ECE4179 Assignment 3

Zhiyue Li

28280016

Zlii0010@student.monash.edu

Q1. Shallow CNN

1.

```
batch_size = 100
n_workers = multiprocessing.cpu_count()
n_workers = 0
trainloader = torch.utils.data.DataLoader(train_set, batch_size=batch_size,
                                          shuffle=True, num_workers=n_workers)
testloader = torch.utils.data.DataLoader(test_set, batch_size=batch_size,
                                          shuffle=True, num_workers=n_workers)
valloader = torch.utils.data.DataLoader(val_set, batch_size=batch_size,
                                          shuffle=True, num_workers=n_workers)
```

PS: Due to the multiprocessing issue, then n_workers is set as 0.

:

```
# Set device to GPU_index if GPU is available
GPU_index = 0
device = torch.device(GPU_index if torch.cuda.is_available() else 'cpu')

n_epochs = 20
learning_rate = 1e-3
training_loss_logger = []
training_acc_logger =[]

validation_loss_logger = []
validation_acc_logger = []

test_loss_logger = []
test_acc_logger = []

def calculate_accuracy(fx, y):
    preds = fx.max(1, keepdim=True)[1]
    correct = preds.eq(y.view_as(preds)).sum()
    acc = correct.float()/preds.shape[0]
    return acc
```

```python
class Model(nn.Module):
    def __init__(self):

        super(Model, self).__init__()

        self.conv1 = nn.Conv2d(3,96, kernel_size=7, stride=2, padding=0)
        self.conv2 = nn.Conv2d(96, 64, kernel_size=5, stride=2, padding=0)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=0)

        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=3, padding=0)

        self.linear1 = nn.Linear(1152, 128)
        self.linear2 = nn.Linear(128, 10)



    def forward(self, x):

        out1 = F.relu(self.conv1(x))
        out2 = F.relu(self.conv2(out1))
        out3 = F.relu(self.conv3(out2))
        out4 = self.maxpool(out3)
        out4 = out4.view(out4.shape[0],-1)
        out5 = F.relu(self.linear1(out4))
        out6 = self.linear2(out5)

        return out6
```

```python
#This function should perform a single training epoch using our training data
def train(net, device, loader, optimizer, Loss_fun, loss_logger,acc_logger):

    #initialise counters
    epoch_loss = 0
    epoch_acc = 0

    #Set Network in train mode
    net.train()

    for i, (x, y) in enumerate(loader):

        #Load images and labels to device
        x = x.to(device) # x is the image
        y = y.type(torch.LongTensor).to(device) # y is the corresponding label

        #Forward pass of image through network and get output
        fx = net(x)

        #Calculate loss using loss function
        loss = Loss_fun(fx, y)

        #calculate the accuracy
        acc = calculate_accuracy(fx, y)

        #Zero Gradents
        optimizer.zero_grad()
        #Backpropagate Gradents
        loss.backward()
        #Do a single optimization step
        optimizer.step()

        #create the cumulative sum of the loss and acc
        epoch_loss += loss.item()
        epoch_acc += acc.item()
        #log the loss for plotting
        loss_logger.append(loss.item())
        acc_logger.append(loss.item())


        #clear_output is a handy function from the IPython.display module
        #it simply clears the output of the running cell

        clear_output(True)
        print("TRAINING: | Itteration [%d/%d] | Loss %.2f |" %(i+1 ,len(loader) , loss.item()))

    #return the avaerage loss and acc from the epoch as well as the logger array
    return epoch_loss / len(loader), epoch_acc / len(loader), loss_logger,acc_logger
```

