Assignment3

December 27, 2021

1 Assignment 3: Image Classification

In this assignment, we will build a convolutional neural network that can predict whether two shoes are from the **same pair** or from two **different pairs**. This kind of application can have real-world applications: for example to help people who are visually impaired to have more independence.

We will explore two convolutional architectures. While we will give you starter code to help make data processing a bit easier, in this assignment you have a chance to build your neural network all by yourself.

You may modify the starter code as you see fit, including changing the signatures of functions and adding/removing helper functions. However, please make sure that we can understand what you are doing and why.

[ ]:
```
import pandas
import numpy as np
import matplotlib.pyplot as plt

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

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

import pickle
```

1.1 Question 1. Data (20%)

Download the data from the course website.

Unzip the file. There are three main folders: `train`, `test_w` and `test_m`. Data in `train` will be used for training and validation, and the data in the other folders will be used for testing. This is so that the entire class will have the same test sets. The dataset is comprised of triplets of pairs, where each such triplet of image pairs was taken in a similar setting (by the same person).

We've separated `test_w` and `test_m` so that we can track our model performance for women's shoes and men's shoes separately. Each of the test sets contain images of either exclusively men's shoes or women's shoes.

Upload this data to Google Colab. Then, mount Google Drive from your Google Colab notebook:

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

```
Mounted at /content/gdrive
```

After you have done so, read this entire section before proceeding. There are right and wrong ways of processing this data. If you don't make the correct choices, you may find yourself needing to start over. Many machine learning projects fail because of the lack of care taken during the data processing stage.

1.1.1 Part (a) -- 8%

Load the training and test data, and separate your training data into training and validation. Create the numpy arrays `train_data`, `valid_data`, `test_w` and `test_m`, all of which should be of shape `[*, 3, 2, 224, 224, 3]`. The dimensions of these numpy arrays are as follows:

- `*` - the number of triplets allocated to train, valid, or test
- `3` - the 3 pairs of shoe images in that triplet
- `2` - the left/right shoes
- `224` - the height of each image
- `224` - the width of each image
- `3` - the colour channels

So, the item `train_data[4,0,0,:,:,:]` should give us the left shoe of the first image of the fifth person.The item `train_data[4,0,1,:,:,:]` should be the right shoe in the same pair. The item `train_data[4,1,1,:,:,:]` should be the right shoe in a different pair of that same person.

When you first load the images using (for example) `plt.imread`, you may see a numpy array of shape `[224, 224, 4]` instead of `[224, 224, 3]`. That last channel is what's called the alpha channel for transparent pixels, and should be removed. The pixel intensities are stored as an integer between 0 and 255. Make sure you normlize your images, namely, divide the intensities by 255 so that you have floating-point values between 0 and 1. Then, subtract 0.5 so that the elements of `train_data`, `valid_data` and `test_data` are between -0.5 and 0.5. **Note that this step actually makes a huge difference in training!**

This function might take a while to run; it can takes several minutes to just load the files from Google Drive. If you want to avoid running this code multiple times, you can save your numpy arrays and load it later: https://docs.scipy.org/doc/numpy/reference/generated/numpy.save.html

[ ]:
```
import re
import glob
import os
import numpy as np
path_train = "/content/gdrive/MyDrive/data" # TODO - UPDATE ME!

data_set = { "train": None , "test_m": None , "test_w": None }
for key in data_set.keys():
    person_id = set()
    images = {}
    for file in glob.glob(f"{path_train}/{key}/*.jpg"):
```

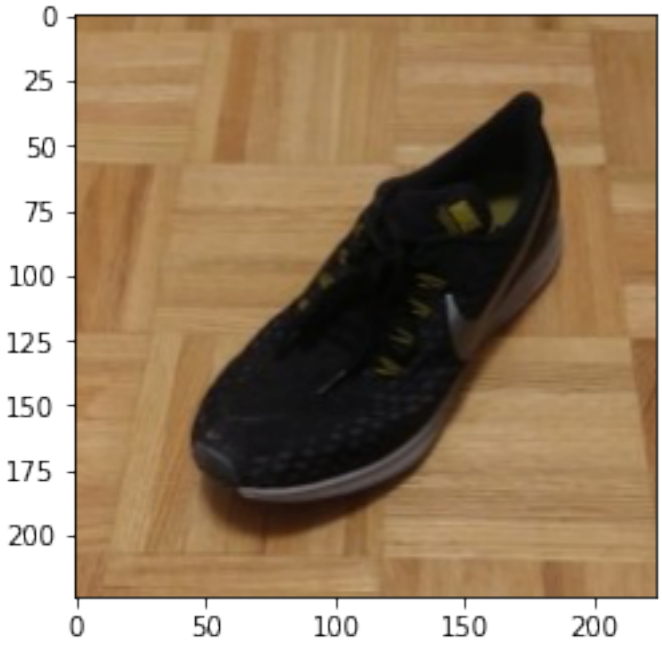
```
        filename = file.split("/")[-1]    # get the name of the .jpg file
        img = plt.imread(file)            # read the image as a numpy array
        person ,triplet, pair, gender = re.findall(r"u(\S+)_(\S+)_(\S+)_(\S+).
↪jpg",filename)[0]
        person_id.add(person)
        images[filename] = img[:, :, :3] # remove the alpha channel

    data_set[key] = np.zeros((len(person_id),3,2,224,224,3), dtype=np.int)
    for index ,person in enumerate(person_id):
      for triplet in range(3):
          filename= f"u{person}_{triplet+1}_right_w.jpg"
          gender = "w" if filename in images else "m"
          data_set[key][index][triplet][1] =␣
↪images[f"u{person}_{triplet+1}_right_{gender}.jpg"]
          data_set[key][index][triplet][0] =␣
↪images[f"u{person}_{triplet+1}_left_{gender}.jpg"]
```

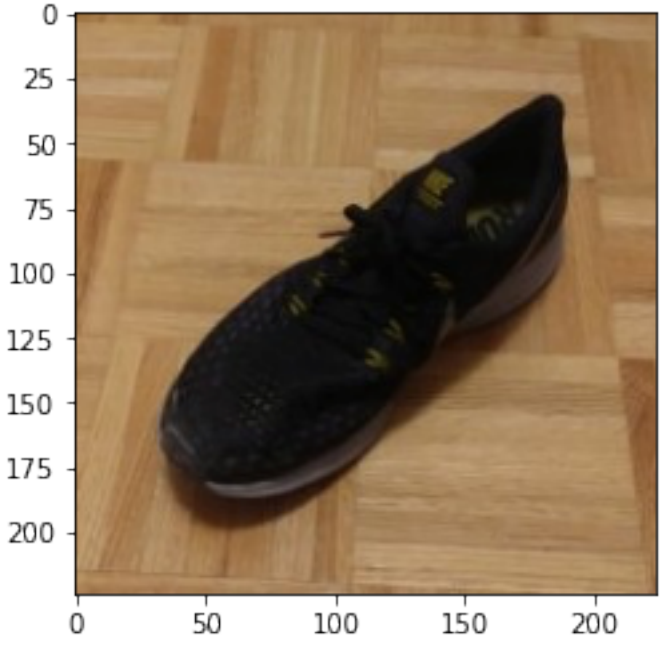
[ ]:
```
from sklearn.model_selection import train_test_split
training_data, validation_data = train_test_split(data_set['train'],␣
↪test_size=0.15)
test_m = data_set["test_m"]
test_w = data_set["test_w"]
```

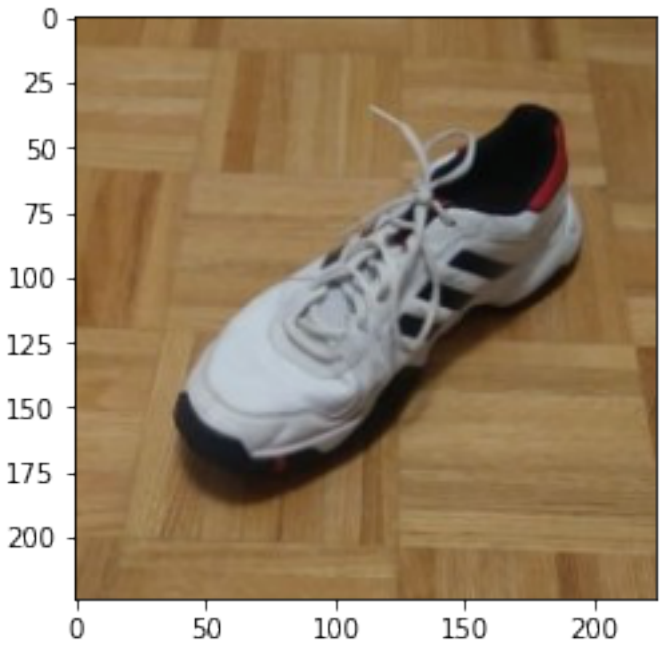
[ ]:
```
training_data = (training_data[:, :, :]/255) - 0.5
validation_data = (validation_data[:, :, :]/255) - 0.5
test_m = (test_m[:, :, :]/255) - 0.5
test_w = (test_w[:, :, :]/255) - 0.5
```

[ ]:
```
# Run this code, include the image in your PDF submission
plt.figure()
plt.imshow(((training_data[4,0,0,:,:,:] + 0.5)*255).astype(np.uint8)) # left␣
↪shoe of first pair submitted by 5th student
plt.figure()
plt.imshow(((training_data[4,0,1,:,:,:] + 0.5)*255).astype(np.uint8)) # right␣
↪shoe of first pair submitted by 5th student
plt.figure()
plt.imshow(((training_data[4,1,1,:,:,:] + 0.5)*255).astype(np.uint8)) # right␣
↪shoe of second pair submitted by 5th student
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7f8d340475d0>
```


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



1.1.2 Part (b) -- 4%

Since we want to train a model that determines whether two shoes come from the **same** pair or **different** pairs, we need to create some labelled training data. Our model will take in an image, either consisting of two shoes from the **same pair** or from **different pairs**. So, we'll need to generate some *positive examples* with images containing two shoes that *are* from the same pair, and some *negative examples* where images containing two shoes that *are not* from the same pair. We'll generate the *positive examples* in this part, and the *negative examples* in the next part.

