CSC413 Assignment 2

Ming Gong 1004709130
Hongyu Chen 1005398197

```
import pandas
import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.optim as optim
```

Question 1:Data

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Part (a)

We care about the accuracies of the men's and women's shoes as two separate measures is because men's and women's shoes are mostly different. Women's shoes have more choices in style, color and shapes. And men's shoes are much bigger in size than women's. Also we do not want our model to end up pairing a man's shoe and a women's shoe together. That's definetly wrong. If we seperate measures it will make the accuracy higher and also the training will be easier.

Part (b)

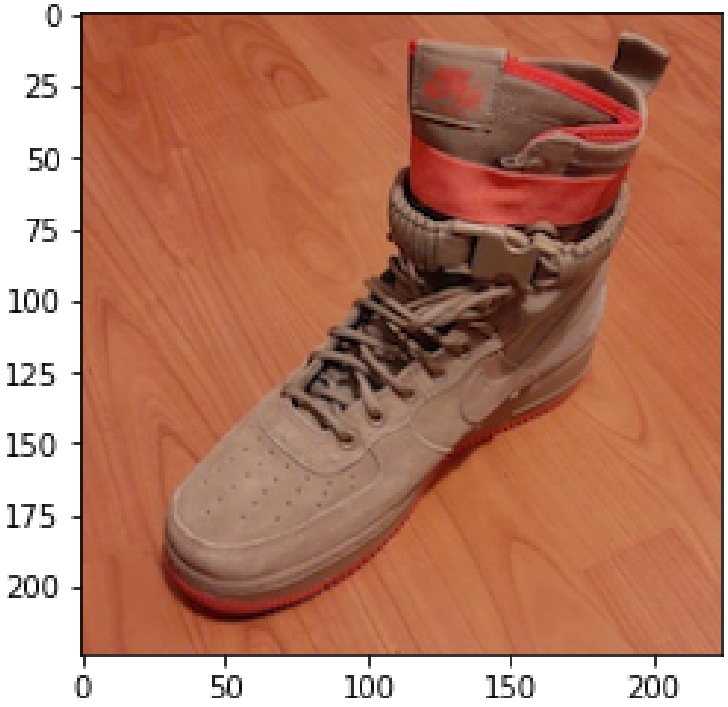
```
import glob
def data_creator(path):
  images = {}
  for file in glob.glob(path):
      filename = file.split("/")[-1]    # get the name of the .jpg
                                          file
      img = plt.imread(file)            # read the image as a numpy
                                          array
      images[filename] = img[:, :, :3] # remove the alpha channel
  img_name = sorted(images.keys(), key=lambda x: (x[0], int(x.
                                      partition('s')[2].partition('_')
                                      [0] or x.partition('u')[2].
                                      partition('_')[0])))
  num_student = len(img_name) // 6
  data = np.zeros((num_student,3,2,224,224,3))
```

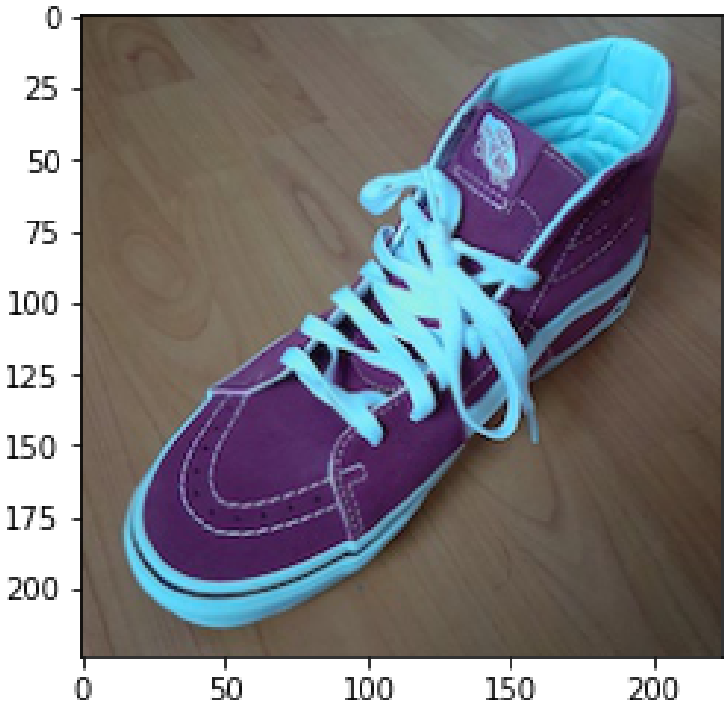
```
  for i in range(len(img_name)):
    name = img_name[i]
    if "left" in name:
      if "_0" in name:
        data[i//6,0,0,:,:,:] = images[name]
      elif "_1" in name:
        data[i//6,1,0,:,:,:] = images[name]
      elif "_2" in name:
        data[i//6,2,0,:,:,:] = images[name]
    else:
      if "_0" in name:
        data[i//6,0,1,:,:,:] = images[name]
      elif "_1" in name:
        data[i//6,1,1,:,:,:] = images[name]
      elif "_2" in name:
        data[i//6,2,1,:,:,:] = images[name]
  return data / 255 - 0.5

#Modify the path to your own
train_data = data_creator("/content/gdrive/My Drive/CSC413/A/A2/
                                    413data/train/*.jpg")[:80]
valid_data = data_creator("/content/gdrive/My Drive/CSC413/A/A2/
                                    413data/train/*.jpg")[80:]
test_w = data_creator("/content/gdrive/My Drive/CSC413/A/A2/413data/
                                    test_m/*.jpg")
test_m = data_creator("/content/gdrive/My Drive/CSC413/A/A2/413data/
                                    test_w/*.jpg")

plt.figure()
plt.imshow(train_data[4,0,0,:,:,:]+0.5) # left shoe of first pair
                                    submitted by 5th student
plt.figure()
plt.imshow(train_data[4,0,1,:,:,:]+0.5) # right shoe of first pair
                                    submitted by 5th student
plt.figure()
plt.imshow(train_data[4,1,1,:,:,:]+0.5) # right shoe of second pair
                                    submitted by 5th student
```


0
25
50
75
100
125
150
175
200
0
50
100
150
200


0
25
50
75
100
125
150
175
200
0
50
100
150
200



Part (c)

```
def generate_same_pair(data):
  ans = np.copy(data)
  return np.reshape(ans,((len(data)*3,448,224,3)))

print(train_data.shape) # if this is [N, 3, 2, 224, 224, 3]
print(generate_same_pair(train_data).shape) # should be [N*3, 448,
                                    224, 3]
plt.imshow(generate_same_pair(train_data)[0]+0.5) # should show 2
                                    shoes from the same pair (added 0.
                                    5 to make the image normal)
```

```
(80, 3, 2, 224, 224, 3)
(240, 448, 224, 3)
```


0
50
100
150
200
250
300
350
400
0
100
200

Part (d)

```
def generate_different_pair(data):
  diff = np.zeros(np.shape(data))
  diff[:,0,0,:,:,:] = data[:,0,0,:,:,:]
  diff[:,1,0,:,:,:] = data[:,2,1,:,:,:]
  diff[:,2,0,:,:,:] = data[:,0,1,:,:,:]
  diff[:,0,1,:,:,:] = data[:,2,0,:,:,:]
  diff[:,1,1,:,:,:] = data[:,1,1,:,:,:]
  diff[:,2,1,:,:,:] = data[:,1,0,:,:,:]
  #1L 3L
  #3R 2R
  #1R 2L
  return generate_same_pair(diff)

print(train_data.shape) # if this is [N, 3, 2, 224, 224, 3]
print(generate_different_pair(train_data).shape) # should be [N*3,
                                    448, 224, 3]
plt.imshow(generate_different_pair(train_data)[0]+0.5) # should show
                                     2 shoes from different pairs(
                                    added 0.5 to make the image normal
                                    )
```

```
(80, 3, 2, 224, 224, 3)
(240, 448, 224, 3)
```

Part (e)

We want to insist that the different pairs of shoes still com from the same student is because we want them to still have the same background.Since they are taken by the same student, two shoes will still has a same(similar) background. So our model

will not focus on the difference between background image, we want the model focus on the shoes itself.

