Architecture - more layer

Model_Name/Parameter	Test loss	Test acc	Parameter
Model1(2_layers)	0.6854330684457507	tensor(0.5692)	batch_size : 16 Dropout : 0.3 after fully connected layer Activation : ReLu BatchNormalization
Model2(3_layers)	0.6859755026442664,	tensor(0.5692)	batch_size : 16 Dropout : 0.3 after fully connected layer Activation : ReLu BatchNormalization
Model3(4_layers)	0.6844106550727572	tensor(0.5692)	batch_size : 16 Dropout : 0.3 after fully connected layer Activation : ReLu BatchNormalization
Model4(5_layers)	0.6834706259625298	tensor(0.5692)	batch_size : 16 Dropout : 0.3 after fully connected layer Activation : ReLu BatchNormalization

batch_size : 16
Dropout : 0.3 after fully connected layer
Activation : ReLu
BatchNormalization

Conclusion1 : more layer , lower accuracy

Dropout

Model_Name/Parameter	Test loss	Test acc	Dropout
Model2_0(3_layers)	0.6859755026442664,	tensor(0.5692)	0.3
Model2_1(3_layers)	0.6859409255640847	tensor(0.5692)	0.6
Model2_2(3_layers**)**	0.685993960925510	tensor(0.5692)	0.7

	9		
Model2_3(3_layers)	0.6857426975454602	tensor(0.5692)	0.8

Basic model1 : Model1 2layers

Optimizer

Model_Name/Parameter	Test loss	Test acc	Optimizer
Model2_0_1(3_layers)	0.6859755026442664,	tensor(0.5692)	Adam
Model2_0_2(3_layers)	0.7247943154403141	tensor(0.5117)	SGD
Model2_0_3(3_layers)	0.7009437999555043	tensor(0.4308)	RMSProp

Test loss; acc (0.6854330684457507, tensor(0.5692))

```
class Net2(nn.Module):
    def __init__(self):
        super().__init__()
        self.modle = nn.Sequential(
            nn.ZeroPad2d(1),
            nn.Conv2d(3,16,kernel_size = 3),
            nn.BatchNorm2d(16),
            nn.ReLU(inplace = True),
            #MaxPool2d(kernel_size = 2 , stride =2),
            nn.Conv2d(16,32,kernel_size = 2),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace = True),
            nn.Dropout2d(0.3),
            #nn.MaxPool2d(kernel_size = 2, stride = 2),
        )
        self.fc = nn.Sequential(nn.Linear(32*9*2,2))

        self.history={'train_accuracy':[],'train_loss':[],'validation_accuracy':[],'validation_loss':[]}
    # Defining the forward pass
    def forward(self, x):
        #print('sizex1: ',x.size())
        x = self.modle(x)
        #print('sizex2: ',x.size())
        x = x.view(x.size(0), -1)
        #print('sizex3: ',x.size())
        x = self.fc(x)
        #print('sizex4: ',x.size())
        return x
```

Basic model2 : Model2 3layers

Test loss; acc 0.6859755026442664,0.5692

```
class Net3(nn.Module):
    def __init__(self):
        super().__init__()
        self.modle = nn.Sequential(

            nn.ZeroPad2d(1),
            nn.Conv2d(3,16,kernel_size = 3),
            nn.BatchNorm2d(16),
            nn.ReLU(inplace = True),
            #MaxPool2d(kernel_size = 2 , stride =2),

            nn.Conv2d(16,32,kernel_size = 2),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace = True),

            nn.Conv2d(32,64,kernel_size = 2),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace = True),
            nn.Dropout2d(0.3),
            #nn.MaxPool2d(kernel_size = 2, stride = 2),
        )
        self.fc = nn.Sequential(nn.Linear(64*8*1,2))
        self.history={'train_accuracy':[],'train_loss':[],'validation_accuracy':[],'validation_loss':[]}
    # Defining the forward pass
    def forward(self, x):
        #print('sizex1: ',x.size())
        x = self.modle(x)
        #print('sizex2: ',x.size())
        x = x.view(x.size(0), -1)
        #print('sizex3: ',x.size())
        x = self.fc(x)
        #print('sizex4: ',x.size())
        return x
```

Basic model3 : Model3 4layers

```
#PyTorch model 3

class Net3(nn.Module):
    def __init__(self):
        super().__init__()
        self.modle = nn.Sequential(

            nn.ZeroPad2d(4),
            nn.Conv2d(3,16,kernel_size = 3),
            nn.BatchNorm2d(16),
            nn.ReLU(inplace = True),
            #MaxPool2d(kernel_size = 2 , stride =2),

            nn.Conv2d(16,32,kernel_size = 2),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace = True),

            nn.Conv2d(32,64,kernel_size = 2),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace = True),

            nn.Conv2d(64,128,kernel_size = 2),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace = True),
            nn.Dropout2d(0.3),
            #nn.MaxPool2d(kernel_size = 2, stride = 2),
        )
        self.fc = nn.Sequential(nn.Linear(128*13*6,2))
        self.history={'train_accuracy':[],'train_loss':[],'validation_accuracy':[],'validation_loss':[]}
    # Defining the forward pass
    def forward(self, x):
        #print('sizex1: ',x.size())
        x = self.modle(x)
        #print('sizex2: ',x.size())
        x = x.view(x.size(0), -1)
        #print('sizex3: ',x.size())
        x = self.fc(x)
        #print('sizex4: ',x.size())
        return x
```

Basic model4 : Model4 5layers

```
#PyTorch model 4
```

```
class Net4(nn.Module):
    def __init__(self):
        super().__init__()
        self.modle = nn.Sequential(

            nn.ZeroPad2d(4),
            nn.Conv2d(3,16,kernel_size = 3),
            nn.BatchNorm2d(16),
            nn.ReLU(inplace = True),
            #MaxPool2d(kernel_size = 2 , stride =2),

            nn.Conv2d(16,32,kernel_size = 2),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace = True),

            nn.Conv2d(32,64,kernel_size = 2),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace = True),

            nn.Conv2d(64,128,kernel_size = 2),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace = True),

            nn.Conv2d(128,256,kernel_size = 2),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace = True),

            nn.Dropout2d(0.3),
            #nn.MaxPool2d(kernel_size = 2, stride = 2),
        )
        self.fc = nn.Sequential(nn.Linear(256*12*5,2))
        self.history={'train_accuracy':[],'train_loss':[],'validation_accuracy':[],'validation_loss':[]}

    # Defining the forward pass
    def forward(self, x):
        #print('sizex1: ',x.size())
        x = self.modle(x)
        #print('sizex2: ',x.size())
        x = x.view(x.size(0), -1)
        #print('sizex3: ',x.size())
        x = self.fc(x)
        #print('sizex4: ',x.size())
        return x
```