Write a function `generate_same_pair()` that takes one of the data sets that you produced in part (a), and generates a numpy array where each pair of shoes in the data set is concatenated together. In particular, we'll be concatenating together images of left and right shoes along the **height** axis. Your function `generate_same_pair` should return a numpy array of shape `[*, 448, 224, 3]`.

While at this stage we are working with numpy arrays, later on, we will need to convert this numpy array into a PyTorch tensor with shape `[*, 3, 448, 224]`. For now, we'll keep the RGB channel as the last dimension since that's what `plt.imshow` requires.

[ ]:
```
# Your code goes here
def generate_same_pair(data_array):
    num_persons = data_array.shape[0] # getting the number of people in our␣
 ↪trainging set
    concat_array = np.zeros((num_persons * 3, 224*2, 224, 3)) # for each person␣
 ↪we have three combinations of same pair
    for triplet in range(3):
        right_pair_array = data_array[:,triplet,1,:,:,:]
        left_pair_array = data_array[:,triplet,0,:,:,:]
```

```
        triplet_array = np.concatenate((right_pair_array, left_pair_array),␣
↪axis=1)
        concat_array[triplet*num_persons:(triplet+1)*num_persons] =␣
↪triplet_array # appending to the pair's list for each triplet
    return concat_array

# Run this code, include the result with your PDF submission
train_data = ((training_data + 0.5) * 255)
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)[1]).astype(np.uint8)) # should show␣
↪2 shoes from the same pair
```

```
(95, 3, 2, 224, 224, 3)
(285, 448, 224, 3)
```

[ ]: `<matplotlib.image.AxesImage at 0x7f8d31a98950>`

1.1.3 Part (c) -- 4%

Write a function `generate_different_pair()` that takes one of the data sets that you produced in part (a), and generates a numpy array in the same shape as part (b). However, each image will contain 2 shoes from a **different** pair, but submitted by the **same student**. Do this by jumbling the 3 pairs of shoes submitted by each student.

Theoretically, for each person (triplet of pairs), there are 6 different combinations of "wrong pairs" that we could produce. To keep our data set *balanced*, we will only produce **three** combinations of wrong pairs per unique person. In other words,`generate_same_pairs` and `generate_different_pairs` should return the same number of training examples.

[ ]:
```
import random

def generate_different_pair(data_array):
    num_of_persons = data_array.shape[0]
    paired_array = np.zeros((num_of_persons * 3, 224*2, 224, 3))
    for triplet in range(3):
        rand_list = [0, 1, 2]
        rand_list.remove(triplet)
        wrong_triplet = random.choice(rand_list)

        right_pair_array = data_array[:,triplet,1,:,:,:]
        left_pair_array = data_array[:,wrong_triplet,0,:,:,:]

        triplet_array = np.concatenate((right_pair_array, left_pair_array),␣
 ↪axis=1)
        paired_array[triplet*num_of_persons:(triplet+1)*num_of_persons] =␣
 ↪triplet_array

    return paired_array


# Run this code, include the result with your PDF submission
train_data = ((training_data + 0.5) * 255)
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].astype(np.uint8)) # should␣
 ↪show 2 shoes from different pairs
```

```
(95, 3, 2, 224, 224, 3)
(285, 448, 224, 3)
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7f8d31a08850>
```

1.1.4 Part (d) -- 2%

Why do we insist that the different pairs of shoes still come from the same person? (Hint: what else do images from the same person have in common?)

Write your explanation here:

We want the model to differentiate between different shoes and not different backgrounds. In all pictures of the same student, the background is similar, while in different students, the background can change. If we randomize the second shoe from another student, the model can learn to differentiate between the background instead of the shoe itself.

1.1.5 Part (e) -- 2%

Why is it important that our data set be *balanced*? In other words suppose we created a data set where 99% of the images are of shoes that are *not* from the same pair, and 1% of the images are shoes that *are* from the same pair. Why could this be a problem?

Write your explanation here:

If we do not use balanced data, the model will learn to predict that the shoe is different all the time, so in 1% of the predictions, it will be wrong, and in 99% of the predictions, it will be correct, regardless of the picture itself.

In other words, we do not want the model to make predictions based on the quantity of the elements in each class.

1.2 Question 2. Convolutional Neural Networks (25%)

Before starting this question, we recommend reviewing the lecture and its associated example notebook on CNNs.

In this section, we will build two CNN models in PyTorch.

1.2.1 Part (a) -- 9%

Implement a CNN model in PyTorch called `CNN` that will take images of size $3 \times 448 \times 224$, and classify whether the images contain shoes from the same pair or from different pairs.

The model should contain the following layers:

- A convolution layer that takes in 3 channels, and outputs n channels.
- A 2×2 downsampling (either using a strided convolution in the previous step, or max pooling)
- A second convolution layer that takes in n channels, and outputs $2 \cdot n$ channels.
- A 2×2 downsampling (either using a strided convolution in the previous step, or max pooling)
- A third convolution layer that takes in $2 \cdot n$ channels, and outputs $4 \cdot n$ channels.
- A 2×2 downsampling (either using a strided convolution in the previous step, or max pooling)
- A fourth convolution layer that takes in $4 \cdot n$ channels, and outputs $8 \cdot n$ channels.
- A 2×2 downsampling (either using a strided convolution in the previous step, or max pooling)
- A fully-connected layer with 100 hidden units
- A fully-connected layer with 2 hidden units

Make the variable n a parameter of your CNN. You can use either 3×3 or 5×5 convolutions kernels. Set your padding to be `(kernel_size - 1) / 2` so that your feature maps have an even height/width.

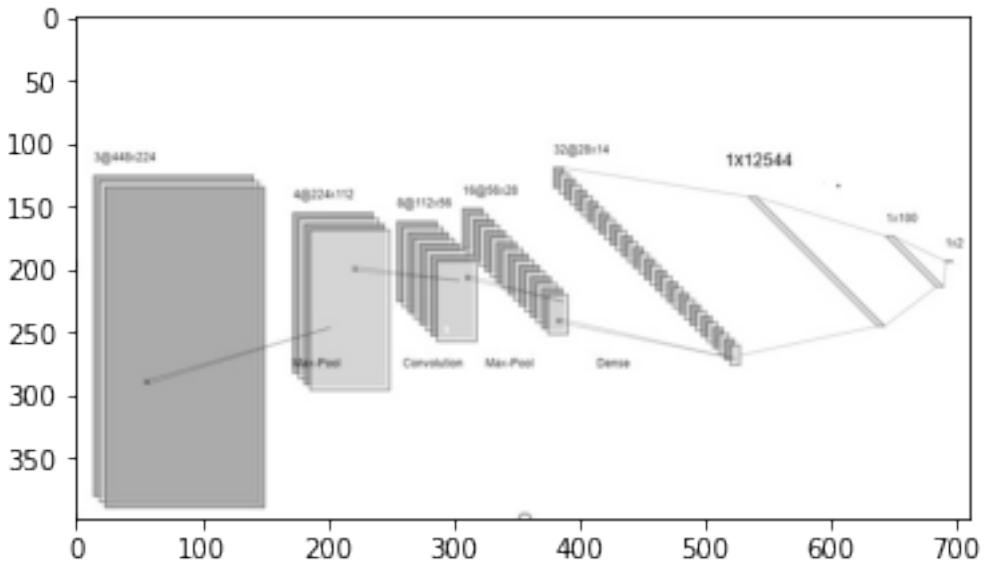
Note that we are omitting in our description certain steps that practitioners will typically not mention, like ReLU activations and reshaping operations. Use the example presented in class to figure out where they are.