Part (f)
It's very important to have balanced data because if we have 99% of the images are the shoes that are not from the same pair, only 1% of the images are shoes from the same pair. Our model will finally start to predict the shoes from different pair more then predict shoes from the same pair since most of our data are not from the same pair. This is definitely not what we want from our model and this will cause a problem.

Question 2. 1D Convolutions
Part (a)
Forward pass:
input x = [1,3,2,1,1,0,1]
First layer: 1D kernel = [-1, 2, -1], $b^{(1)} = 0$
For elements in x:

1st element "3" we have $\begin{matrix} 1 & 3 & 2 \\ -1 & 2 & -1 \end{matrix}$ = -1 + 6 + -2 = 3

2nd element "2" we have $\begin{matrix} 3 & 2 & 1 \\ -1 & 2 & -1 \end{matrix}$ = -3 + 4 + -1 = 0

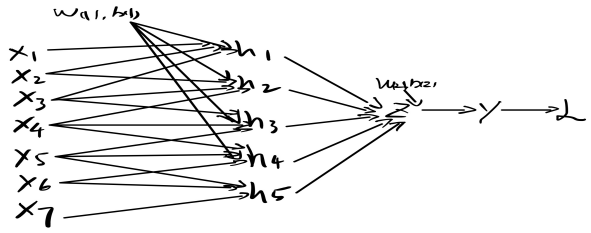
3rd element "1" we have $\begin{matrix} 2 & 1 & 1 \\ -1 & 2 & -1 \end{matrix}$ = -2 + 2 + -1 = -1

4th element "1" we have $\begin{matrix} 1 & 1 & 0 \\ -1 & 2 & -1 \end{matrix}$ = -1 + 2 + 0 = 1

5th element "0" we have $\begin{matrix} 1 & 0 & 1 \\ -1 & 2 & -1 \end{matrix}$ = -1 + 0 + -1 = -2

We have [3,0,-1,1,-2]. Then apply the ReLu activation after this layer we get [3,0,0,1,0]. For the second layer: $W_k^{(2)} = 1$ and $b^{(2)} = 1$. We have 3+1+1 = 5. Then apply the sigmoid function we get $\frac{e^5}{e^5-1} \approx 0.9933$. Then we will use the cross-entropy loss. Since t = 0, $\mathcal{L}(y,t) = -log(1-y) = 7.22316$

Part (b)



$\mathcal{L}(y,t) = -log(1-y)$ since t =0. $y = \frac{1}{1+e^{-z}}$ (sigmoid function). $z = W^{(2)}(h_1 + h_2 + h_3 + h_4 + h_5) + b^{(2)}$. $h_{(k)} = W^{(1)}(x_k + x_{k+1} + x_{k+2}) + b^{(1)}$
$\frac{\partial \mathcal{L}}{\partial W^{(1)}} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial z}(\frac{\partial z}{\partial h_1}\frac{\partial h_1}{\partial W^{(1)}} + \frac{\partial z}{\partial h_2}\frac{\partial h_2}{\partial W^{(1)}} + \frac{\partial z}{\partial h_3}\frac{\partial h_3}{\partial W^{(1)}} + \frac{\partial z}{\partial h_4}\frac{\partial h_4}{\partial W^{(1)}} + \frac{\partial z}{\partial h_5}\frac{\partial h_5}{\partial W^{(1)}})$
$= \frac{1}{1-y}\frac{e^z}{(e^z+1)^2}(W^{(2)}(x_1+x_2+x_3) + 0(x_2+x_3+x_4) + 0(x_3+x_4+x_5) + W^{(2)}(x_4+x_5+x_6) + 0(x_5+x_6+x_7))$
$= \frac{1}{1-0.9933}\frac{e^5}{(e^5+1)^2}[(1+3+2)+(1+1+0)]$ we know from part(a) y = 0.9933 and z =5
= 0.9922*8
= 7.9376

Part (c)
Using a 2D convolutional kernel is not quite reasonable because each word is a one-hot vector that represents a word. 2D convolutional kernel is to combine information from all channels into a 2D output. It's not useful for 1 dimension. If we choose a 3x3 kernel here, it doesn't quite make sense that this filter will cover part of the one-hot vector represent a specific and another part of another one-hot vector represent another word. We want the 1D convolution filter to at least covers the entire word. Which means a V dimensional convolution filter. So our filter can move through each words one by one to fully scan the data and also do it properly.

Part (d)
input_size - $(K-1)(N-1)$

Question 3. CNN Implementation
Part (a)

```
class CNN(nn.Module):
    def __init__(self, n=4):
        super(CNN, self).__init__()
        self.n = n
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=n,
                                            kernel_size=5, stride=2,
                                            padding=2)
```

```
        self.conv2 = nn.Conv2d(in_channels=n, out_channels=n*2,
                                            kernel_size=5, stride=2,
                                            padding=2)
        self.conv3 = nn.Conv2d(in_channels=n*2, out_channels=n*4,
                                            kernel_size=5, stride=2,
                                            padding=2)
        self.conv4 = nn.Conv2d(in_channels=n*4, out_channels=n*8,
                                            kernel_size=5, stride=2,
                                            padding=2)
        self.fc1 = nn.Linear(n*8*28*14, 100)
        self.fc2 = nn.Linear(100, 2)

    def forward(self, x):
        x = x.transpose(2, 3)
        x = torch.relu(self.conv1(x))
        x = torch.relu(self.conv2(x))
        x = torch.relu(self.conv3(x))
        x = torch.relu(self.conv4(x))
        x = x.view(-1, self.n*8 * 28 * 14)
        x = torch.relu(self.fc1(x))
        return self.fc2(x)
```

Part (b)

```
class CNNChannel(nn.Module):
    def __init__(self, n=4):
        super(CNNChannel, self).__init__()
        self.n = n
        self.conv1 = nn.Conv2d(in_channels=6, out_channels=n,
                                            kernel_size=5, stride=2,
                                            padding=2)
        self.conv2 = nn.Conv2d(in_channels=n, out_channels=n*2,
                                            kernel_size=5, stride=2,
                                            padding=2)
        self.conv3 = nn.Conv2d(in_channels=n*2, out_channels=n*4,
                                            kernel_size=5, stride=2,
                                            padding=2)
        self.conv4 = nn.Conv2d(in_channels=n*4, out_channels=n*8,
                                            kernel_size=5, stride=2,
                                            padding=2)
        self.fc1 = nn.Linear(n*8*14*14, 100)
        self.fc2 = nn.Linear(100, 2)

    def forward(self,x):
        left = x[:, :, :, :224]
```

```
        right = x[:, :, :, 224:]
        x = torch.cat((left, right), axis=1)
        x = torch.relu(self.conv1(x))
        x = torch.relu(self.conv2(x))
        x = torch.relu(self.conv3(x))
        x = torch.relu(self.conv4(x))
        x = x.view(-1, self.n*8 * 14 * 14)
        x = torch.relu(self.fc1(x))
        return self.fc2(x)
```

Part (c)

The sigmoid function is used in BCEWithLogitsLoss and softmax function is used in CrossEntropyLoss. This architecture will generally give us better performance is because softmax activation function normalize the output of a network to a probability distribution over predicted output classes. Prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying softmax, each component will be in the interval (0,1), and the components will add up to 1, so that they can be interpreted as probabilities. But the sigmoid function will not. Since we will use one-hot vector to represent our target value. By using Cross Entropy Loss with softmax activation function we will have a probability distribution for each pair of the input shoes. After we get both the probability and the one-hot target value, we can figure out the loss easily. It will also save us some computational time.