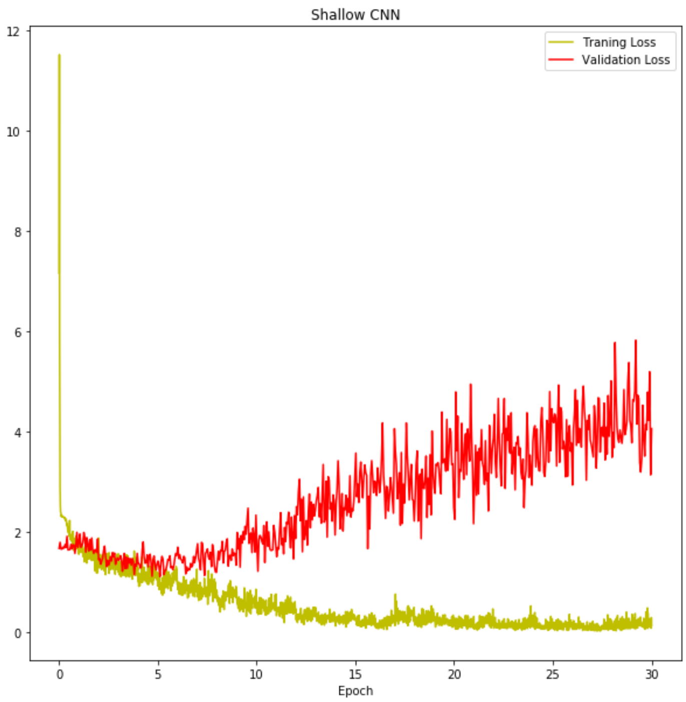
```python
#This function should perform a single evaluation epoch and will be passed our validation or evaluation/test data
#it WILL NOT be used to train out model
def evaluate(net, device, loader, Loss_fun, loss_logger = None,acc_logger= None):

    epoch_loss = 0
    epoch_acc = 0

    #Set network in evaluation mode
    #Layers like Dropout will be disabled
    #Layers like Batchnorm will stop calculating running mean and standard deviation
    #and use current stored values
    net.eval()

    with torch.no_grad():
        for i, (x, y) in enumerate(loader):

            #Load images and labels to device
            x = x.to(device)
            y = y.type(torch.LongTensor).to(device) # y is the corresponding label

            #Forward pass of image through network
            fx = net(x)

            #Calculate loss using loss function
            loss = Loss_fun(fx, y)

            #calculate the accuracy
            acc = calculate_accuracy(fx, y)

            #log the cumulative sum of the loss and acc
            epoch_loss += loss.item()
            epoch_acc += acc.item()

            #log the loss for plotting if we passed a logger to the function
            if not (loss_logger is None):
                loss_logger.append(loss.item())

            if not (acc_logger is None):
                acc_logger.append(acc.item())

            #clear_output(True)
            print("EVALUATION: | Itteration [%d/%d] | Loss %.2f | Accuracy %.2f%% |" %(i+1 ,len(loader), loss.item(), 100*(epoch_

    #return the avaerage loss and acc from the epoch as well as the logger array
    return epoch_loss / len(loader), epoch_acc / len(loader), loss_logger,acc_logger

# Transfer for model to GPU
net = Model().to(device)
# Use the Adam optimiser to update the weights of the model
optimizer = optim.Adam(net.parameters(), lr = learning_rate)
#Cross entropy -- softmax over the class and negative log likelihood loss
loss_fn = nn.CrossEntropyLoss()
```