[ ]:
```
import torch.nn.functional as F
class CNN(nn.Module):
    def __init__(self, n=4, kernel=5,):
        super(CNN, self).__init__()
        padding_size = int((kernel-1)/2)
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=n,␣
↪kernel_size=kernel, padding=padding_size)
        self.conv2 = nn.Conv2d(in_channels=n, out_channels=2*n,␣
↪kernel_size=kernel, padding=padding_size)
        self.conv3 = nn.Conv2d(in_channels=2*n, out_channels=4*n,␣
↪kernel_size=kernel, padding=padding_size)
        self.conv4 = nn.Conv2d(in_channels=4*n, out_channels=8*n,␣
↪kernel_size=kernel, padding=padding_size)

        self.fc1 = nn.Linear(8*n*14*28, 100)
        # The initial image is 448X224X3, we have reached this number by␣
↪knowing that when we choose the
        # padding size to be equal to (kernel-1)/2 then the act of convolution␣
↪with the padding
```

```
        # (as we defined above) does not change the size of the entry. So the␣
↪first layer the image size
        # is 4 times smaller (we chose kernel size = 2 for MAXPOOLING) so it␣
↪repeats 4 layers so we get
        # to the dimensions of 28X14 when the number of channels is 8n so the␣
↪total parameters
        #  if done flat is 8 * n * 14 * 28
        self.fc2 = nn.Linear(100, 2)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), kernel_size=2)
        x = F.max_pool2d(F.relu(self.conv2(x)), kernel_size=2)
        x = F.max_pool2d(F.relu(self.conv3(x)), kernel_size=2)
        x = F.max_pool2d(F.relu(self.conv4(x)), kernel_size=2)
        x = nn.Flatten()(x)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        return x
```

[ ]:
```
from google.colab import files
from io import BytesIO
from PIL import Image
import matplotlib.pyplot as plt

uploaded = files.upload()
im = Image.open(BytesIO(uploaded['CNN.jpeg']))

plt.imshow(im)
plt.show()
```

```
<IPython.core.display.HTML object>


Saving CNN.jpeg to CNN.jpeg
```



1.2.2 Part (b) -- 8%

Implement a CNN model in PyTorch called `CNNChannel` that contains the same layers as in the Part (a), but with one crucial difference: instead of starting with an image of shape $3 \times 448 \times 224$, we will first manipulate the image so that the left and right shoes images are concatenated along the **channel** dimension.

Complete the manipulation in the `forward()` method (by slicing and using the function `torch.cat`). The input to the first convolutional layer should have 6 channels instead of 3 (input shape $6 \times 224 \times 224$).

Use the same hyperparameter choices as you did in part (a), e.g. for the kernel size, choice of downsampling, and other choices.

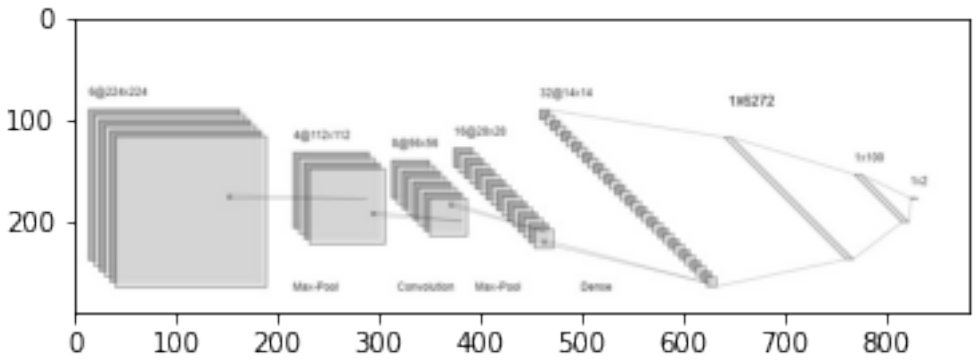
[ ]:
```
import torch.nn.functional as F
# The activation that we applied is ctified linear unit (RELU)
class CNNChannel(nn.Module):
    def __init__(self, n=4, kernel=5):
        super(CNNChannel, self).__init__()
        #remains the input size
        padding_size = int((kernel-1)/2)
        self.conv1 = nn.Conv2d(in_channels=6, out_channels=n,␣
↪kernel_size=kernel, padding=padding_size)
        self.conv2 = nn.Conv2d(in_channels=n, out_channels=2*n,␣
↪kernel_size=kernel, padding=padding_size)
        self.conv3 = nn.Conv2d(in_channels=2*n, out_channels=4*n,␣
↪kernel_size=kernel, padding=padding_size)
        self.conv4 = nn.Conv2d(in_channels=4*n, out_channels=8*n,␣
↪kernel_size=kernel, padding=padding_size)
        self.fc1 = nn.Linear(8*n*14*14, 100)
```

```
        # The initial image is 224X224X6, we have reached this number by␣
↪knowing that when we choose the
        # padding size to be equal to (kernel-1)/2 then the act of convolution␣
↪with the padding
        # (as we defined above) does not change the size of the entry. So the␣
↪first layer the image size
        # is 4 times smaller (we chose kernel size = 2 for MAXPOOLING) so it␣
↪repeats 4 layers so we get
        # to the dimensions of 14X14 when the number of channels is 8n so the␣
↪total parameters
        #  if done flat is 8 * n * 14 * 14
        self.fc2 = nn.Linear(100, 2)


    def forward(self, x):
        x = torch.cat((x[:,:,:,:224], x[:,:,:,224:]), 1) # because we doing for␣
↪our tensor the operation transpose(3, 1) the last channel is contain 448␣
↪that we need
        # to divide
        x = F.max_pool2d(F.relu(self.conv1(x)), kernel_size=2)
        x = F.max_pool2d(F.relu(self.conv2(x)), kernel_size=2)
        x = F.max_pool2d(F.relu(self.conv3(x)), kernel_size=2)
        x = F.max_pool2d(F.relu(self.conv4(x)), kernel_size=2)

        x = nn.Flatten()(x)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        return x
```

[ ]:
```
from google.colab import files
from io import BytesIO
from PIL import Image
import matplotlib.pyplot as plt

uploaded = files.upload()
im = Image.open(BytesIO(uploaded['CNNCHANNEL.jpeg']))

plt.imshow(im)
plt.show()
```

```
<IPython.core.display.HTML object>
```

```
Saving CNNCHANNEL.jpeg to CNNCHANNEL (1).jpeg
```



1.3 Part (c) -- 4%

The two models are quite similar, and should have almost the same number of parameters. However, one of these models will perform better, showing that architecture choices **do** matter in machine learning. Explain why one of these models performs better.

Write your explanation here:

The CNN model performs well in extracting local features from the data. In the first model (CNN), we are concatenating the 2 images in the same channel and therefore violating the principle of local features in the connection point of the two images. In the second model (CNNChannel), we split the two different pairs of shoes into two different channels. Therefore the CNN can extract equivalent local features from both of the shoes, and then in the fully connected layer, the model can decide if the features are matching.

1.4 Part (d) -- 4%

The function `get_accuracy` is written for you. You may need to modify this function depending on how you set up your model and training.

Unlike in the previous assignment, her we will separately compute the model accuracy on the positive and negative samples. Explain why we may wish to track the false positives and false negatives separately.

Write your explanation here:

We may wish to track the false positives and false negatives separately because the consequence of being wrong in one set is different from the other set. For example, being wrong in identifying if two pairs are the same when they are really the same is much less bad than being wrong and deciding that two pairs of the shoe are the same when they are not.

[ ]:
```
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)
        xs = xs.to(device)
        zs = model(xs)
        pred = zs.max(1, keepdim=True)[1] # get the index of the max logit
        pred = pred.detach().cpu().numpy() # we need to convert to numpy so we␣
↪add this line
        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)
        xs = xs.to(device)
        zs = model(xs)
        pred = zs.max(1, keepdim=True)[1] # get the index of the max logit
        pred = pred.detach().cpu().numpy() # we need to convert to numpy so we␣
↪add this line
        neg_correct += (pred == 0).sum()

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

1.5 Question 3. Training (40%)

Now, we will write the functions required to train the model.

Although our task is a binary classification problem, we will still use the architecture of a multi-class classification problem. That is, we'll use a one-hot vector to represent our target (like we did in the previous assignment). We'll also use `CrossEntropyLoss` instead of `BCEWithLogitsLoss` (this is a standard practice in machine learning because this architecture often performs better).

1.5.1 Part (a) -- 22%

Write the function `train_model` that takes in (as parameters) the model, training data, validation data, and other hyperparameters like the batch size, weight decay, etc. This function should be

somewhat similar to the training code that you wrote in Assignment 2, but with a major difference in the way we treat our training data.

Since our positive (shoes of the same pair) and negative (shoes of different pairs) training sets are separate, it is actually easier for us to generate separate minibatches of positive and negative training data. In each iteration, we'll take `batch_size / 2` positive samples and `batch_size / 2` negative samples. We will also generate labels of 1's for the positive samples, and 0's for the negative samples.

Here is what your training function should include:

- main training loop; choice of loss function; choice of optimizer
- obtaining the positive and negative samples
- shuffling the positive and negative samples at the start of each epoch
- in each iteration, take `batch_size / 2` positive samples and `batch_size / 2` negative samples as our input for this batch
- in each iteration, take `np.ones(batch_size / 2)` as the labels for the positive samples, and `np.zeros(batch_size / 2)` as the labels for the negative samples
- conversion from numpy arrays to PyTorch tensors, making sure that the input has dimensions $N \times C \times H \times W$ (known as NCHW tensor), where N is the number of images batch size, C is the number of channels, H is the height of the image, and W is the width of the image.
- computing the forward and backward passes
- after every epoch, report the accuracies for the training set and validation set
- track the training curve information and plot the training curve

It is also recommended to checkpoint your model (save a copy) after every epoch, as we did in Assignment 2.

[ ]:
```
# Write your code here
from sklearn.utils import shuffle

def train_model(model,
                training_data,
                validation_data,
                batch_size=100,
                learning_rate=0.001,
                weight_decay=0,
                epochs=5,
                checkpoint_path='/content/gdrive/My Drive/Intro_to_Deep_Learning/
 ↪ckpt-{}.pk',
                save_best_model=False
                ):
  criterion = nn.CrossEntropyLoss()
  optimizer = optim.Adam(model.parameters(),
                         lr=learning_rate,
                         weight_decay=weight_decay)
  if (save_best_model):
    best_epoch, best_pos_and_neg_avg_acc, best_neg_acc, best_pos_acc = 0, 0, 0,␣
 ↪0 #initilize variables for saving best model
```