Part (d)

CNN model:

k = 5

1. The input layer has 0 parameters
2. Parameter in conv1: [(5*5*3)+1]*n = 76n
3. Parameter in conv2: [(5*5*n)+1]*n*2 = $50n^2 + 2n$
4. Parameter in conv3: [(5*5*n*2)+1]*n*4 = $200n^2 + 4n$
5. Parameter in conv4: [(5*5*n*4)+1]*n*8 = $800n^2 + 8n$
6. Parameter in fc1: n*8*28*14*100 + 100 = 313600n+100
7. Parameter in fc2: 100*2+ 2 = 202

CNNChannel model:

k = 5

1. The input layer has 0 parameters
2. Parameter in conv1: [(5*5*6)+1]*n = 151n
3. Parameter in conv2: [(5*5*n)+1]*n*2 = $50n^2 + 2n$
4. Parameter in conv3: [(5*5*n*2)+1]*n*4 = $200n^2 + 4n$

5. Parameter in conv4: [(5*5*n*4)+1]*n*8 = $800n^2 + 8n$
6. Parameter in fc1: n*8*14*14*100 + 100 = $156800n + 100$
7. Parameter in fc2: 100*2+ 2 = 202

Part (e)

CNNChannel model will perform better. In CNNChannel model, images are concatenated along the channel dimension that it will be easier to compare images' differences. Unlike CNNChannel, in CNN model images are compared side by side that will make comparison harder.

Part (f)

Since we're studying units from the center of the feature map/image so you don't have to worry about hitting the image boundary. So we do not need to really care about the padding here.

Let k be the kernel size (we can choose 3 or 5 here) and all of the kernel size for each convolution layers are the same. The number of pixels (from one shoe image) affect the activation value of that unit after

The first down sampling operation is:
$(k * k) * (2 * 2) = 2k * 2k$

The second down sampling operation is:
$(2k * 2k) * (k * k) * (2 * 2) = 4k^2 * 4k^2$

The third down sampling operation is:
$(4k^2 * 4k^2) * (k * k) * (2 * 2) = 8k^3 * 8k^3$

The fourth down sampling operation is:
$(8k^3 * 8k^3) * (k * k) * (2 * 2) = 16k^4 * 16k^4$

Part (g)

We wish to track these two values separately because positive samples are the data from same pairs and the negative samples are the data from different pairs. It is important for the model to predict in both samples.

```
def get_accuracy(model, data, batch_size=50):
    """Compute the model accuracy on the data set. This function
                                        returns two
```

```
    separate values: the model accuracy on the positive samples,
    and the model accuracy on the negative samples.

    Example Usage:

    >>> model = CNN() # create untrained model
    >>> pos_acc, neg_acc= get_accuracy(model, valid_data)
    >>> false_positive = 1 - pos_acc
    >>> false_negative = 1 - neg_acc
    """

    model.eval()
    n = data.shape[0]

    data_pos = generate_same_pair(data)      # should have shape [n
                                        * 3, 448, 224, 3]
    data_neg = generate_different_pair(data) # should have shape [n
                                        * 3, 448, 224, 3]

    pos_correct = 0
    for i in range(0, len(data_pos), batch_size):
        xs = torch.Tensor(data_pos[i:i+batch_size]).transpose(1, 3)
        zs = model(xs)
        pred = zs.max(1, keepdim=True)[1] # get the index of the max
                                          logit
        pred = pred.detach().numpy()
        pos_correct += (pred == 1).sum()

    neg_correct = 0
    for i in range(0, len(data_neg), batch_size):
        xs = torch.Tensor(data_neg[i:i+batch_size]).transpose(1, 3)
        zs = model(xs)
        pred = zs.max(1, keepdim=True)[1] # get the index of the max
                                          logit
        pred = pred.detach().numpy()
        neg_correct += (pred == 0).sum()

    return pos_correct / (n * 3), neg_correct / (n * 3)
```

Question 4. Training

Part (a)

```
def train_model(model, train_data, validation_data, batch_size=100,
                                    learning_rate=0.001,weight_decay=0
                                    ,epochs=1000,checkpoint_path=None)
```

```
                                    :
n = 0
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(),lr=learning_rate,
                                        weight_decay=weight_decay)
data_pos = generate_same_pair(train_data)
data_neg = generate_different_pair(train_data)
iters, losses = [], []
train_accs, val_accs = [], []
pos_train_accs, pos_val_accs = [], []
neg_train_accs, neg_val_accs = [], []
for i in range(epochs):
    np.random.shuffle(data_pos)
    np.random.shuffle(data_neg)
    p1, p2 = 0, batch_size//2
    while p1 < np.shape(data_pos)[0]:
      n += 1
      p2 = p1+batch_size//2
      if p2 >= np.shape(data_pos)[0]:
        break
      xs_p = data_pos[p1:p2]
      xs_n = data_neg[p1:p2]
      p1=p2
      xs_p = (torch.Tensor(xs_p)).transpose(1,3)
      xs_n = (torch.Tensor(xs_n)).transpose(1,3)
      labels_pos = np.ones(batch_size // 2)
      labels_neg = np.zeros(batch_size // 2)
      labels = torch.Tensor(np.concatenate((labels_pos,
                                        labels_neg),axis=0)).
                                        long()
      model.train()
      zs = model(torch.cat((xs_p, xs_n),axis=0))
      loss = criterion(zs,labels)
      loss.backward()
      optimizer.step()
      optimizer.zero_grad()
    losses.append(float(loss)/batch_size)
    train_cost = float(loss.detach().numpy())
    pt,nt = get_accuracy(model, train_data)
    pos_train_accs.append(pt)
    neg_train_accs.append(nt)
    pv,nv = get_accuracy(model, validation_data)
    pos_val_accs.append(pv)
    neg_val_accs.append(nv)
    print("Iter %d. [Positive Val Acc %.0f%%] [Positive Train
                                    Acc %.0f%%] [Negative Val
```

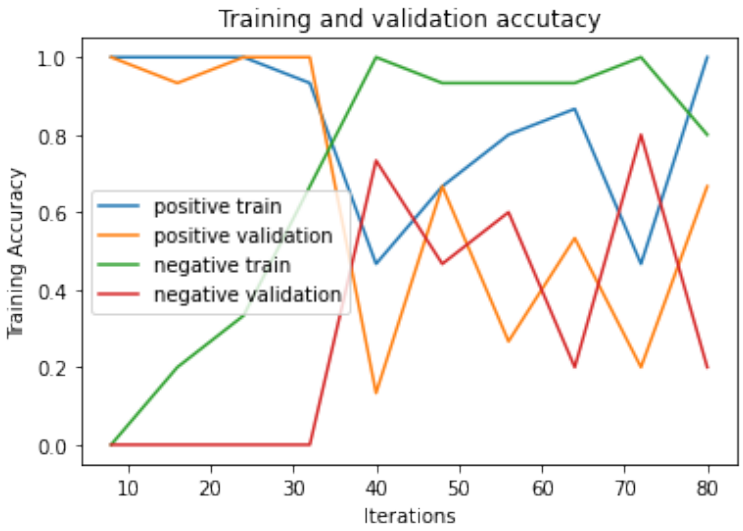
```
                                        Acc %.0f%%] [Negative
                                        Train Acc %.0f%%], Loss %f
                                        " % (
      n, pv * 100, pt * 100,nv * 100, nt * 100, train_cost))
        iters.append(n)
        if (checkpoint_path is not None) and n > 0:
          torch.save(model.state_dict(), checkpoint_path.format(n))
    plt.title("Training and validation accutacy")
    plt.xlabel("Iterations")
    plt.ylabel("Training Accuracy")
    plt.plot(iters, pos_train_accs, label="positive train")
    plt.plot(iters, pos_val_accs, label="positive validation")
    plt.plot(iters, neg_train_accs, label="negative train")
    plt.plot(iters, neg_val_accs, label="negative validation")
    plt.legend()
    plt.show()
    plt.title("Iterations v.s. Loss")
    plt.plot(iters, losses)
    plt.xlabel("Iterations")
    plt.ylabel("Loss")
    plt.show()
```