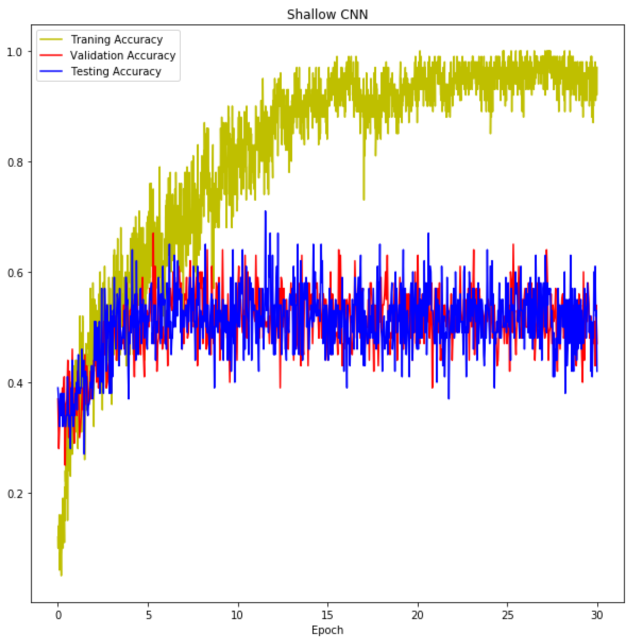
```python
optim_valid_acc = 0
for epoch in range(n_epochs):
    print(epoch)
    #call the training function and pass training dataloader etc
    train_loss, train_acc, training_loss_logger,training_acc_logger= train(net, device, trainloader, optimizer, loss_fn, training

    #call the evaluate function and pass validation dataloader etc
    valid_loss, valid_acc, validation_loss_logger,validation_acc_logger = evaluate(net, device, valloader, loss_fn, validation_lo

    #If this model has the highest performace on the validation set
    #then save a checkpoint
    #{} define a dictionary, each entry of the dictionary is indexed with a string
    if (valid_acc > optim_valid_acc):
        print("Saving Model")
        best_model={
            'epoch':                epoch,
            'model_state_dict':     net.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'train_acc':            train_acc,
            'valid_acc':            valid_acc,
        }

    print(f'| Epoch: {epoch+1:02} | Train Loss: {train_loss:.3f} | Train Acc: {train_acc*100:05.2f}% | Val. Loss: {valid_loss:.3f
```

The multiple snipping photos above are the code structure applied for Question 1.

In this question, the learning rate was defined as 1e-3 and number of epochs are 60. Adam optimizer is applied.

The size of feature map after the convolution and pooling is a resolution of 3 X 3 with 128 channels.






```
Saving Model
| Epoch: 30 | Train Loss: 0.165 | Train Acc: 94.72% | Val. Loss: 4.241 | Val. Acc: 50.45% |
```

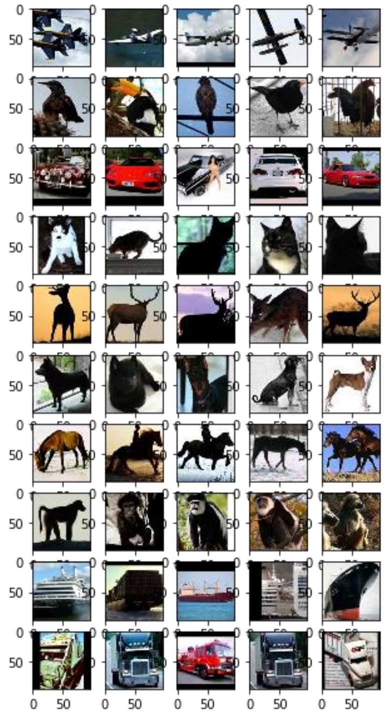
Conclusion: From the left plot above, we can tell that the loss of validation test has diverged(increased) when the loss of training has started to converge(decreased). It indicates the overfitting in this case. In addition, for the right plot above, we can tell that the accuracy of the model on validation and testing sets have reached around 50%, and their peaks reached about 56%.