```
 losses, train_accs_pos, val_accs_pos, train_accs_neg, val_accs_neg  = [], [],␣
↪[] ,[], []
 class_batch_size = int(batch_size/2)

 #create same pairs and different pair datasets
 train_data_pos = generate_same_pair(training_data)
 train_data_neg = generate_different_pair(training_data)

 for epoch in range(epochs):
   running_loss = 0
   #shuffling the positive and negative samples at the start of each epoch
   train_data_pos = shuffle(train_data_pos)
   train_data_neg = shuffle(train_data_neg)

   for i in range(0, len(train_data_pos), class_batch_size):
     if (i + class_batch_size) > len(train_data_pos):
           break

     # get the input and targets of a minibatch -create then as tensors
     xt_pos = torch.Tensor(train_data_pos[i:i+(class_batch_size)]).
↪transpose(1, 3)
     st_pos = torch.ones((class_batch_size,), dtype=torch.long)
     xt_neg = torch.Tensor(train_data_neg[i:i+(class_batch_size)]).
↪transpose(1, 3)
     st_neg = torch.zeros((class_batch_size,), dtype=torch.long)
     xt = torch.cat((xt_pos, xt_neg), 0)
     st = torch.cat((st_pos, st_neg), 0)

     # send to GPU device
     xt, st = xt.to(device), st.to(device)

     # compute prediction logit
     zs = model(xt)
     # compute the total loss
     loss = criterion(zs, st)
     # zero the gradients before we calc our gradients is a clean up step
     # for PyTorch
     optimizer.zero_grad()
     # backward pass to compute the gradient of loss with respect to our
     # learnable params
     loss.backward()
     # make the updates for each parameter with our optimizer that we
     # define earlier (Adam)
     optimizer.step()

     running_loss += loss
```

```
    # save the current training information
    num_of_training_samples = 2*i if (i + class_batch_size) >␣
↪len(train_data_pos) else 2*len(train_data_pos)
    losses.append(float(running_loss)/num_of_training_samples)  # compute␣
↪*average* loss of epoch
    epoch_train_loss = float(running_loss.detach().cpu().numpy())/
↪num_of_training_samples
    train_acc_pos, train_acc_neg = get_accuracy(model, training_data,␣
↪batch_size)
    train_accs_pos.append(train_acc_pos)
    train_accs_neg.append(train_acc_neg)
    val_acc_pos, val_acc_neg = get_accuracy(model, validation_data, batch_size)
    val_accs_pos.append(val_acc_pos)
    val_accs_neg.append(val_acc_neg)
    print("epoch %d. [Val Acc (all %.0f%%), (positives %.0f%%), (negatives: %.
↪0f%%)] [Train Acc (all %.0f%%), (positive: %.0f%%), (negatives: %.0f%%),␣
↪Loss %f]" % (
          epoch, ((val_acc_pos+val_acc_neg)/2) * 100, val_acc_pos * 100,␣
↪val_acc_neg * 100, ((train_acc_pos+train_acc_neg)/2) * 100,  train_acc_pos *␣
↪100, train_acc_neg * 100, epoch_train_loss))

    #save best model based of validation - if avarge is higher
    if (save_best_model):
      if ((val_acc_pos+val_acc_neg)/2 > best_pos_and_neg_avg_acc):
        print("saving epoch", epoch, "as best epoch")
        best_pos_and_neg_avg_acc = (val_acc_pos+val_acc_neg)/2
        best_pos_acc, best_neg_acc = val_acc_pos, val_acc_neg
        best_epoch = epoch
        torch.save(model.state_dict(), checkpoint_path.format(epoch))
      elif ((val_acc_pos+val_acc_neg)/2 == best_pos_and_neg_avg_acc):
        if (abs(val_acc_pos-val_acc_neg) < abs(best_pos_acc-best_neg_acc)):
          print("saving epoch", epoch, "as best epoch")
          best_pos_acc, best_neg_acc = val_acc_pos, val_acc_neg
          best_epoch = epoch
          torch.save(model.state_dict(), checkpoint_path.format(epoch))

  return epochs, losses, train_accs_pos, val_accs_pos, train_accs_neg,␣
↪val_accs_neg


def plot_learning_curve(epochs, losses, train_accs_pos=None, val_accs_pos=None,␣
↪train_accs_neg=None, val_accs_neg=None, average=False):
    """
    Plot the learning curve.
    """
```

```
    plt.title("Learning Curve: Loss per epoch")
    plt.plot(range(epochs), losses, label="Train")
    plt.xlabel("epochs")
    plt.ylabel("Loss")
    plt.show()

    plt.title("Learning Curve: Accuracy per epoch")
    if (average==False):
      if train_accs_pos is not None:
        plt.plot(range(epochs), train_accs_pos, label="Train positives")
      if val_accs_pos is not None:
        plt.plot(range(epochs), val_accs_pos, label="Validation positives")
      if train_accs_neg is not None:
        plt.plot(range(epochs), train_accs_neg, label="Train negatives")
      if val_accs_neg is not None:
        plt.plot(range(epochs), val_accs_neg, label="Validation negatives")
    else:
        if(train_accs_pos is not None):
          train_accs_pos ,train_accs_neg = np.array(train_accs_pos), np.
↪array(train_accs_neg)
          plt.plot(range(epochs), (train_accs_pos+train_accs_neg)/2,␣
↪label="Train average of postives and negatives")
        if(val_accs_pos is not None):
          val_accs_pos, val_accs_neg = np.array(val_accs_pos), np.
↪array(val_accs_neg)
          plt.plot(range(epochs), (val_accs_pos+val_accs_neg)/2,␣
↪label="Validation average of postives and negatives")
    plt.xlabel("epochs")
    plt.ylabel("Accuracy")
    plt.legend(loc='best')
    plt.show()
```

1.5.2 Part (b) -- 6%

Sanity check your code from Q3(a) and from Q2(a) and Q2(b) by showing that your models can memorize a very small subset of the training set (e.g. 5 images). You should be able to achieve 90%+ accuracy (don't forget to calculate the accuracy) relatively quickly (within ~30 or so iterations).

(Start with the second network, it is easier to converge)

Try to find the general parameters combination that work for each network, it can help you a little bit later.

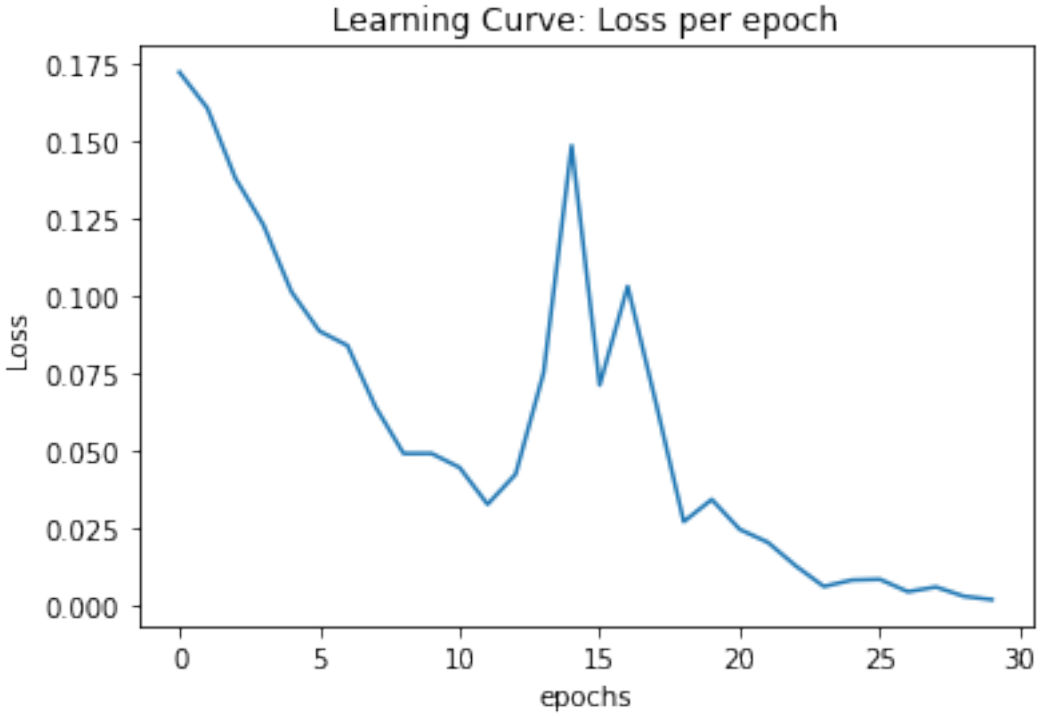
CNNChannel model:

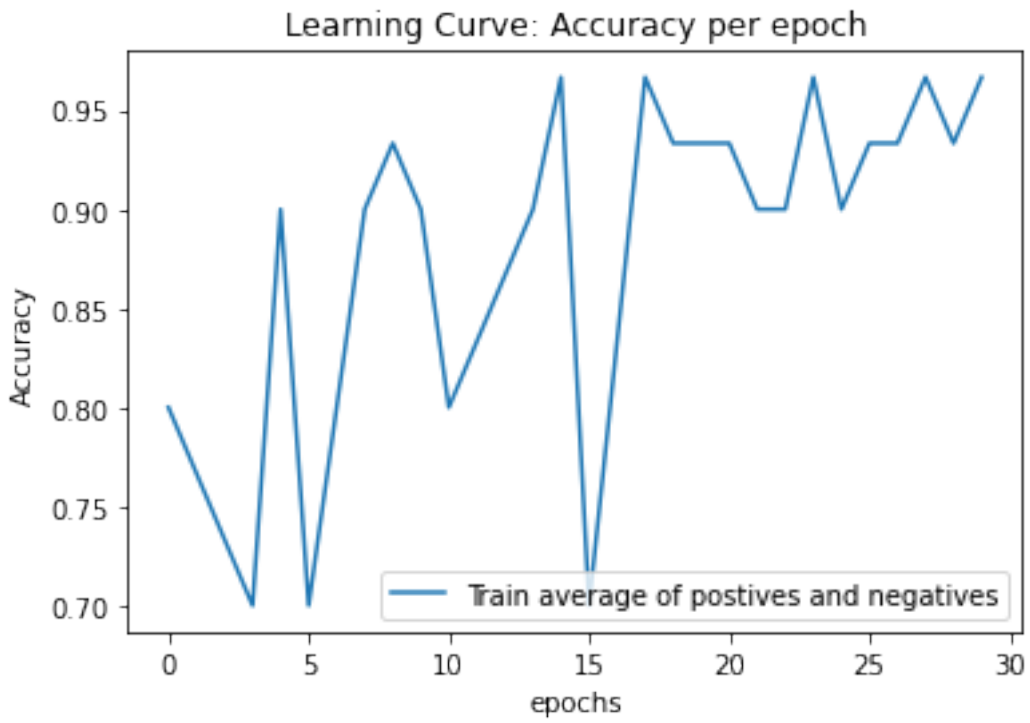
[ ]:
```
# Write your code here. Remember to include your results so that we can
# see that your model attains a high training accuracy.
model_CNNChannel = CNNChannel()
```