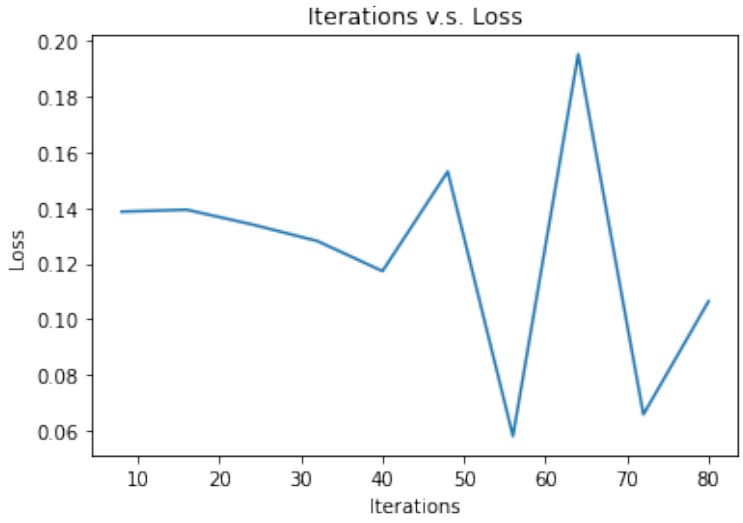
Part (b)

```
cnn_model = CNN()
train_model(cnn_model, train_data[:5], valid_data[:5], batch_size=5,
                  weight_decay=0.0, learning_rate=0.001, epochs=10)
```

```
Iter 8. [Positive Val Acc 100%] [Positive Train Acc 100%] [Negative
                                  Val Acc 0%] [Negative Train Acc 0%
                                  ], Loss 0.693126
Iter 16. [Positive Val Acc 93%] [Positive Train Acc 100%] [Negative
                                  Val Acc 0%] [Negative Train Acc 20
                                  %], Loss 0.696554
Iter 24. [Positive Val Acc 100%] [Positive Train Acc 100%] [Negative
                                   Val Acc 0%] [Negative Train Acc
                                  33%], Loss 0.670122
Iter 32. [Positive Val Acc 100%] [Positive Train Acc 93%] [Negative
                                  Val Acc 0%] [Negative Train Acc 67
                                  %], Loss 0.640402
Iter 40. [Positive Val Acc 13%] [Positive Train Acc 47%] [Negative
                                  Val Acc 73%] [Negative Train Acc
                                  100%], Loss 0.586380
Iter 48. [Positive Val Acc 67%] [Positive Train Acc 67%] [Negative
                                  Val Acc 47%] [Negative Train Acc
```

```
                                    93%], Loss 0.765248
Iter 56. [Positive Val Acc 27%] [Positive Train Acc 80%] [Negative
                                    Val Acc 60%] [Negative Train Acc
                                    93%], Loss 0.290120
Iter 64. [Positive Val Acc 53%] [Positive Train Acc 87%] [Negative
                                    Val Acc 20%] [Negative Train Acc
                                    93%], Loss 0.975636
Iter 72. [Positive Val Acc 20%] [Positive Train Acc 47%] [Negative
                                    Val Acc 80%] [Negative Train Acc
                                    100%], Loss 0.328946
Iter 80. [Positive Val Acc 67%] [Positive Train Acc 100%] [Negative
                                    Val Acc 20%] [Negative Train Acc
                                    80%], Loss 0.532335
```

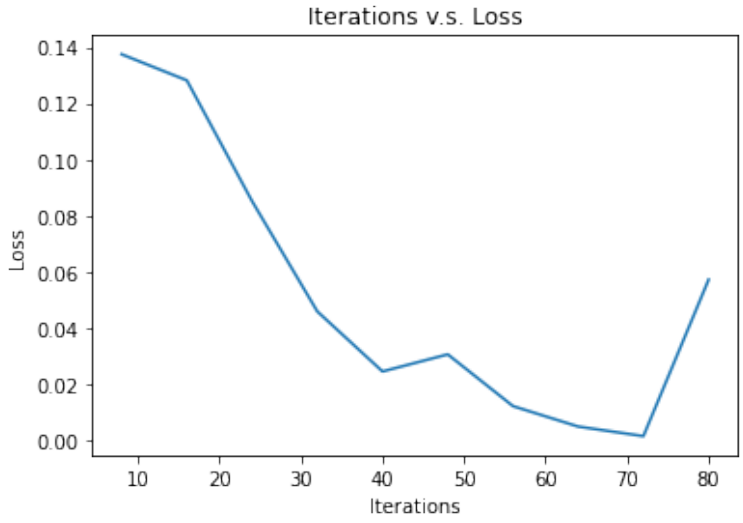





```
cnn_model = CNNChannel()
train_model(cnn_model, train_data[:5], valid_data[:5], batch_size=5,
                  weight_decay=0.0, learning_rate=0.001, epochs=10)
```

```
Iter 8. [Positive Val Acc 100%] [Positive Train Acc 100%] [Negative
                                  Val Acc 33%] [Negative Train Acc
                                  47%], Loss 0.687285
Iter 16. [Positive Val Acc 73%] [Positive Train Acc 100%] [Negative
                                  Val Acc 67%] [Negative Train Acc
                                  93%], Loss 0.640869
```

```
Iter 24. [Positive Val Acc 73%] [Positive Train Acc 100%] [Negative
                                  Val Acc 60%] [Negative Train Acc
                                  80%], Loss 0.425986
Iter 32. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                  Val Acc 40%] [Negative Train Acc
                                  87%], Loss 0.229880
Iter 40. [Positive Val Acc 73%] [Positive Train Acc 100%] [Negative
                                  Val Acc 60%] [Negative Train Acc
                                  100%], Loss 0.123061
Iter 48. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                  Val Acc 53%] [Negative Train Acc
                                  100%], Loss 0.153466
Iter 56. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                  Val Acc 47%] [Negative Train Acc
                                  100%], Loss 0.061453
Iter 64. [Positive Val Acc 47%] [Positive Train Acc 93%] [Negative
                                  Val Acc 67%] [Negative Train Acc
                                  100%], Loss 0.025040
Iter 72. [Positive Val Acc 93%] [Positive Train Acc 100%] [Negative
                                  Val Acc 40%] [Negative Train Acc
                                  73%], Loss 0.008060
Iter 80. [Positive Val Acc 13%] [Positive Train Acc 73%] [Negative
                                  Val Acc 80%] [Negative Train Acc
                                  93%], Loss 0.286346
```



Part (c)

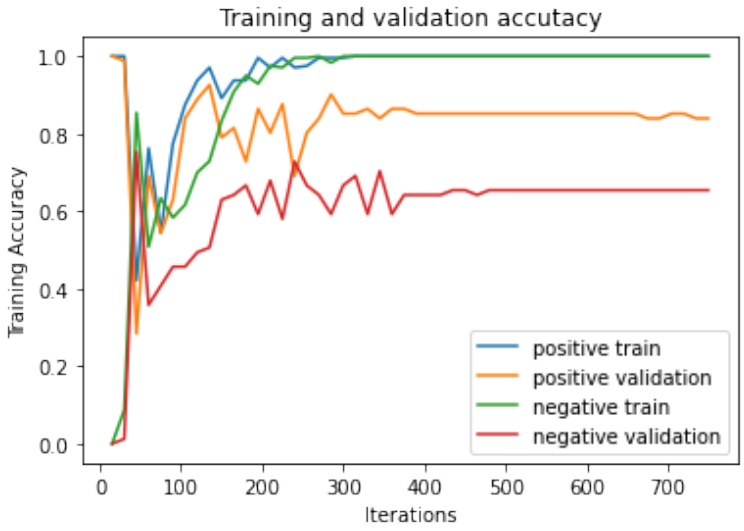
```
cnn_model = CNN()
train_model(cnn_model, train_data, valid_data, batch_size=32,
                  weight_decay=0.0, learning_rate=0.001, epochs=50,
                                                    checkpoint_path=
                                                     '/content/
                                                    gdrive/My Drive/
                                                    CSC413/A/A2/m1/
                                                    ckpt-{}.pk')
```