2. The images below are the codes applied for question 2, 3 and 4.

```
## Part IV, might need double check
## Test the matrix one by one
dataloader = torch.utils.data.DataLoader(train_set, batch_size=batch_size,
                                         shuffle=True, num_workers=n_workers)

dev = torch.device('cpu')

max_array =[[],[],[],[],[],[],[],[],[],[]]
max_image_array=[[],[],[],[],[],[],[],[],[],[]]
rows = len(max_array)
columns = len(max_array[0])
min_array=np.copy(max_array)
min_image_array=np.copy(max_image_array)
confusion_matrix = [[0 for col in range(columns)] for row in range(rows)]
```

```python
for i, (x, y) in enumerate(dataloader):
        image = x.to(device)
        output = net(image)
        pred = torch.argmax(output,dim=1,keepdim=True)
        ## Checking whether the classification == labels
        ## Indexed images array by the boolean and found the maximum score array
        if (y.type(torch.LongTensor)==pred.to(dev)):
          p=pred.to(dev)[0][0].numpy()
          result=output[0][pred].to(dev)[0][0].detach().numpy()
          max_array[p].append(result)
          max_image_array[p].append(x)

          output=y.numpy()[0]
          confusion_mat[output][p]+=1

        else:

          p=pred.to(dev)[0][0].numpy()
          result=output[0][pred].to(dev)[0][0].detach().numpy()
          min_arr[p].append(result)
          min_image_array[p].append(x)

          result=y.numpy()[0]
          confusion_mat[result][p]+=1

print("Validation Set")
print(np.matrix(confusion_matrix))
```

```python
print("Validation Set")
print(np.matrix(confusion_matrix))

fig = plt.figure(figsize=(10,10))
#plt.xlabel("Initial randomized weights")
for index in range(rows):
  for j in range(5):
    index_matrix=np.where(max_arr[index] == np.amax(max_arr[index]))
    ## Updating on the index for each row
    index_matrix=index_matrix[0][0]
    sub = fig.add_subplot(10,5,i*5+j+1)
    img=max_image_array[i][index_matrix].numpy()
    ## img.shape
    ## Updating
    img=img[0,:,:,:]
    img=np.moveaxis(img,0,2)
    sub.imshow(img/255.0)
    del max_arr[index][index_matrix]
    del max_img_arr[index][index_matrix]

fig = plt.figure(figsize=(10,10))
for index in range(rows):
  for j in range(5):
    index_matrix=np.where(min_array[index] == np.amax(min_array[index]))
    ## Updating on the index for each row
    index_matrix=index_matrix[0][0]
    sub = fig.add_subplot(10,5,i*5+j+1)
    img=min_image_array[i][index_matrix].numpy()
    ## img.shape
    ## Updating
    img=img[0,:,:,:]
    img=np.moveaxis(img,0,2)
    sub.imshow(img/255.0)
    del min_array[index][index_matrix]
    del min_image_array[index][index_matrix]
```



3.

4. Confusion matrix

Validation Set

```
[[137  14   8   0   2   2   5   3  18   8]
 [ 13  73   2  24   9  26   8  23   4   5]
 [  4   7 144   4   1   1   5   2   5  31]
 [  5  20   2  68  19  36   8  37   1   4]
 [  2  14   3  34  94  23  21  14   4   0]
 [  1  22   3  33  17  55  25  43   0   2]
 [  1   6   2  16  21  32 105  13   1   4]
 [  2  16   2  20   5  27  21 106   2   1]
 [ 12   5  13   3   3   2   2   1 141  20]
 [ 10   4  30   6   2   2   4   3  12 124]]
```

Testing Set

```
[[210  16  18   2   6   5   5   0  23  18]
 [ 23 152   8  34  20  21  15  33   3   4]
 [ 11   5 222   1   0   2   3   1   9  42]
 [  7  19   5 109  39  42  10  56   6   7]
 [  3  15   1  38 146  32  25  24   5   2]
 [  3  30   6  36  26  99  40  56   1   2]
 [  0  12   2  15  21  28 193  23   0   5]
 [  3  27   4  39  19  34  30 139   1   2]
 [ 26   3  15   5   2   2   1   1 217  26]
 [ 16   8  54   6   3   6   6   0  25 179]]
```

