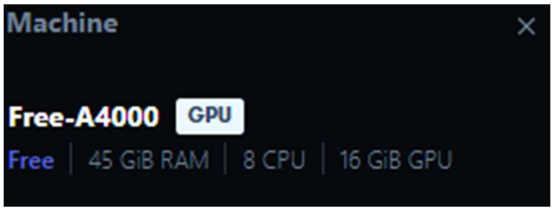
CSE 6363-007: Machine Learning Assignment 6

Name: Ananthula, Vineeth Kumar. UTA ID: 1001953922

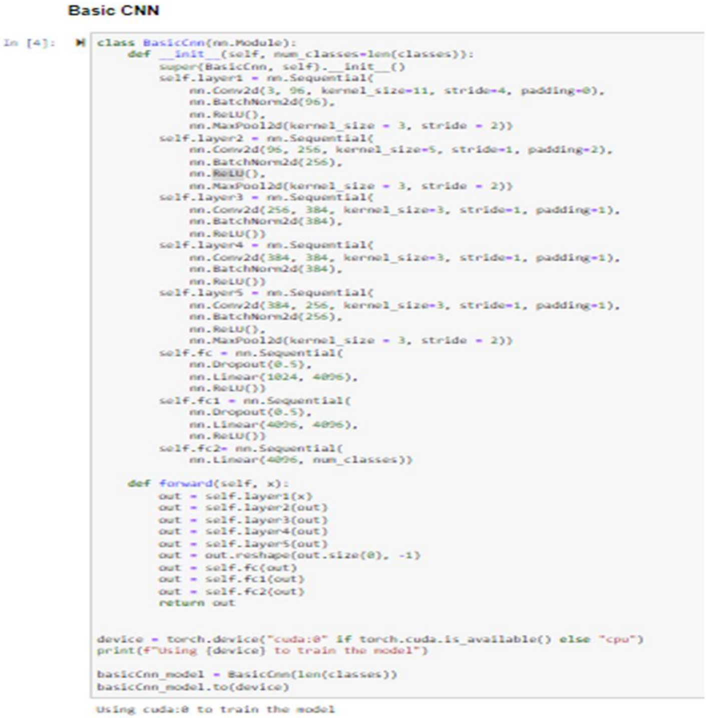
Basic CNN:

I have implemented layers in this model used Entropy loss optimizer used is stochastic gradient descent (SGD). Used ReLU for all the layers. GPU is used if its available if not cpu is used to train and test, with GPU the model trains and test very fast compared to CPU.