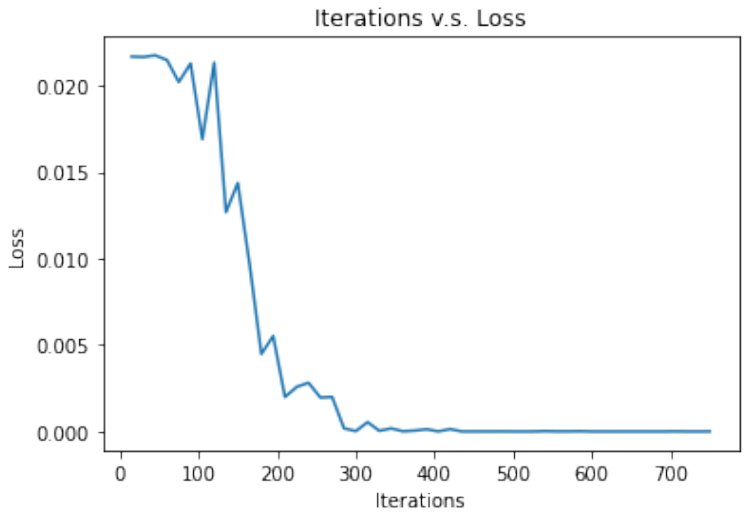
```
Iter 15. [Positive Val Acc 100%] [Positive Train Acc 100%] [Negative
                                  Val Acc 0%] [Negative Train Acc 0
                                 %], Loss 0.693636
Iter 30. [Positive Val Acc 99%] [Positive Train Acc 100%] [Negative
                                  Val Acc 1%] [Negative Train Acc 9%
                                 ], Loss 0.692970
Iter 45. [Positive Val Acc 28%] [Positive Train Acc 42%] [Negative
                                  Val Acc 75%] [Negative Train Acc
                                  85%], Loss 0.695957
Iter 60. [Positive Val Acc 69%] [Positive Train Acc 76%] [Negative
                                  Val Acc 36%] [Negative Train Acc
                                  51%], Loss 0.687246
Iter 75. [Positive Val Acc 54%] [Positive Train Acc 55%] [Negative
                                  Val Acc 41%] [Negative Train Acc
                                  63%], Loss 0.646588
Iter 90. [Positive Val Acc 63%] [Positive Train Acc 78%] [Negative
                                  Val Acc 46%] [Negative Train Acc
                                  58%], Loss 0.680975
Iter 105. [Positive Val Acc 84%] [Positive Train Acc 88%] [Negative
                                  Val Acc 46%] [Negative Train Acc
                                  62%], Loss 0.540921
Iter 120. [Positive Val Acc 89%] [Positive Train Acc 94%] [Negative
                                  Val Acc 49%] [Negative Train Acc
                                  70%], Loss 0.682389
Iter 135. [Positive Val Acc 93%] [Positive Train Acc 97%] [Negative
                                  Val Acc 51%] [Negative Train Acc
                                  73%], Loss 0.405775
Iter 150. [Positive Val Acc 79%] [Positive Train Acc 89%] [Negative
                                  Val Acc 63%] [Negative Train Acc
                                  83%], Loss 0.459907
Iter 165. [Positive Val Acc 81%] [Positive Train Acc 94%] [Negative
                                  Val Acc 64%] [Negative Train Acc
                                  91%], Loss 0.309313
Iter 180. [Positive Val Acc 73%] [Positive Train Acc 94%] [Negative
                                  Val Acc 67%] [Negative Train Acc
                                  95%], Loss 0.143407
Iter 195. [Positive Val Acc 86%] [Positive Train Acc 100%] [Negative
                                  Val Acc 59%] [Negative Train Acc
                                  93%], Loss 0.176644
Iter 210. [Positive Val Acc 80%] [Positive Train Acc 97%] [Negative
                                  Val Acc 68%] [Negative Train Acc
                                  98%], Loss 0.063812
Iter 225. [Positive Val Acc 88%] [Positive Train Acc 100%] [Negative
                                  Val Acc 58%] [Negative Train Acc
                                  97%], Loss 0.082684
```

```
Iter 240. [Positive Val Acc 69%] [Positive Train Acc 97%] [Negative
 Val Acc 73%] [Negative Train Acc
 100%], Loss 0.090151
Iter 255. [Positive Val Acc 80%] [Positive Train Acc 98%] [Negative
 Val Acc 67%] [Negative Train Acc
 100%], Loss 0.062838
Iter 270. [Positive Val Acc 84%] [Positive Train Acc 100%] [Negative
  Val Acc 64%] [Negative Train Acc
 100%], Loss 0.063895
Iter 285. [Positive Val Acc 90%] [Positive Train Acc 100%] [Negative
  Val Acc 59%] [Negative Train Acc
 98%], Loss 0.005963
Iter 300. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
  Val Acc 67%] [Negative Train Acc
 100%], Loss 0.000643
Iter 315. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
  Val Acc 69%] [Negative Train Acc
 100%], Loss 0.017297
Iter 330. [Positive Val Acc 86%] [Positive Train Acc 100%] [Negative
  Val Acc 59%] [Negative Train Acc
 100%], Loss 0.001285
Iter 345. [Positive Val Acc 84%] [Positive Train Acc 100%] [Negative
  Val Acc 70%] [Negative Train Acc
 100%], Loss 0.005768
Iter 360. [Positive Val Acc 86%] [Positive Train Acc 100%] [Negative
  Val Acc 59%] [Negative Train Acc
 100%], Loss 0.000407
Iter 375. [Positive Val Acc 86%] [Positive Train Acc 100%] [Negative
  Val Acc 64%] [Negative Train Acc
 100%], Loss 0.001702
Iter 390. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
  Val Acc 64%] [Negative Train Acc
 100%], Loss 0.004169
Iter 405. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
  Val Acc 64%] [Negative Train Acc
 100%], Loss 0.000219
Iter 420. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
  Val Acc 64%] [Negative Train Acc
 100%], Loss 0.004426
Iter 435. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
  Val Acc 65%] [Negative Train Acc
 100%], Loss 0.000111
Iter 450. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
  Val Acc 65%] [Negative Train Acc
 100%], Loss 0.000116
Iter 465. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
```

```
                                    Val Acc 64%] [Negative Train Acc
                                   100%], Loss 0.000108
Iter 480. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
                                   100%], Loss 0.000184
Iter 495. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
                                   100%], Loss 0.000117
Iter 510. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
                                   100%], Loss 0.000040
Iter 525. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
                                   100%], Loss 0.000058
Iter 540. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
                                   100%], Loss 0.000653
Iter 555. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
                                   100%], Loss 0.000283
Iter 570. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
                                   100%], Loss 0.000270
Iter 585. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
                                   100%], Loss 0.000541
Iter 600. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
                                   100%], Loss 0.000041
Iter 615. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
                                   100%], Loss 0.000062
Iter 630. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
                                   100%], Loss 0.000030
Iter 645. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
                                   100%], Loss 0.000051
Iter 660. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
                                   100%], Loss 0.000058
Iter 675. [Positive Val Acc 84%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
                                   100%], Loss 0.000033
Iter 690. [Positive Val Acc 84%] [Positive Train Acc 100%] [Negative
                                    Val Acc 65%] [Negative Train Acc
```

```
                                    100%], Loss 0.000054
Iter 705. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                     Val Acc 65%] [Negative Train Acc
                                    100%], Loss 0.000269
Iter 720. [Positive Val Acc 85%] [Positive Train Acc 100%] [Negative
                                     Val Acc 65%] [Negative Train Acc
                                    100%], Loss 0.000042
Iter 735. [Positive Val Acc 84%] [Positive Train Acc 100%] [Negative
                                     Val Acc 65%] [Negative Train Acc
                                    100%], Loss 0.000025
Iter 750. [Positive Val Acc 84%] [Positive Train Acc 100%] [Negative
                                     Val Acc 65%] [Negative Train Acc
                                    100%], Loss 0.000175
```