Training Set

```
[[800   0   0   0   0   0   0   0   0   0]
 [  0 800   0   0   0   0   0   0   0   0]
 [  0   0 800   0   0   0   0   0   0   0]
 [  0   0   0 800   0   0   0   0   0   0]
 [  0   0   0   0 800   0   0   0   0   0]
 [  0   0   0   0   0 800   0   0   0   0]
 [  0   0   0   0   0   0 800   0   0   0]
 [  0   0   0   0   0   0   0 800   0   0]
 [  0   0   0   0   0   0   0   0 800   0]
 [  0   0   0   0   0   0   0   0   0 800]]
```

Conclusion:

Due to the overfitting characteristics on the model after training data, there will not misclassified images existing. Moreover, if we check the validation set and testing set, they have similar patterns. For example, in the images set, the planes are most mistaken for ships and the ships are most commonly (likely) mistaken for the trucks. And in the meanwhile, the trucks are mistaken for vehicles, cars. The classes that are mentioned before are also classified correctly, there is more of a spread in the middle of some classes, which contains animals. The dog class seems to be the most spread out while the mistakes mainly concentrate on some animals' class like cats, horses.

Q2. Deep CNN.

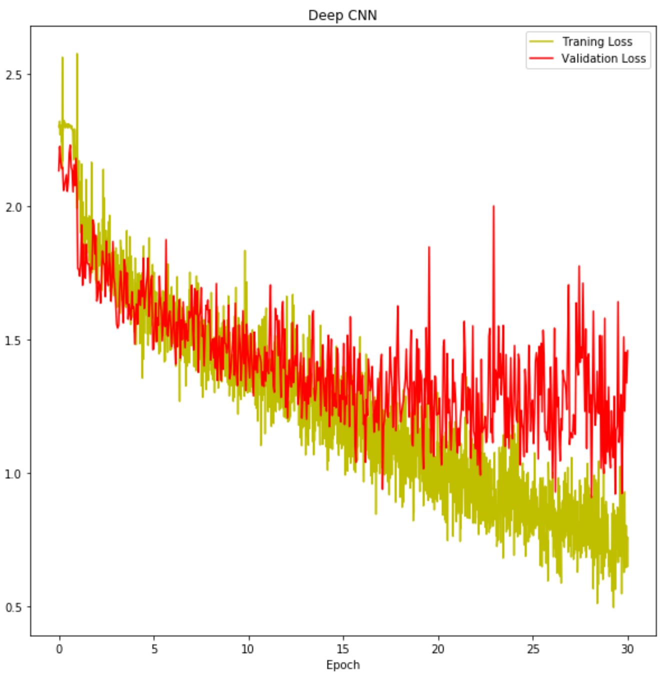
1.