```
epochs, losses, train_accs_pos, val_accs_pos, train_accs_neg, val_accs_neg =␣
 ↪train_model(model_CNNChannel, training_data=training_data[:5],␣
 ↪validation_data=validation_data, batch_size=4, learning_rate=0.0005,␣
 ↪weight_decay=0, epochs=30)
plot_learning_curve(epochs, losses, train_accs_pos=train_accs_pos,␣
 ↪val_accs_pos=None, train_accs_neg=train_accs_neg, val_accs_neg=None,␣
 ↪average=True)
```

```
epoch 0. [Val Acc (all 57%), (positives 59%), (negatives: 55%)] [Train Acc (all
80%), (positive: 93%), (negatives: 67%), Loss 0.172312]
epoch 1. [Val Acc (all 57%), (positives 37%), (negatives: 76%)] [Train Acc (all
77%), (positive: 73%), (negatives: 80%), Loss 0.160668]
epoch 2. [Val Acc (all 61%), (positives 53%), (negatives: 69%)] [Train Acc (all
73%), (positive: 87%), (negatives: 60%), Loss 0.138064]
epoch 3. [Val Acc (all 69%), (positives 86%), (negatives: 51%)] [Train Acc (all
70%), (positive: 100%), (negatives: 40%), Loss 0.122999]
epoch 4. [Val Acc (all 64%), (positives 41%), (negatives: 86%)] [Train Acc (all
90%), (positive: 87%), (negatives: 93%), Loss 0.101404]
epoch 5. [Val Acc (all 70%), (positives 86%), (negatives: 53%)] [Train Acc (all
70%), (positive: 100%), (negatives: 40%), Loss 0.088701]
epoch 6. [Val Acc (all 72%), (positives 51%), (negatives: 92%)] [Train Acc (all
80%), (positive: 73%), (negatives: 87%), Loss 0.084048]
epoch 7. [Val Acc (all 71%), (positives 78%), (negatives: 63%)] [Train Acc (all
90%), (positive: 100%), (negatives: 80%), Loss 0.064273]
epoch 8. [Val Acc (all 74%), (positives 61%), (negatives: 86%)] [Train Acc (all
93%), (positive: 93%), (negatives: 93%), Loss 0.049202]
epoch 9. [Val Acc (all 72%), (positives 55%), (negatives: 88%)] [Train Acc (all
90%), (positive: 87%), (negatives: 93%), Loss 0.049217]
epoch 10. [Val Acc (all 74%), (positives 98%), (negatives: 49%)] [Train Acc (all
80%), (positive: 100%), (negatives: 60%), Loss 0.044691]
epoch 11. [Val Acc (all 73%), (positives 59%), (negatives: 86%)] [Train Acc (all
83%), (positive: 80%), (negatives: 87%), Loss 0.032799]
epoch 12. [Val Acc (all 78%), (positives 96%), (negatives: 61%)] [Train Acc (all
87%), (positive: 100%), (negatives: 73%), Loss 0.042525]
epoch 13. [Val Acc (all 75%), (positives 71%), (negatives: 80%)] [Train Acc (all
90%), (positive: 100%), (negatives: 80%), Loss 0.075292]
epoch 14. [Val Acc (all 75%), (positives 69%), (negatives: 82%)] [Train Acc (all
97%), (positive: 93%), (negatives: 100%), Loss 0.148591]
epoch 15. [Val Acc (all 65%), (positives 100%), (negatives: 29%)] [Train Acc
(all 70%), (positive: 100%), (negatives: 40%), Loss 0.071296]
epoch 16. [Val Acc (all 75%), (positives 67%), (negatives: 82%)] [Train Acc (all
83%), (positive: 73%), (negatives: 93%), Loss 0.103158]
epoch 17. [Val Acc (all 75%), (positives 75%), (negatives: 76%)] [Train Acc (all
97%), (positive: 100%), (negatives: 93%), Loss 0.065837]
epoch 18. [Val Acc (all 78%), (positives 94%), (negatives: 63%)] [Train Acc (all
93%), (positive: 100%), (negatives: 87%), Loss 0.027195]
epoch 19. [Val Acc (all 77%), (positives 84%), (negatives: 71%)] [Train Acc (all
```