Machine used: https://console.paperspace.com/tnpw9t1l40/notebook/rsfblubx0pe0wjn



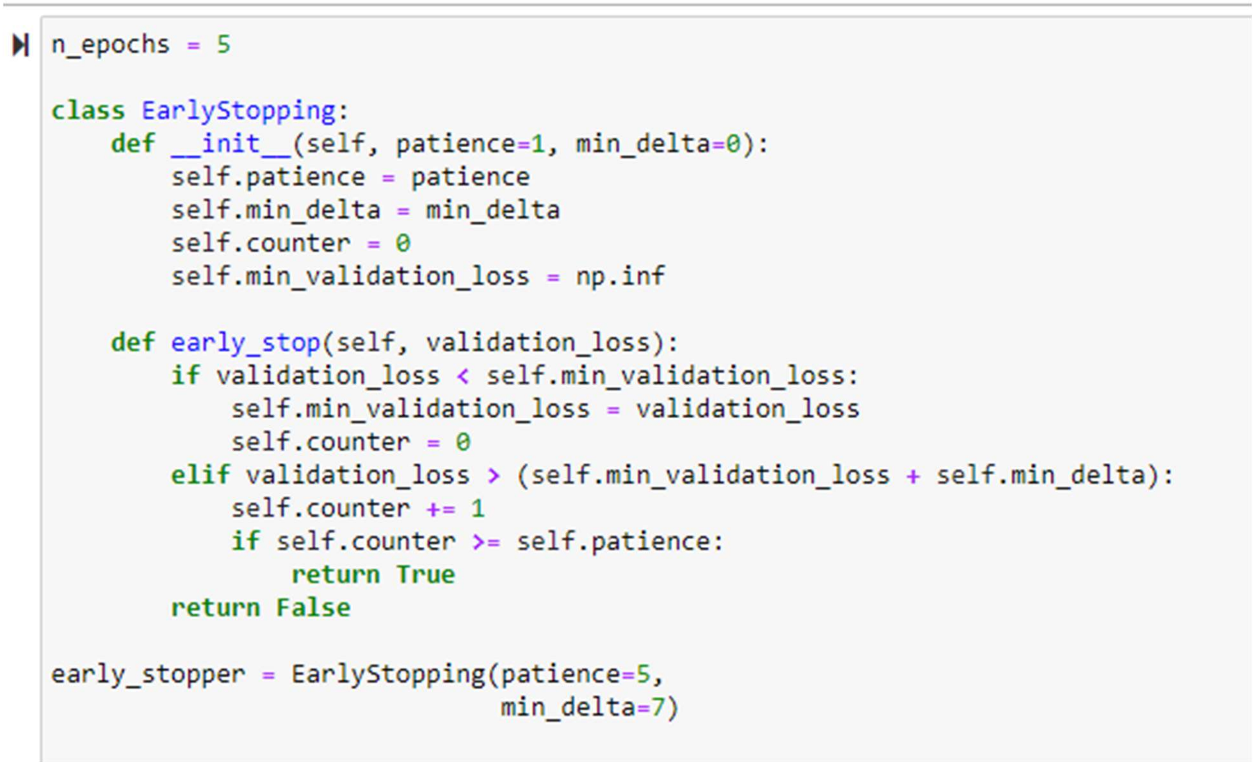
Code:



Basic CNN

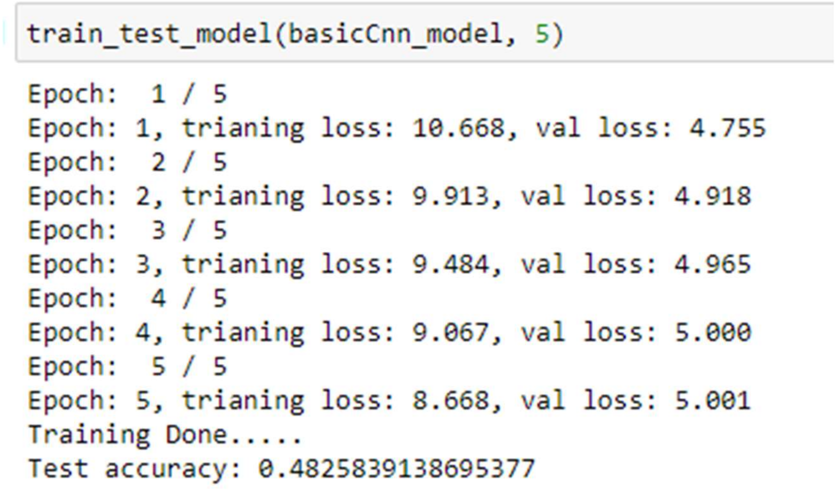
```
In [4]:  class BasicCnn(nn.Module):
             def __init__(self, num_classes=len(classes)):
                 super(BasicCnn, self).__init__()
                 self.layer1 = nn.Sequential(
                     nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=0),
                     nn.BatchNorm2d(96),
                     nn.ReLU(),
                     nn.MaxPool2d(kernel_size = 3, stride = 2))
                 self.layer2 = nn.Sequential(
                     nn.Conv2d(96, 256, kernel_size=5, stride=1, padding=2),
                     nn.BatchNorm2d(256),
                     nn.ReLU(),
                     nn.MaxPool2d(kernel_size = 3, stride = 2))
                 self.layer3 = nn.Sequential(
                     nn.Conv2d(256, 384, kernel_size=3, stride=1, padding=1),
                     nn.BatchNorm2d(384),
                     nn.ReLU())
                 self.layer4 = nn.Sequential(
                     nn.Conv2d(384, 384, kernel_size=3, stride=1, padding=1),
                     nn.BatchNorm2d(384),
                     nn.ReLU())
                 self.layer5 = nn.Sequential(
                     nn.Conv2d(384, 256, kernel_size=3, stride=1, padding=1),
                     nn.BatchNorm2d(256),
                     nn.ReLU(),
                     nn.MaxPool2d(kernel_size = 3, stride = 2))
                 self.fc = nn.Sequential(
                     nn.Dropout(0.5),
                     nn.Linear(1024, 4096),
                     nn.ReLU())
                 self.fc1 = nn.Sequential(
                     nn.Dropout(0.5),
                     nn.Linear(4096, 4096),
                     nn.ReLU())
                 self.fc2= nn.Sequential(
                     nn.Linear(4096, num_classes))

             def forward(self, x):
                 out = self.layer1(x)
                 out = self.layer2(out)
                 out = self.layer3(out)
                 out = self.layer4(out)
                 out = self.layer5(out)
                 out = out.reshape(out.size(0), -1)
                 out = self.fc(out)
                 out = self.fc1(out)
                 out = self.fc2(out)
                 return out


         device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
         print(f"Using {device} to train the model")

         basicCnn_model = BasicCnn(len(classes))
         basicCnn_model.to(device)

         Using cuda:0 to train the model
```