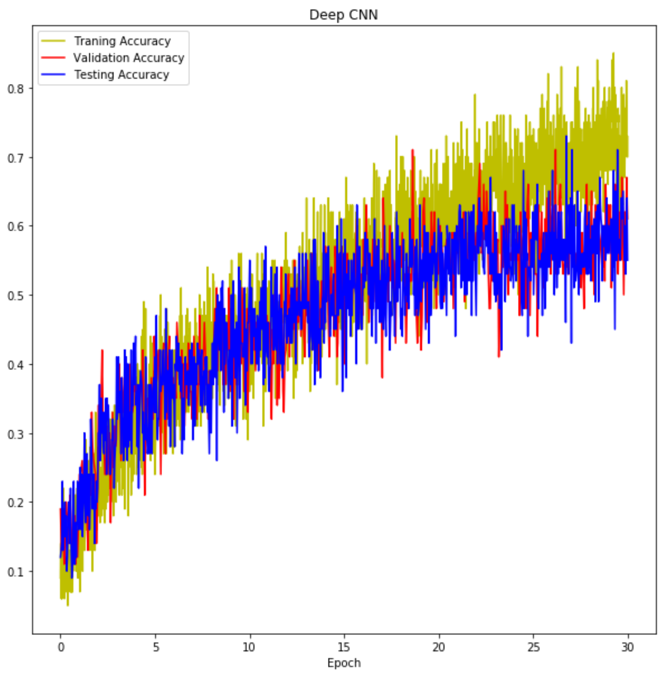
```
class Model(nn.Module):
    def __init__(self):

        super(Model, self).__init__()

        self.convblk1 = nn.Sequential (nn.Conv2d(3, 32, kernel_size=3, padding=1, stride=2),
                                       nn.ReLU(),
                                       nn.Conv2d(32, 32, kernel_size=1, padding=0, stride=1),
                                       nn.ReLU(),
                                       nn.Conv2d(32, 32, kernel_size=3, padding=1, stride=1),
                                       nn.ReLU(),

                                      )

        self.convblk2 = nn.Sequential (nn.Conv2d(32, 64, kernel_size=3, padding=1, stride=2),
                                       nn.ReLU(),
                                       nn.Conv2d(64, 64, kernel_size=1, padding=0, stride=1),
                                       nn.ReLU(),
                                       nn.Conv2d(64, 64, kernel_size=3, padding=1, stride=1),
                                       nn.ReLU(),

                                      )

        self.convblk3 = nn.Sequential (nn.Conv2d(64, 128, kernel_size=3, padding=1, stride=2),
                                       nn.ReLU(),
                                       nn.Conv2d(128, 128, kernel_size=1, padding=0, stride=1),
                                       nn.ReLU(),
                                       nn.Conv2d(128, 128, kernel_size=3, padding=1, stride=1),
                                       nn.ReLU(),

                                      )

        self.convblk4 = nn.Sequential (nn.Conv2d(128, 192, kernel_size=3, padding=1, stride=2),
                                       nn.ReLU(),
                                       nn.Conv2d(192, 192, kernel_size=1, padding=0, stride=1),
                                       nn.ReLU(),
                                       nn.Conv2d(192, 192, kernel_size=3, padding=1, stride=1),
                                       nn.ReLU(),

                                      )

        self.avgpool = nn.AvgPool2d(6)

        self.linear1 = nn.Linear(192, 10)


    def forward(self, x):

        out1 = self.convblk1(x)
        out2 = self.convblk2(out1)
        out3 = self.convblk3(out2)
        out4 = self.convblk4(out3)
        out5 = self.avgpool(out4)
        out5 = out5.view(out5.shape[0],-1)
        out6 = self.linear1(out5)

        return out6
```

```python
        return data


#This function should perform a single training epoch using our training data
def train(net, device, loader, optimizer, Loss_fun, loss_logger,acc_logger):

    #initialise counters
    epoch_loss = 0
    epoch_acc = 0

    #Set Network in train mode
    net.train()

    for i, (x, y) in enumerate(loader):

        #Load images and labels to device
        x = x.to(device) # x is the image
        y = y.type(torch.LongTensor).to(device) # y is the corresponding label

        #Forward pass of image through network and get output
        fx = net(x)

        #Calculate loss using loss function
        loss = Loss_fun(fx, y)

        #calculate the accuracy
        acc = calculate_accuracy(fx, y)

        #Zero Gradents
        optimizer.zero_grad()
        #Backpropagate Gradents
        loss.backward()
        #Do a single optimization step
        optimizer.step()

        #create the cumulative sum of the loss and acc
        epoch_loss += loss.item()
        epoch_acc += acc.item()
        #log the loss for plotting
        loss_logger.append(loss.item())
        acc_logger.append(acc.item())


        #clear_output is a handy function from the IPython.display module
        #it simply clears the output of the running cell

        #clear_output(True)
        print("TRAINING: | Itteration [%d/%d] | Loss %.2f |" %(i+1 ,len(loader) , loss.item()))

    #return the avaerage loss and acc from the epoch as well as the logger array