```
93%), (positive: 100%), (negatives: 87%), Loss 0.034362]
epoch 20. [Val Acc (all 75%), (positives 76%), (negatives: 75%)] [Train Acc (all
93%), (positive: 100%), (negatives: 87%), Loss 0.024699]
epoch 21. [Val Acc (all 77%), (positives 84%), (negatives: 71%)] [Train Acc (all
90%), (positive: 100%), (negatives: 80%), Loss 0.020540]
epoch 22. [Val Acc (all 75%), (positives 82%), (negatives: 69%)] [Train Acc (all
90%), (positive: 100%), (negatives: 80%), Loss 0.012916]
epoch 23. [Val Acc (all 79%), (positives 84%), (negatives: 75%)] [Train Acc (all
97%), (positive: 100%), (negatives: 93%), Loss 0.006282]
epoch 24. [Val Acc (all 78%), (positives 86%), (negatives: 71%)] [Train Acc (all
90%), (positive: 100%), (negatives: 80%), Loss 0.008348]
epoch 25. [Val Acc (all 76%), (positives 86%), (negatives: 67%)] [Train Acc (all
93%), (positive: 100%), (negatives: 87%), Loss 0.008595]
epoch 26. [Val Acc (all 77%), (positives 80%), (negatives: 75%)] [Train Acc (all
93%), (positive: 100%), (negatives: 87%), Loss 0.004593]
epoch 27. [Val Acc (all 77%), (positives 84%), (negatives: 71%)] [Train Acc (all
97%), (positive: 100%), (negatives: 93%), Loss 0.006121]
epoch 28. [Val Acc (all 76%), (positives 84%), (negatives: 69%)] [Train Acc (all
93%), (positive: 100%), (negatives: 87%), Loss 0.003109]
epoch 29. [Val Acc (all 76%), (positives 84%), (negatives: 69%)] [Train Acc (all
97%), (positive: 100%), (negatives: 93%), Loss 0.002029]
```





CNN model:

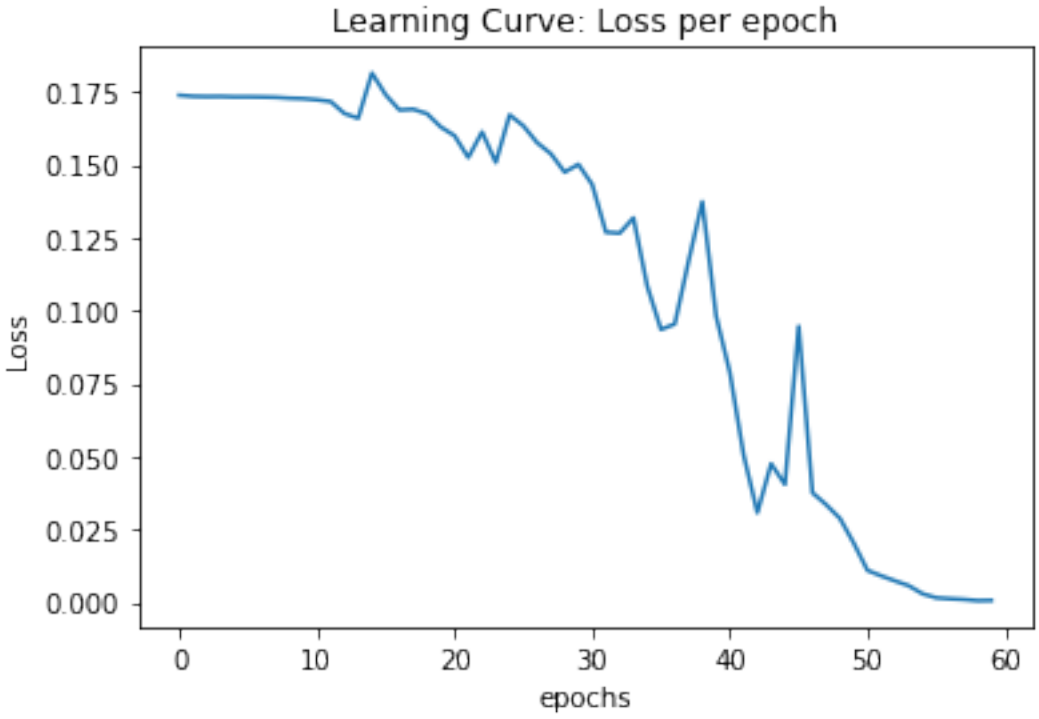
[ ]:
```
# Write your code here. Remember to include your results so that we can
# see that your model attains a high training accuracy.

model_CNN = CNN().to(device)
epochs, losses, train_accs_pos, val_accs_pos, train_accs_neg, val_accs_neg =␣
 ↪train_model(model_CNN, training_data=training_data[:5],␣
 ↪validation_data=validation_data, batch_size=5, learning_rate=0.0005,␣
 ↪weight_decay=0, epochs=60)
plot_learning_curve(epochs, losses, train_accs_pos=train_accs_pos,␣
 ↪val_accs_pos=None, train_accs_neg=train_accs_neg, val_accs_neg=None,␣
 ↪average=True)
```

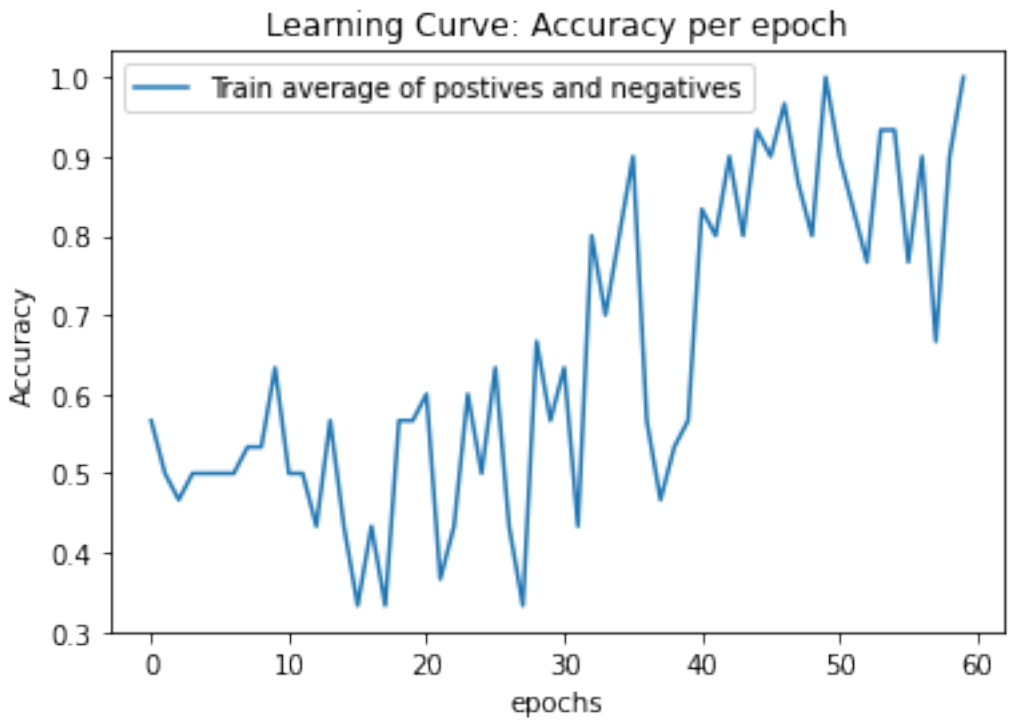
```
epoch 0. [Val Acc (all 50%), (positives 4%), (negatives: 96%)] [Train Acc (all
57%), (positive: 13%), (negatives: 100%), Loss 0.173762]
epoch 1. [Val Acc (all 50%), (positives 0%), (negatives: 100%)] [Train Acc (all
50%), (positive: 0%), (negatives: 100%), Loss 0.173388]
epoch 2. [Val Acc (all 48%), (positives 96%), (negatives: 0%)] [Train Acc (all
47%), (positive: 93%), (negatives: 0%), Loss 0.173300]
epoch 3. [Val Acc (all 50%), (positives 100%), (negatives: 0%)] [Train Acc (all
50%), (positive: 100%), (negatives: 0%), Loss 0.173340]
epoch 4. [Val Acc (all 50%), (positives 0%), (negatives: 100%)] [Train Acc (all
50%), (positive: 0%), (negatives: 100%), Loss 0.173207]
epoch 5. [Val Acc (all 50%), (positives 100%), (negatives: 0%)] [Train Acc (all
```

```
50%), (positive: 100%), (negatives: 0%), Loss 0.173214]
epoch 6. [Val Acc (all 48%), (positives 82%), (negatives: 14%)] [Train Acc (all
50%), (positive: 60%), (negatives: 40%), Loss 0.173150]
epoch 7. [Val Acc (all 51%), (positives 18%), (negatives: 84%)] [Train Acc (all
53%), (positive: 33%), (negatives: 73%), Loss 0.173047]
epoch 8. [Val Acc (all 44%), (positives 88%), (negatives: 0%)] [Train Acc (all
53%), (positive: 87%), (negatives: 20%), Loss 0.172744]
epoch 9. [Val Acc (all 51%), (positives 33%), (negatives: 69%)] [Train Acc (all
63%), (positive: 33%), (negatives: 93%), Loss 0.172602]
epoch 10. [Val Acc (all 47%), (positives 14%), (negatives: 80%)] [Train Acc (all
50%), (positive: 33%), (negatives: 67%), Loss 0.172214]
epoch 11. [Val Acc (all 48%), (positives 53%), (negatives: 43%)] [Train Acc (all
50%), (positive: 60%), (negatives: 40%), Loss 0.171608]
epoch 12. [Val Acc (all 52%), (positives 76%), (negatives: 27%)] [Train Acc (all
43%), (positive: 67%), (negatives: 20%), Loss 0.167501]
epoch 13. [Val Acc (all 47%), (positives 45%), (negatives: 49%)] [Train Acc (all
57%), (positive: 47%), (negatives: 67%), Loss 0.165872]
epoch 14. [Val Acc (all 49%), (positives 22%), (negatives: 76%)] [Train Acc (all
43%), (positive: 40%), (negatives: 47%), Loss 0.181359]
epoch 15. [Val Acc (all 55%), (positives 65%), (negatives: 45%)] [Train Acc (all
33%), (positive: 60%), (negatives: 7%), Loss 0.173901]
epoch 16. [Val Acc (all 51%), (positives 96%), (negatives: 6%)] [Train Acc (all
43%), (positive: 87%), (negatives: 0%), Loss 0.168705]
epoch 17. [Val Acc (all 53%), (positives 69%), (negatives: 37%)] [Train Acc (all
33%), (positive: 67%), (negatives: 0%), Loss 0.168958]
epoch 18. [Val Acc (all 46%), (positives 49%), (negatives: 43%)] [Train Acc (all
57%), (positive: 53%), (negatives: 60%), Loss 0.167435]
epoch 19. [Val Acc (all 52%), (positives 53%), (negatives: 51%)] [Train Acc (all
57%), (positive: 53%), (negatives: 60%), Loss 0.162889]
epoch 20. [Val Acc (all 45%), (positives 61%), (negatives: 29%)] [Train Acc (all
60%), (positive: 60%), (negatives: 60%), Loss 0.159903]
epoch 21. [Val Acc (all 50%), (positives 57%), (negatives: 43%)] [Train Acc (all
37%), (positive: 60%), (negatives: 13%), Loss 0.152405]
epoch 22. [Val Acc (all 48%), (positives 78%), (negatives: 18%)] [Train Acc (all
43%), (positive: 87%), (negatives: 0%), Loss 0.161175]
epoch 23. [Val Acc (all 49%), (positives 12%), (negatives: 86%)] [Train Acc (all
60%), (positive: 27%), (negatives: 93%), Loss 0.150725]
epoch 24. [Val Acc (all 50%), (positives 90%), (negatives: 10%)] [Train Acc (all
50%), (positive: 87%), (negatives: 13%), Loss 0.167031]
epoch 25. [Val Acc (all 51%), (positives 90%), (negatives: 12%)] [Train Acc (all
63%), (positive: 87%), (negatives: 40%), Loss 0.163280]
epoch 26. [Val Acc (all 47%), (positives 53%), (negatives: 41%)] [Train Acc (all
43%), (positive: 40%), (negatives: 47%), Loss 0.157502]
epoch 27. [Val Acc (all 51%), (positives 51%), (negatives: 51%)] [Train Acc (all
33%), (positive: 40%), (negatives: 27%), Loss 0.153714]
epoch 28. [Val Acc (all 55%), (positives 75%), (negatives: 35%)] [Train Acc (all
67%), (positive: 60%), (negatives: 73%), Loss 0.147401]
epoch 29. [Val Acc (all 59%), (positives 78%), (negatives: 39%)] [Train Acc (all
```