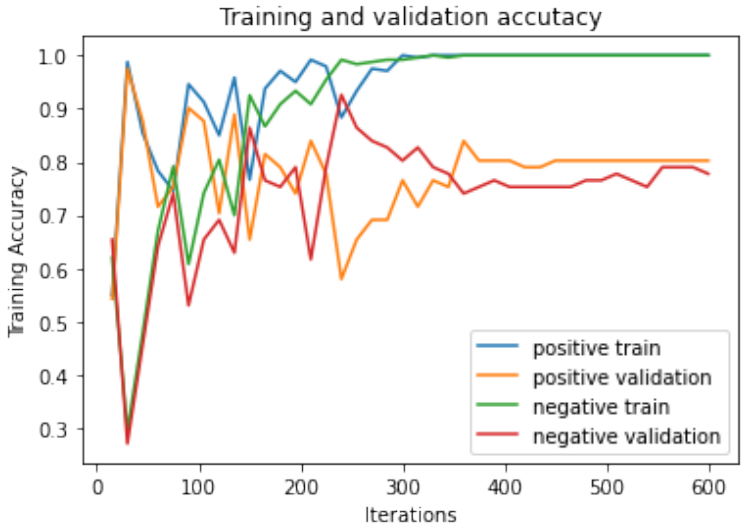

```
cnn_channel = CNNChannel()
train_model(cnn_channel, train_data, valid_data, batch_size=32,
                  weight_decay=0.0, learning_rate=0.001, epochs=40,
                                                      checkpoint_path=
                                                       '/content/
                                                      gdrive/My Drive/
                                                      CSC413/A/A2/m2/
                                                      ckpt-{}.pk')
```

```
Iter 15. [Positive Val Acc 54%] [Positive Train Acc 55%] [Negative
                                  Val Acc 65%] [Negative Train Acc
                                  62%], Loss 0.678375
Iter 30. [Positive Val Acc 98%] [Positive Train Acc 99%] [Negative
                                  Val Acc 27%] [Negative Train Acc
                                  30%], Loss 0.524371
Iter 45. [Positive Val Acc 88%] [Positive Train Acc 85%] [Negative
                                  Val Acc 46%] [Negative Train Acc
                                  48%], Loss 0.605651
Iter 60. [Positive Val Acc 72%] [Positive Train Acc 78%] [Negative
                                  Val Acc 64%] [Negative Train Acc
                                  67%], Loss 0.504089
Iter 75. [Positive Val Acc 75%] [Positive Train Acc 75%] [Negative
                                  Val Acc 74%] [Negative Train Acc
                                  79%], Loss 0.700081
Iter 90. [Positive Val Acc 90%] [Positive Train Acc 95%] [Negative
                                  Val Acc 53%] [Negative Train Acc
                                  61%], Loss 0.427581
```

```
Iter 105. [Positive Val Acc 88%] [Positive Train Acc 91%] [Negative
 Val Acc 65%] [Negative Train Acc
 74%], Loss 0.470535
Iter 120. [Positive Val Acc 70%] [Positive Train Acc 85%] [Negative
 Val Acc 69%] [Negative Train Acc
 80%], Loss 0.371880
Iter 135. [Positive Val Acc 89%] [Positive Train Acc 96%] [Negative
 Val Acc 63%] [Negative Train Acc
 70%], Loss 0.324277
Iter 150. [Positive Val Acc 65%] [Positive Train Acc 77%] [Negative
 Val Acc 86%] [Negative Train Acc
 92%], Loss 0.337620
Iter 165. [Positive Val Acc 81%] [Positive Train Acc 94%] [Negative
 Val Acc 77%] [Negative Train Acc
 87%], Loss 0.232123
Iter 180. [Positive Val Acc 79%] [Positive Train Acc 97%] [Negative
 Val Acc 75%] [Negative Train Acc
 91%], Loss 0.447908
Iter 195. [Positive Val Acc 74%] [Positive Train Acc 95%] [Negative
 Val Acc 79%] [Negative Train Acc
 93%], Loss 0.113111
Iter 210. [Positive Val Acc 84%] [Positive Train Acc 99%] [Negative
 Val Acc 62%] [Negative Train Acc
 91%], Loss 0.124983
Iter 225. [Positive Val Acc 78%] [Positive Train Acc 98%] [Negative
 Val Acc 79%] [Negative Train Acc
 95%], Loss 0.064291
Iter 240. [Positive Val Acc 58%] [Positive Train Acc 88%] [Negative
 Val Acc 93%] [Negative Train Acc
 99%], Loss 0.124311
Iter 255. [Positive Val Acc 65%] [Positive Train Acc 93%] [Negative
 Val Acc 86%] [Negative Train Acc
 98%], Loss 0.181335
Iter 270. [Positive Val Acc 69%] [Positive Train Acc 98%] [Negative
 Val Acc 84%] [Negative Train Acc
 99%], Loss 0.090760
Iter 285. [Positive Val Acc 69%] [Positive Train Acc 97%] [Negative
 Val Acc 83%] [Negative Train Acc
 99%], Loss 0.096465
Iter 300. [Positive Val Acc 77%] [Positive Train Acc 100%] [Negative
  Val Acc 80%] [Negative Train Acc
 99%], Loss 0.036255
Iter 315. [Positive Val Acc 72%] [Positive Train Acc 100%] [Negative
  Val Acc 83%] [Negative Train Acc
 100%], Loss 0.039751
Iter 330. [Positive Val Acc 77%] [Positive Train Acc 100%] [Negative
```

```
                                    Val Acc 79%] [Negative Train Acc
                                   100%], Loss 0.004960
Iter 345. [Positive Val Acc 75%] [Positive Train Acc 100%] [Negative
                                    Val Acc 78%] [Negative Train Acc
                                   100%], Loss 0.043138
Iter 360. [Positive Val Acc 84%] [Positive Train Acc 100%] [Negative
                                    Val Acc 74%] [Negative Train Acc
                                   100%], Loss 0.001436
Iter 375. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                    Val Acc 75%] [Negative Train Acc
                                   100%], Loss 0.003672
Iter 390. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                    Val Acc 77%] [Negative Train Acc
                                   100%], Loss 0.001743
Iter 405. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                    Val Acc 75%] [Negative Train Acc
                                   100%], Loss 0.001493
Iter 420. [Positive Val Acc 79%] [Positive Train Acc 100%] [Negative
                                    Val Acc 75%] [Negative Train Acc
                                   100%], Loss 0.000796
Iter 435. [Positive Val Acc 79%] [Positive Train Acc 100%] [Negative
                                    Val Acc 75%] [Negative Train Acc
                                   100%], Loss 0.002312
Iter 450. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                    Val Acc 75%] [Negative Train Acc
                                   100%], Loss 0.000399
Iter 465. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                    Val Acc 75%] [Negative Train Acc
                                   100%], Loss 0.000614
Iter 480. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                    Val Acc 77%] [Negative Train Acc
                                   100%], Loss 0.000305
Iter 495. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                    Val Acc 77%] [Negative Train Acc
                                   100%], Loss 0.000227
Iter 510. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                    Val Acc 78%] [Negative Train Acc
                                   100%], Loss 0.000342
Iter 525. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                    Val Acc 77%] [Negative Train Acc
                                   100%], Loss 0.000650
Iter 540. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                    Val Acc 75%] [Negative Train Acc
                                   100%], Loss 0.000544
Iter 555. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                    Val Acc 79%] [Negative Train Acc
```

```
                                    100%], Loss 0.000157
Iter 570. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                     Val Acc 79%] [Negative Train Acc
                                    100%], Loss 0.000476
Iter 585. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                     Val Acc 79%] [Negative Train Acc
                                    100%], Loss 0.000153
Iter 600. [Positive Val Acc 80%] [Positive Train Acc 100%] [Negative
                                     Val Acc 78%] [Negative Train Acc
                                    100%], Loss 0.000259
```



How to tune hyperparameters: Find reasonable batch size. Increase it if it takes long time to complete. Find the reasonable epoch. Increase it if the accuracy is increasing. Decrease it if the accuracy get flat.