    return epoch_loss / len(loader), epoch_acc / len(loader), loss_logger,acc_logger
```

```python
#This function should perform a single evaluation epoch and will be passed our validation or evaluation/test data
#it WILL NOT be used to train out model
def evaluate(net, device, loader, Loss_fun, loss_logger = None,acc_logger= None):

    epoch_loss = 0
    epoch_acc = 0

    #Set network in evaluation mode
    #Layers like Dropout will be disabled
    #Layers like Batchnorm will stop calculating running mean and standard deviation
    #and use current stored values
    net.eval()

    with torch.no_grad():
        for i, (x, y) in enumerate(loader):

            #Load images and labels to device
            x = x.to(device)
            y = y.type(torch.LongTensor).to(device) # y is the corresponding label

            #Forward pass of image through network
            fx = net(x)

            #Calculate loss using loss function
            loss = Loss_fun(fx, y)

            #calculate the accuracy
            acc = calculate_accuracy(fx, y)

            #log the cumulative sum of the loss and acc
            epoch_loss += loss.item()
            epoch_acc += acc.item()

            #log the loss for plotting if we passed a logger to the function
            if not (loss_logger is None):
                loss_logger.append(loss.item())

            if not (acc_logger is None):
                acc_logger.append(acc.item())

            #clear_output(True)
            print("EVALUATION: | Itteration [%d/%d] | Loss %.2f | Accuracy %.2f%% |" %(i+1 ,len(loader), loss.item(), 100*(epoch_

    #return the avaerage loss and acc from the epoch as well as the logger array
    return epoch_loss / len(loader), epoch_acc / len(loader), loss_logger,acc_logger
```

```
optim_valid_acc = 0
for epoch in range(n_epochs):
    print(epoch)

    #call the training function and pass training dataloader etc
    train_loss, train_acc, training_loss_logger,training_acc_logger= train(net, device, trainloader, optimizer, loss_fn, training

    valid_loss, valid_acc, validation_loss_logger,validation_acc_logger = evaluate(net, device, valloader, loss_fn, validation_lo

    #call the evaluate function and pass validation dataloader etc
    test_loss, test_acc, test_loss_logger,test_acc_logger = evaluate(net, device, testloader, loss_fn, test_loss_logger,test_acc_

    #If this model has the highest performace on the validation set
    #then save a checkpoint
    #{} define a dictionary, each entry of the dictionary is indexed with a string
    if (valid_acc > optim_valid_acc):
        print("Saving Model")
        best_model={
            'epoch':                epoch,
            'model_state_dict':      net.state_dict(),
            'optimizer_state_dict':  optimizer.state_dict(),
            'train_acc':             train_acc,
            'valid_acc':             valid_acc,
        }

    print(f'| Epoch: {epoch+1:02} | Train Loss: {train_loss:.3f} | Train Acc: {train_acc*100:05.2f}% | Val. Loss: {valid_loss:.3f


plt.figure(figsize=(10,10))
train_x = np.linspace(0, n_epochs, len(training_loss_logger))
plt.plot(train_x, training_loss_logger, c = "y")
valid_x = np.linspace(0, n_epochs, len(validation_loss_logger))
plt.plot(valid_x, validation_loss_logger, c = "r")

plt.title("Deep CNN")
plt.legend(["Traning Loss", "Validation Loss"])
plt.xlabel("Epoch")

plt.figure(figsize = (10,10))
train_x = np.linspace(0, n_epochs, len(training_acc_logger))
plt.plot(train_x, (training_acc_logger), c = "y")
valid_x = np.linspace(0, n_epochs, len(validation_acc_logger))
plt.plot(valid_x, (validation_acc_logger), c = "r")
valid_x = np.linspace(0, n_epochs, len(test_acc_logger))
plt.plot(valid_x, (test_acc_logger), c = "b")


plt.title("Deep CNN")
plt.legend(["Traning Accuracy", "Validation Accuracy","Testing Accuracy"])
plt.xlabel("Epoch")
```