Early stopper:



```
n_epochs = 5

class EarlyStopping:
    def __init__(self, patience=1, min_delta=0):
        self.patience = patience
        self.min_delta = min_delta
        self.counter = 0
        self.min_validation_loss = np.inf

    def early_stop(self, validation_loss):
        if validation_loss < self.min_validation_loss:
            self.min_validation_loss = validation_loss
            self.counter = 0
        elif validation_loss > (self.min_validation_loss + self.min_delta):
            self.counter += 1
            if self.counter >= self.patience:
                return True
        return False

early_stopper = EarlyStopping(patience=5,
                              min_delta=7)
```

Accuracy:



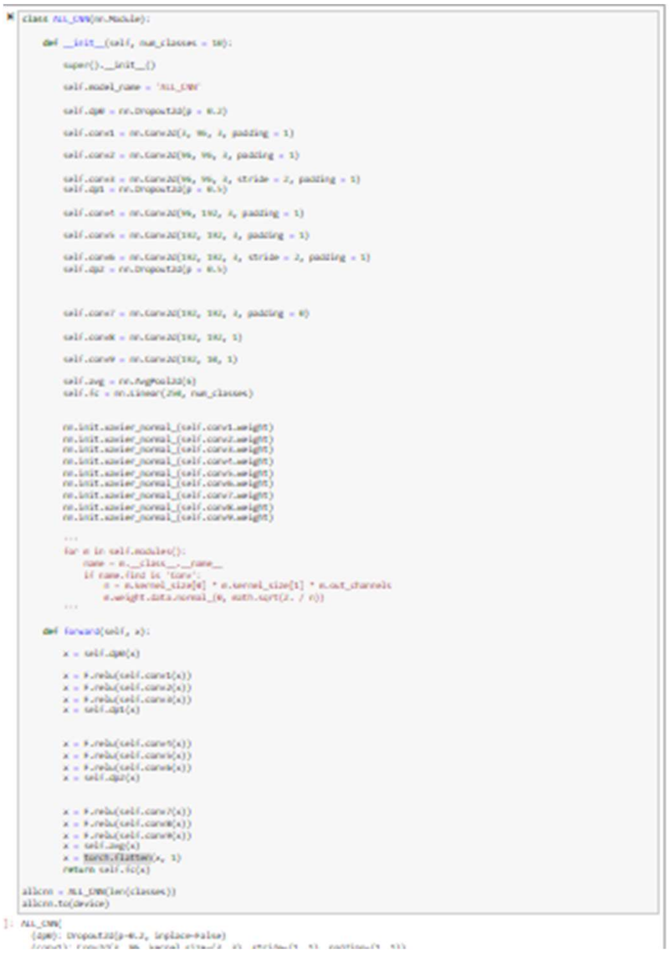
```
train_test_model(basicCnn_model, 5)
```

```
Epoch:  1 / 5
Epoch: 1, trianing loss: 10.668, val loss: 4.755
Epoch:  2 / 5
Epoch: 2, trianing loss: 9.913, val loss: 4.918
Epoch:  3 / 5
Epoch: 3, trianing loss: 9.484, val loss: 4.965
Epoch:  4 / 5
Epoch: 4, trianing loss: 9.067, val loss: 5.000
Epoch:  5 / 5
Epoch: 5, trianing loss: 8.668, val loss: 5.001
Training Done.....
Test accuracy: 0.4825839138695377
```