Part (d)

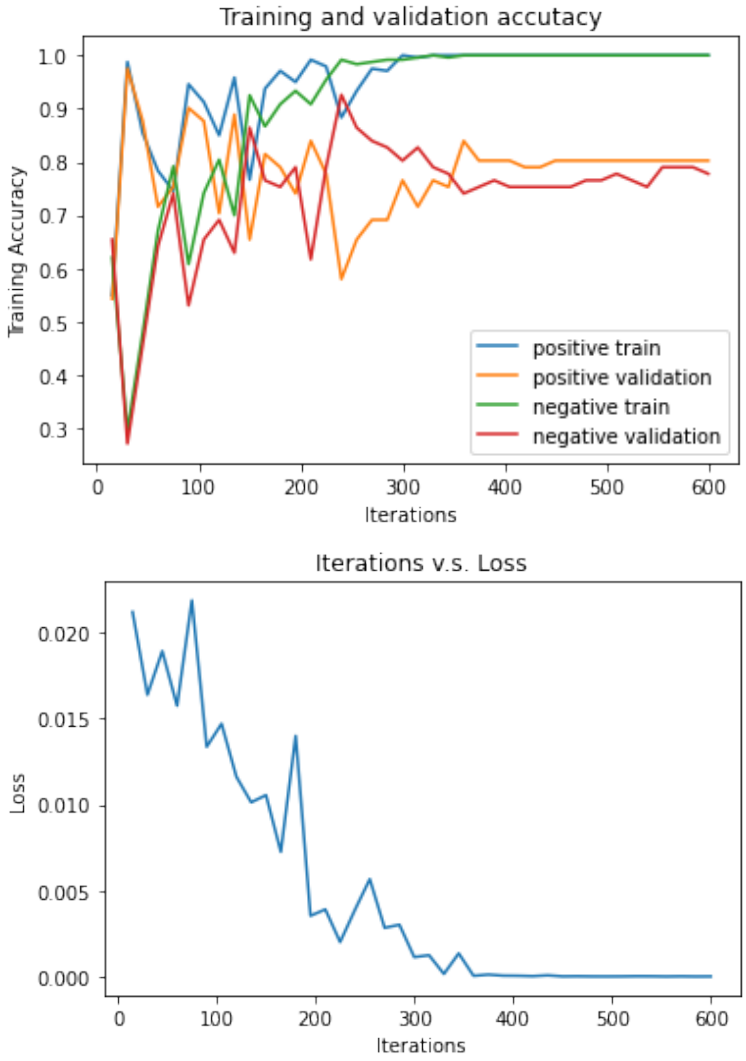


Figure 1: CNNChannel model

Question 5. Test Accuracy

Part (a)

```
def load_checkpoint(model, checkpoint_PATH):
    model_CKPT = torch.load(checkpoint_PATH)
    model.load_state_dict(model_CKPT)
    return model

best_cnn_channel = CNNChannel()
best_cnn_channel = load_checkpoint(best_cnn_channel,'/content/gdrive
                                   /My Drive/CSC413/A/A2/m2/ckpt-600.
                                   pk')
m1, m2 = get_accuracy(best_cnn_channel, test_m)
print("Men: [Positive Acc {}%]   [Negative Acc {}%])".format( m1*100
                                    , m2*100))
w1, w2 = get_accuracy(best_cnn_channel, test_w)
print("Women: [Positive Acc {}%]   [Negative Acc {}%])".format(w1*
                                   100 , w2*100))
```

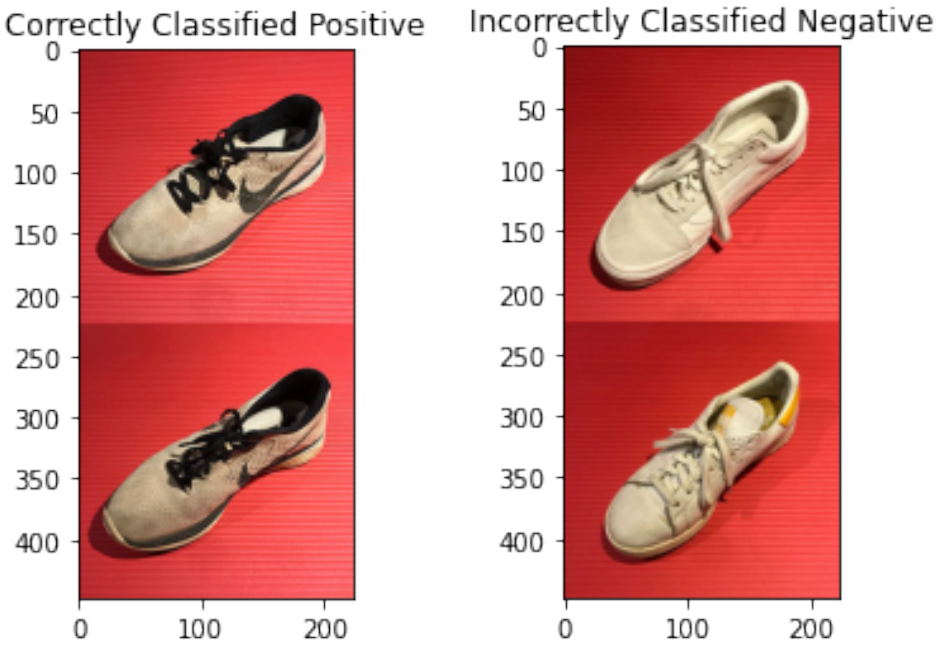
```
Men: [Positive Acc 86.66666666666667%]   [Negative Acc 80.0%])
Women: [Positive Acc 83.33333333333334%]   [Negative Acc 86.
                                   66666666666667%])
```

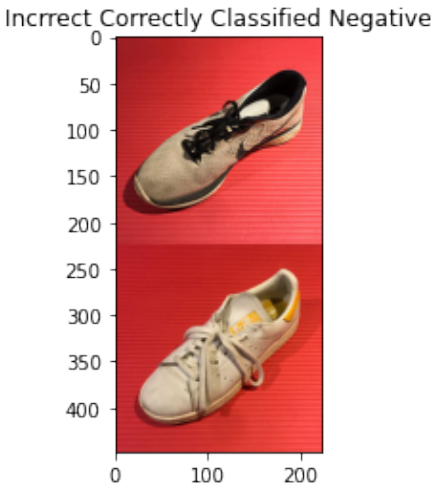
Part (b)

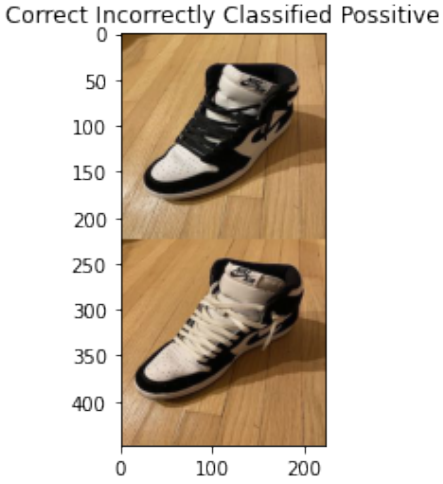
```
def display(data,model):
  same = generate_same_pair(data)
  diff = generate_different_pair(data)
  pc, pi, nc, ni = 0,0,0,0
  for i in range(len(same)):
    pred_pos = model(torch.Tensor(same[i:i+1]).transpose(1, 3)).max(
                                        1, keepdim=True)[1]
    pred_neg = model(torch.Tensor(diff[i:i+1]).transpose(1, 3)).max(
                                        1, keepdim=True)[1]
    if bool(pred_pos.detach().numpy()) and pc == 0:
        pc = 1
        plt.title("Correctly Classified Positive")
        plt.imshow(same[i]+0.5)
        plt.show()
    elif not bool(pred_pos.detach().numpy()) and pi == 0:
        pi = 1
        plt.title("Correct Incorrectly Classified Possitive")
        plt.imshow(same[i]+0.5)
        plt.show()
    elif not bool(pred_neg.detach().numpy()) and nc == 0:
```

```
            nc = 1
            plt.title("Incorrect Correctly Classified Negative")
            plt.imshow(diff[i]+0.5)
            plt.show()
        elif bool(pred_neg.detach().numpy()) and ni == 0:
            ni = 1
            plt.title("Incorrectly Classified Negative")
            plt.imshow(diff[i]+0.5)
            plt.show()
        if pc and pi and nc and ni:
          break
```

```
display(test_m,best_cnn_channel)
```