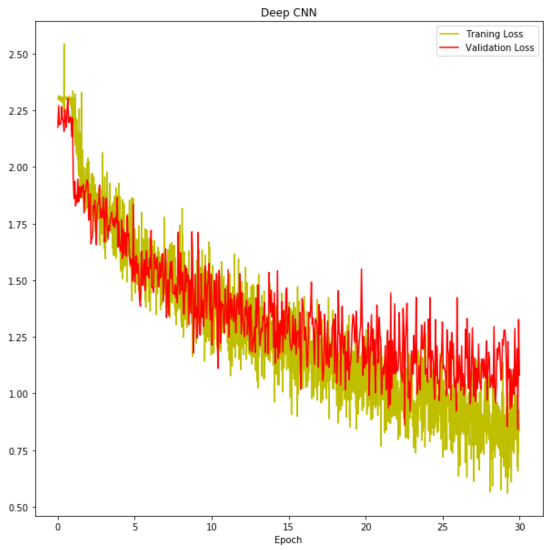
```
Saving Model
| Epoch: 30 | Train Loss: 0.735 | Train Acc: 72.67% | Val. Loss: 1.217 | Val. Acc: 59.35% |
```

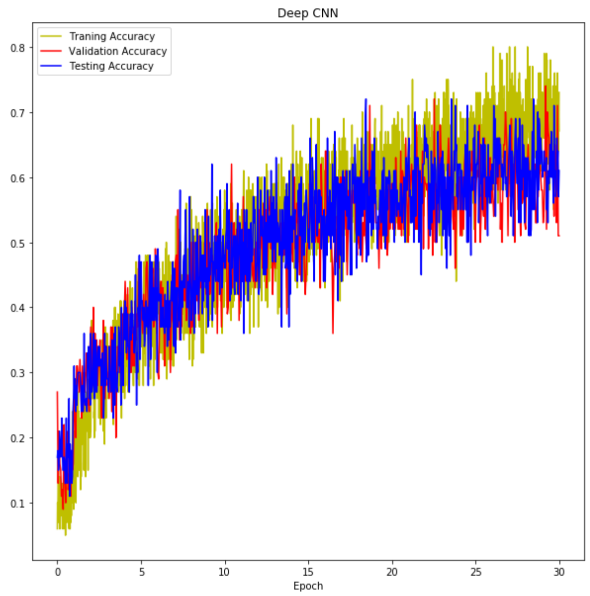
Conclusion:

The model was trained for 30 epochs, and its learning rate is 1e-3. Adam optimizer is applied. The loss of validation begins to diverge while the loss of training decrease when the epoch is more than 17. The accuracy of the model over the validation and testing sets are around 59%.

2.

```
transform = transforms.Compose([
                                T.ToPILImage(),
                                #T.RandomResizedCrop(64,ratio = (1,1)),
                                #T.RandomApply([choices], 0.5),
                                T.RandomCrop(image_size),      #Random crop
                                T.RandomHorizontalFlip(p=0.5), #random horizontal flip with 50% chance
                                T.ToTensor(),
                                T.Normalize(mean=[0.485, 0.456, 0.406],
                                            std=[0.229, 0.224, 0.225])])
```






```
Saving Model
| Epoch: 30 | Train Loss: 0.839 | Train Acc: 68.70% | Val. Loss: 1.091 | Val. Acc: 60.65% |
```

Conclusion:

A couple of transformations were applied including random cropping, random horizontal flipping of images. Channel normalization is also applied.

At 30 epochs, the accuracy for the testing set and validation set were floating around 60%. After introducing the transformation, the validation loss has not diverged unlike what happen before. It means the model has become more robust.

3.

Ideas to improve the performance of model:

1. We can add dropout layers to regularize the network with the position of dropouts and the dropout probabilities.
2. We could try replacing the Adam optimizer with the RMSProp.
3. We can perform the Early Stopping. When it detects the overfitting performance of the model on training and a held validation dataset for each epoch, then training will stop.
4. Increasing the learning rate to reduce the number of epochs.
5. Replace the activation function (ReLU) with Swish activation function.