```
57%), (positive: 53%), (negatives: 60%), Loss 0.150067]
epoch 30. [Val Acc (all 53%), (positives 69%), (negatives: 37%)] [Train Acc (all
63%), (positive: 60%), (negatives: 67%), Loss 0.143182]
epoch 31. [Val Acc (all 57%), (positives 73%), (negatives: 41%)] [Train Acc (all
43%), (positive: 53%), (negatives: 33%), Loss 0.126824]
epoch 32. [Val Acc (all 54%), (positives 75%), (negatives: 33%)] [Train Acc (all
80%), (positive: 73%), (negatives: 87%), Loss 0.126494]
epoch 33. [Val Acc (all 53%), (positives 80%), (negatives: 25%)] [Train Acc (all
70%), (positive: 100%), (negatives: 40%), Loss 0.131727]
epoch 34. [Val Acc (all 57%), (positives 67%), (negatives: 47%)] [Train Acc (all
80%), (positive: 60%), (negatives: 100%), Loss 0.108360]
epoch 35. [Val Acc (all 54%), (positives 67%), (negatives: 41%)] [Train Acc (all
90%), (positive: 80%), (negatives: 100%), Loss 0.093490]
epoch 36. [Val Acc (all 53%), (positives 96%), (negatives: 10%)] [Train Acc (all
57%), (positive: 100%), (negatives: 13%), Loss 0.095349]
epoch 37. [Val Acc (all 55%), (positives 39%), (negatives: 71%)] [Train Acc (all
47%), (positive: 47%), (negatives: 47%), Loss 0.116871]
epoch 38. [Val Acc (all 59%), (positives 88%), (negatives: 29%)] [Train Acc (all
53%), (positive: 87%), (negatives: 20%), Loss 0.137286]
epoch 39. [Val Acc (all 61%), (positives 94%), (negatives: 27%)] [Train Acc (all
57%), (positive: 100%), (negatives: 13%), Loss 0.098197]
epoch 40. [Val Acc (all 67%), (positives 84%), (negatives: 49%)] [Train Acc (all
83%), (positive: 93%), (negatives: 73%), Loss 0.078825]
epoch 41. [Val Acc (all 65%), (positives 92%), (negatives: 37%)] [Train Acc (all
80%), (positive: 100%), (negatives: 60%), Loss 0.050567]
epoch 42. [Val Acc (all 69%), (positives 75%), (negatives: 63%)] [Train Acc (all
90%), (positive: 87%), (negatives: 93%), Loss 0.030721]
epoch 43. [Val Acc (all 62%), (positives 98%), (negatives: 25%)] [Train Acc (all
80%), (positive: 100%), (negatives: 60%), Loss 0.047473]
epoch 44. [Val Acc (all 65%), (positives 92%), (negatives: 37%)] [Train Acc (all
93%), (positive: 87%), (negatives: 100%), Loss 0.040412]
epoch 45. [Val Acc (all 67%), (positives 100%), (negatives: 33%)] [Train Acc
(all 90%), (positive: 100%), (negatives: 80%), Loss 0.094606]
epoch 46. [Val Acc (all 65%), (positives 96%), (negatives: 33%)] [Train Acc (all
97%), (positive: 100%), (negatives: 93%), Loss 0.037545]
epoch 47. [Val Acc (all 68%), (positives 96%), (negatives: 39%)] [Train Acc (all
87%), (positive: 100%), (negatives: 73%), Loss 0.033450]
epoch 48. [Val Acc (all 62%), (positives 98%), (negatives: 25%)] [Train Acc (all
80%), (positive: 93%), (negatives: 67%), Loss 0.028743]
epoch 49. [Val Acc (all 67%), (positives 98%), (negatives: 35%)] [Train Acc (all
100%), (positive: 100%), (negatives: 100%), Loss 0.020307]
epoch 50. [Val Acc (all 64%), (positives 98%), (negatives: 29%)] [Train Acc (all
90%), (positive: 100%), (negatives: 80%), Loss 0.010927]
epoch 51. [Val Acc (all 69%), (positives 98%), (negatives: 39%)] [Train Acc (all
83%), (positive: 100%), (negatives: 67%), Loss 0.009054]
epoch 52. [Val Acc (all 68%), (positives 98%), (negatives: 37%)] [Train Acc (all
77%), (positive: 100%), (negatives: 53%), Loss 0.007339]
epoch 53. [Val Acc (all 68%), (positives 98%), (negatives: 37%)] [Train Acc (all
```

```
93%), (positive: 100%), (negatives: 87%), Loss 0.005682]
epoch 54. [Val Acc (all 65%), (positives 100%), (negatives: 29%)] [Train Acc
(all 93%), (positive: 100%), (negatives: 87%), Loss 0.002910]
epoch 55. [Val Acc (all 69%), (positives 100%), (negatives: 37%)] [Train Acc
(all 77%), (positive: 100%), (negatives: 53%), Loss 0.001538]
epoch 56. [Val Acc (all 67%), (positives 100%), (negatives: 33%)] [Train Acc
(all 90%), (positive: 100%), (negatives: 80%), Loss 0.001215]
epoch 57. [Val Acc (all 69%), (positives 100%), (negatives: 37%)] [Train Acc
(all 67%), (positive: 100%), (negatives: 33%), Loss 0.000972]
epoch 58. [Val Acc (all 67%), (positives 100%), (negatives: 33%)] [Train Acc
(all 90%), (positive: 100%), (negatives: 80%), Loss 0.000562]
epoch 59. [Val Acc (all 65%), (positives 100%), (negatives: 29%)] [Train Acc
(all 100%), (positive: 100%), (negatives: 100%), Loss 0.000631]
```





1.5.3 Part (c) -- 8%

Train your models from Q2(a) and Q2(b). Change the values of a few hyperparameters, including the learning rate, batch size, choice of n, and the kernel size. You do not need to check all values for all hyperparameters. Instead, try to make big changes to see how each change affect your scores. (try to start with finding a resonable learning rate for each network, that start changing the other parameters, the first network might need bigger n and kernel size)

In this section, explain how you tuned your hyperparameters.

Write your explanation here:

Note: We found the learning process to be, in some cases, very stochastic, and we made our decisions based on our observations. Therefore, there might be better hyperparameters.

Note that we chose our best weights based on the best average accuracy of the positive and negative validation samples.

In addition we saved the best epoch weights so we could run on a lot of ephocs since we all the time took the best results and not the last results.

For the CNN model:

We started by changing the learing rate from high to low values: [0.01, 0.001, 0.0001, 0.0005, 0.00005, 0.00001, 0.000001]) We noticed that the learning curve for this model is kind of stable, So we decided to take the small value 0.00001. For the same reason, we add weight_decay and tried different combinations from [5e-2, 5e-4, 5e-7]). We noticed that for high values there is not learning at all (loss is not decreasing). So for summerizing, we found the combination of weight_decay=5e-7 and learning rate from [0.00005, 0.00001] to be best.

After that, we tried different batch sizes [16,32,64,128], we assumed that bigger batch size will estimate the predicted function better, and found that as we increase the size, the convergence of

the validation accuracy is much faster, and as a result, we chose batch_size=128, which is relativly high value.

In the last stage, we tried to increase the n value [4, 6, 8, 10] and found n=10 to have the best results.

Due to the low learning rate, we choose the number of epochs to be 200; however, we got the best model in epoch 119

For the CNN Channel model:

The hyperparameters tunning for this model was easier compare to CNN model, since the learning process was much more stable. We started with the hyperparameters we found in the CNN model.

Because the learning of this model is much more stable, we tried to increase the learning rate and found that, indeed, convergence of the accuracy is much better. In the first stage, We increased the learning rate from 0.00001 to 0.0001 and we converged faster and with better numbers.

After that, we Reduced the batch size from 128 to 64 and this increased the accuracy. In addition, we tried to disable the weight_decay but it yeld lower results so we remained it 5e-7.

In the last stage, we decided to try several n values by taking values from: [6, 8, 10, 12, 16, 20, 24, 28, 32], we found that n=24 is brings the best accuracy. After this stage.

Here the convergence was much faster so we took 50 epochs but we got 92 percent after 33 epochs.

[ ]:
```
# Include the training curves for the two models.
model_CNN = CNN(n=10).to(device)
learning_curve_info = train_model(model_CNN, training_data=training_data,␣
↪validation_data=validation_data, batch_size=128, learning_rate=0.00001,␣
↪weight_decay=5e-7, epochs=200, checkpoint_path='/content/gdrive/My Drive/
↪CNN_1_ckpt-{}.pk', save_best_model=True)
```