All Convolutional Net:

This model has been implemented with conventional layers followed by pool layer. Used Relu for all conv layers. torch.flatten provides the output by reshaping it into a one-dimensional tensor.

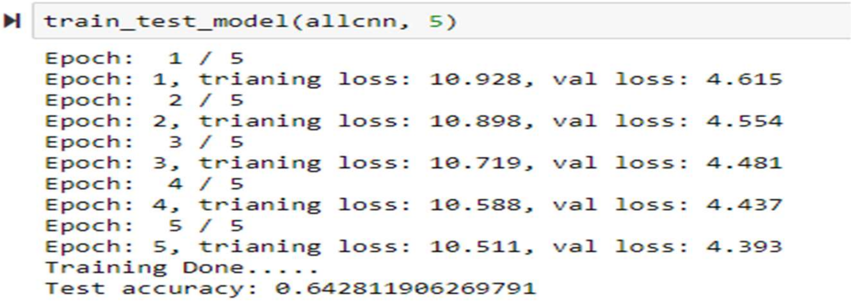
Code:



```
class All_CNN(nn.Module):

    def __init__(self, num_classes = 10):

        super().__init__()

        self.model_name = 'All_CNN'

        self.dp0 = nn.Dropout2d(p = 0.2)

        self.conv1 = nn.Conv2d(3, 96, 3, padding = 1)

        self.conv2 = nn.Conv2d(96, 96, 3, padding = 1)

        self.conv3 = nn.Conv2d(96, 96, 3, stride = 2, padding = 1)
        self.dp1 = nn.Dropout2d(p = 0.5)

        self.conv4 = nn.Conv2d(96, 192, 3, padding = 1)

        self.conv5 = nn.Conv2d(192, 192, 3, padding = 1)

        self.conv6 = nn.Conv2d(192, 192, 3, stride = 2, padding = 1)
        self.dp2 = nn.Dropout2d(p = 0.5)


        self.conv7 = nn.Conv2d(192, 192, 3, padding = 0)

        self.conv8 = nn.Conv2d(192, 192, 1)

        self.conv9 = nn.Conv2d(192, 10, 1)

        self.avg = nn.AvgPool2d(6)
        self.fc = nn.Linear(250, num_classes)


        nn.init.xavier_normal_(self.conv1.weight)
        nn.init.xavier_normal_(self.conv2.weight)
        nn.init.xavier_normal_(self.conv3.weight)
        nn.init.xavier_normal_(self.conv4.weight)
        nn.init.xavier_normal_(self.conv5.weight)
        nn.init.xavier_normal_(self.conv6.weight)
        nn.init.xavier_normal_(self.conv7.weight)
        nn.init.xavier_normal_(self.conv8.weight)
        nn.init.xavier_normal_(self.conv9.weight)

        '''
        for m in self.modules():
            name = m.__class__.__name__
            if name.find is 'Conv':
                n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                m.weight.data.normal_(0, math.sqrt(2. / n))
        '''

    def forward(self, x):

        x = self.dp0(x)

        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = self.dp1(x)


        x = F.relu(self.conv4(x))
        x = F.relu(self.conv5(x))
        x = F.relu(self.conv6(x))
        x = self.dp2(x)


        x = F.relu(self.conv7(x))
        x = F.relu(self.conv8(x))
        x = F.relu(self.conv9(x))
        x = self.avg(x)
        x = torch.flatten(x, 1)
        return self.fc(x)

allcnn = All_CNN(len(classes))
allcnn.to(device)
```

```
]: All_CNN(
     (dp0): Dropout2d(p=0.2, inplace=False)
     (conv1): Conv2d(3, 96, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

Accuracy:



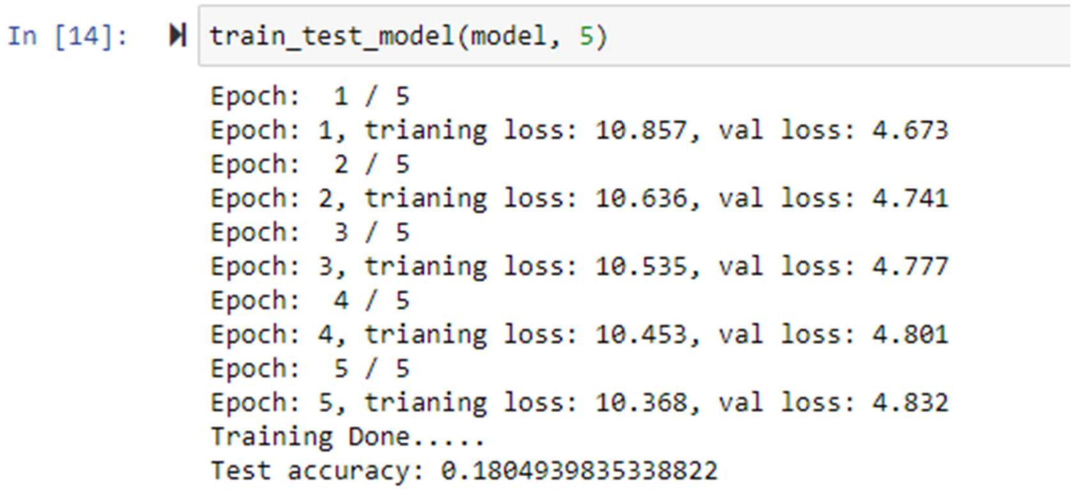
```
train_test_model(allcnn, 5)
```