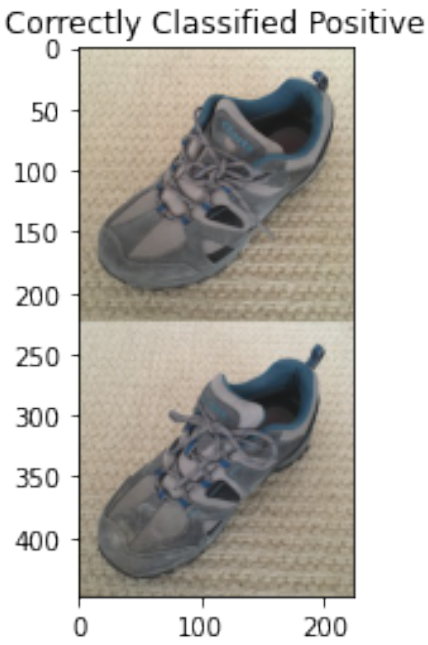


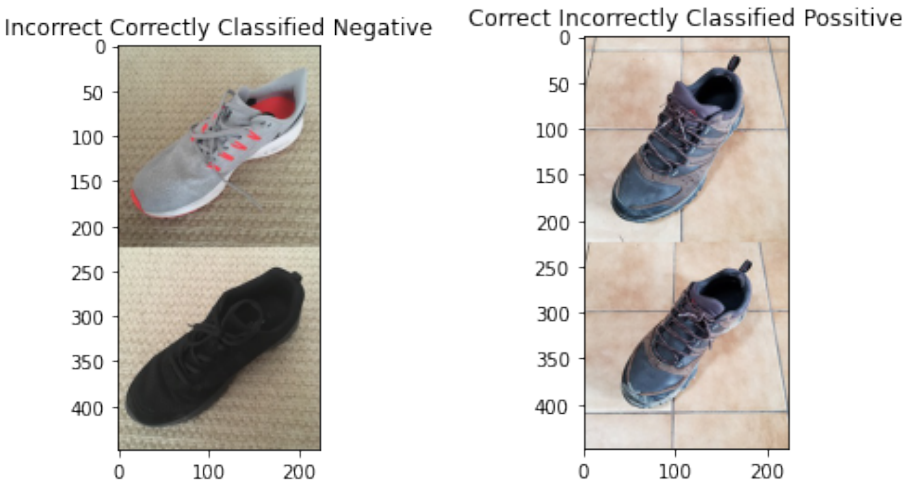




Part (c)

```
display(test_w,best_cnn_channel)
```





Question 6. Graph Neural Networks

Part (a)

a) $h_1^{(1)} = f\left(\sum_{u \in N(v) \cup \{v\}} W^1 h_u^0\right)$ $\quad N(v_1) \cup \{v_1\} = \{v_1, v_2\}$

$= f(W^1 h_1^0 + W^1 h_2^0)$

$= f([1\ 3] + [1\ 2])$

$= f([2\ 5])$ (we will use nonlinearity ReLU as f)

$= [2\ 5]$

$h_2^{(1)} = f(W^1 h_1^0 + W^1 h_2^0 + W^1 h_3^0 + W^1 h_4^0)$

$= f([1\ 3] + [1\ 2] + [0\ 1] + [-1\ -2])$

$= [1\ 4]$

$h_3^{(1)} = f(W^1 h_2^0 + W^1 h_3^0 + W^1 h_4^0)$

$= f([1\ 2] + [0\ 1] + [-1\ -2])$

$= [0 \ 1]$

$h_4^{(1)} = [0 \ 1]$

Part (b)

b) $y = \sigma(q^T(\sum_{v \in V} h_v))$

$= \sigma(q^T([2\ 5] + [1\ 4] + [0\ 1] + [0\ 1]))$

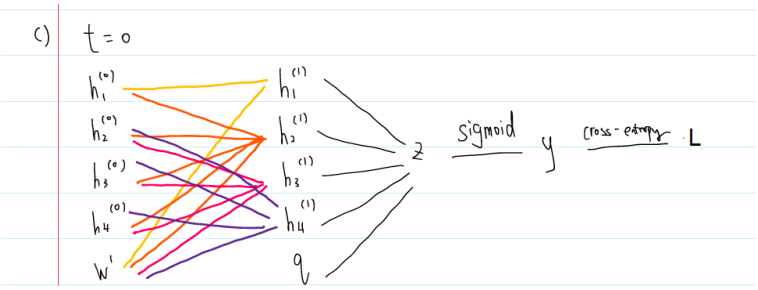
$= \sigma(q^T([3\ 11]))$

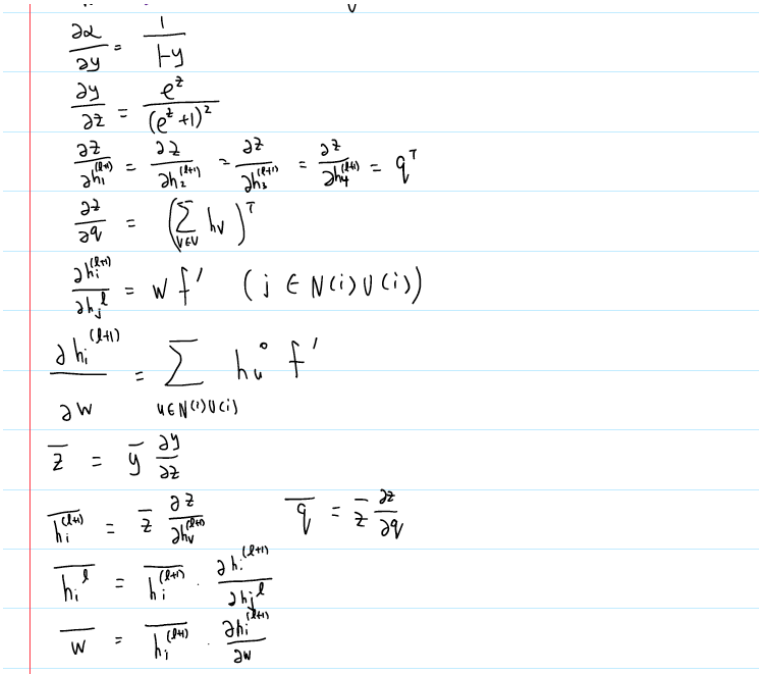
$= \sigma(14)$

$\frac{1}{1+e^{-14}} = 0.99$

$= 0.99$

Part (c)





Question 7. Work Allocation

```
#Ming worked on the assignment on Feb 20 - Mar 5, Hongyu worked on
                                    the assignment on Feb 20 - Mar 5
# We had meetings on Feb 20, 25 and Mar 1, 4. We also had several
                                    short chats during our working
                                    time.
# Ming did Question 1, 2, 3 and Hongyu did question 3, 4, 5, 6
# We built code structure, checked each other's python code function
                                     and discuss the math problems
                                    together.
```