```
Epoch:  1 / 5
Epoch: 1, trianing loss: 10.928, val loss: 4.615
Epoch:  2 / 5
Epoch: 2, trianing loss: 10.898, val loss: 4.554
Epoch:  3 / 5
Epoch: 3, trianing loss: 10.719, val loss: 4.481
Epoch:  4 / 5
Epoch: 4, trianing loss: 10.588, val loss: 4.437
Epoch:  5 / 5
Epoch: 5, trianing loss: 10.511, val loss: 4.393
Training Done.....
Test accuracy: 0.642811906269791
```

Regularization:

Used BasicCNN model. For the augmentation we used random cropping, random vertical flipping, and random horizontal flipping.

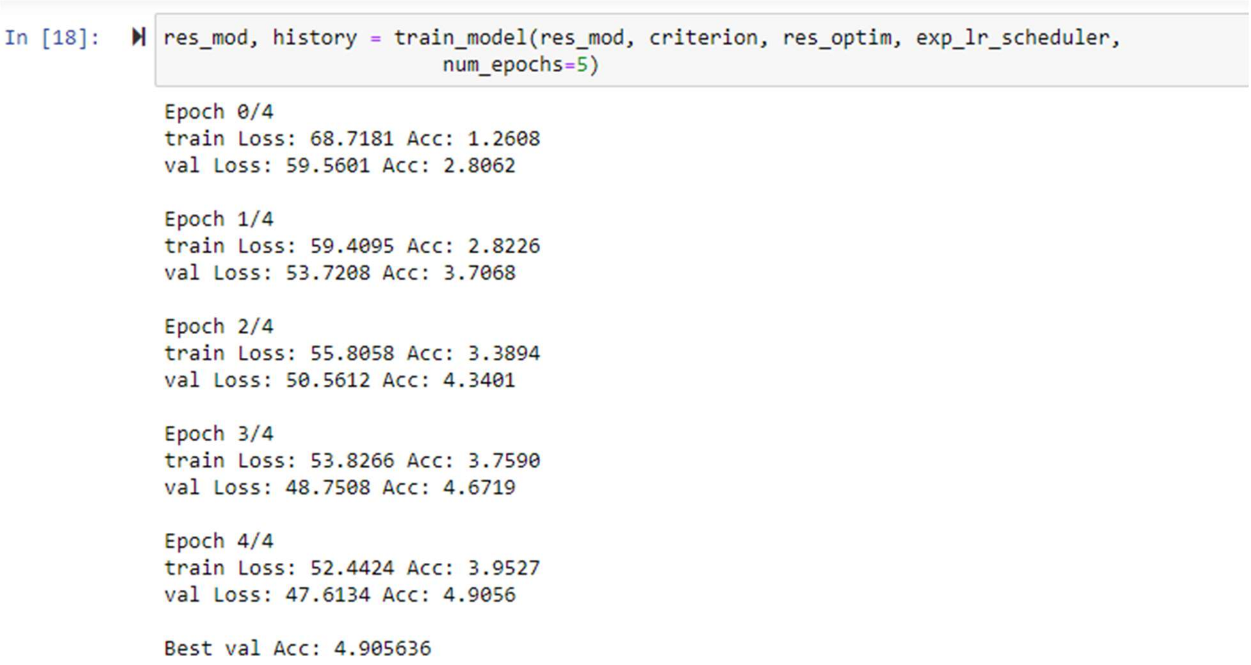
Accuracy:



```
In [14]:    train_test_model(model, 5)

            Epoch:  1 / 5
            Epoch: 1, trianing loss: 10.857, val loss: 4.673
            Epoch:  2 / 5
            Epoch: 2, trianing loss: 10.636, val loss: 4.741
            Epoch:  3 / 5
            Epoch: 3, trianing loss: 10.535, val loss: 4.777
            Epoch:  4 / 5
            Epoch: 4, trianing loss: 10.453, val loss: 4.801
            Epoch:  5 / 5
            Epoch: 5, trianing loss: 10.368, val loss: 4.832
            Training Done.....
            Test accuracy: 0.1804939835338822
```

Transfer Learning:

Used ResNet architecture for transfer learning changed the last layer's output shape to 101, then trained the last layer of the network redefined the training function to train the model.



```
In [18]:    res_mod, history = train_model(res_mod, criterion, res_optim, exp_lr_scheduler,
                                           num_epochs=5)

            Epoch 0/4
            train Loss: 68.7181 Acc: 1.2608
            val Loss: 59.5601 Acc: 2.8062

            Epoch 1/4
            train Loss: 59.4095 Acc: 2.8226
            val Loss: 53.7208 Acc: 3.7068

            Epoch 2/4
            train Loss: 55.8058 Acc: 3.3894
            val Loss: 50.5612 Acc: 4.3401

            Epoch 3/4
            train Loss: 53.8266 Acc: 3.7590
            val Loss: 48.7508 Acc: 4.6719

            Epoch 4/4
            train Loss: 52.4424 Acc: 3.9527
            val Loss: 47.6134 Acc: 4.9056

            Best val Acc: 4.905636
```