```
epoch 0. [Val Acc (all 51%), (positives 2%), (negatives: 100%)] [Train Acc (all
50%), (positive: 2%), (negatives: 99%), Loss 0.005416]
saving epoch 0 as best epoch
epoch 1. [Val Acc (all 49%), (positives 98%), (negatives: 0%)] [Train Acc (all
50%), (positive: 99%), (negatives: 0%), Loss 0.005415]
epoch 2. [Val Acc (all 50%), (positives 100%), (negatives: 0%)] [Train Acc (all
50%), (positive: 100%), (negatives: 0%), Loss 0.005415]
epoch 3. [Val Acc (all 49%), (positives 98%), (negatives: 0%)] [Train Acc (all
50%), (positive: 100%), (negatives: 0%), Loss 0.005416]
epoch 4. [Val Acc (all 46%), (positives 78%), (negatives: 14%)] [Train Acc (all
50%), (positive: 69%), (negatives: 31%), Loss 0.005415]
epoch 5. [Val Acc (all 51%), (positives 22%), (negatives: 80%)] [Train Acc (all
51%), (positive: 15%), (negatives: 87%), Loss 0.005415]
saving epoch 5 as best epoch
epoch 6. [Val Acc (all 50%), (positives 6%), (negatives: 94%)] [Train Acc (all
52%), (positive: 6%), (negatives: 97%), Loss 0.005415]
epoch 7. [Val Acc (all 50%), (positives 2%), (negatives: 98%)] [Train Acc (all
51%), (positive: 3%), (negatives: 98%), Loss 0.005415]
epoch 8. [Val Acc (all 49%), (positives 2%), (negatives: 96%)] [Train Acc (all
51%), (positive: 3%), (negatives: 99%), Loss 0.005415]
epoch 9. [Val Acc (all 51%), (positives 10%), (negatives: 92%)] [Train Acc (all
```

```
52%), (positive: 8%), (negatives: 96%), Loss 0.005415]
epoch 10. [Val Acc (all 53%), (positives 31%), (negatives: 75%)] [Train Acc (all
51%), (positive: 23%), (negatives: 80%), Loss 0.005414]
saving epoch 10 as best epoch
epoch 11. [Val Acc (all 53%), (positives 49%), (negatives: 57%)] [Train Acc (all
52%), (positive: 38%), (negatives: 67%), Loss 0.005415]
saving epoch 11 as best epoch
epoch 12. [Val Acc (all 52%), (positives 71%), (negatives: 33%)] [Train Acc (all
54%), (positive: 60%), (negatives: 48%), Loss 0.005414]
epoch 13. [Val Acc (all 47%), (positives 75%), (negatives: 20%)] [Train Acc (all
51%), (positive: 69%), (negatives: 32%), Loss 0.005414]
epoch 14. [Val Acc (all 51%), (positives 86%), (negatives: 16%)] [Train Acc (all
52%), (positive: 76%), (negatives: 28%), Loss 0.005414]
epoch 15. [Val Acc (all 56%), (positives 69%), (negatives: 43%)] [Train Acc (all
53%), (positive: 59%), (negatives: 47%), Loss 0.005414]
saving epoch 15 as best epoch
epoch 16. [Val Acc (all 57%), (positives 61%), (negatives: 53%)] [Train Acc (all
55%), (positive: 46%), (negatives: 64%), Loss 0.005414]
saving epoch 16 as best epoch
epoch 17. [Val Acc (all 59%), (positives 47%), (negatives: 71%)] [Train Acc (all
55%), (positive: 35%), (negatives: 75%), Loss 0.005414]
saving epoch 17 as best epoch
epoch 18. [Val Acc (all 54%), (positives 29%), (negatives: 78%)] [Train Acc (all
54%), (positive: 20%), (negatives: 88%), Loss 0.005413]
epoch 19. [Val Acc (all 57%), (positives 31%), (negatives: 82%)] [Train Acc (all
55%), (positive: 21%), (negatives: 88%), Loss 0.005413]
epoch 20. [Val Acc (all 57%), (positives 35%), (negatives: 78%)] [Train Acc (all
57%), (positive: 26%), (negatives: 88%), Loss 0.005413]
epoch 21. [Val Acc (all 63%), (positives 53%), (negatives: 73%)] [Train Acc (all
58%), (positive: 36%), (negatives: 79%), Loss 0.005412]
saving epoch 21 as best epoch
epoch 22. [Val Acc (all 61%), (positives 71%), (negatives: 51%)] [Train Acc (all
60%), (positive: 61%), (negatives: 60%), Loss 0.005412]
epoch 23. [Val Acc (all 62%), (positives 86%), (negatives: 37%)] [Train Acc (all
61%), (positive: 79%), (negatives: 43%), Loss 0.005412]
epoch 24. [Val Acc (all 62%), (positives 86%), (negatives: 37%)] [Train Acc (all
61%), (positive: 84%), (negatives: 38%), Loss 0.005411]
epoch 25. [Val Acc (all 63%), (positives 86%), (negatives: 39%)] [Train Acc (all
62%), (positive: 80%), (negatives: 44%), Loss 0.005411]
epoch 26. [Val Acc (all 63%), (positives 73%), (negatives: 53%)] [Train Acc (all
63%), (positive: 70%), (negatives: 56%), Loss 0.005410]
epoch 27. [Val Acc (all 55%), (positives 69%), (negatives: 41%)] [Train Acc (all
64%), (positive: 64%), (negatives: 65%), Loss 0.005409]
epoch 28. [Val Acc (all 59%), (positives 86%), (negatives: 31%)] [Train Acc (all
64%), (positive: 84%), (negatives: 43%), Loss 0.005409]
epoch 29. [Val Acc (all 65%), (positives 73%), (negatives: 57%)] [Train Acc (all
66%), (positive: 69%), (negatives: 64%), Loss 0.005408]
saving epoch 29 as best epoch
```

```
epoch 30. [Val Acc (all 64%), (positives 73%), (negatives: 55%)] [Train Acc (all
64%), (positive: 67%), (negatives: 62%), Loss 0.005407]
epoch 31. [Val Acc (all 63%), (positives 80%), (negatives: 45%)] [Train Acc (all
66%), (positive: 73%), (negatives: 59%), Loss 0.005405]
epoch 32. [Val Acc (all 62%), (positives 69%), (negatives: 55%)] [Train Acc (all
64%), (positive: 59%), (negatives: 70%), Loss 0.005403]
epoch 33. [Val Acc (all 65%), (positives 57%), (negatives: 73%)] [Train Acc (all
62%), (positive: 44%), (negatives: 80%), Loss 0.005403]
epoch 34. [Val Acc (all 71%), (positives 65%), (negatives: 76%)] [Train Acc (all
64%), (positive: 49%), (negatives: 79%), Loss 0.005401]
saving epoch 34 as best epoch
epoch 35. [Val Acc (all 75%), (positives 75%), (negatives: 75%)] [Train Acc (all
69%), (positive: 67%), (negatives: 72%), Loss 0.005399]
saving epoch 35 as best epoch
epoch 36. [Val Acc (all 66%), (positives 73%), (negatives: 59%)] [Train Acc (all
68%), (positive: 67%), (negatives: 68%), Loss 0.005397]
epoch 37. [Val Acc (all 70%), (positives 73%), (negatives: 67%)] [Train Acc (all
68%), (positive: 64%), (negatives: 73%), Loss 0.005393]
epoch 38. [Val Acc (all 71%), (positives 67%), (negatives: 75%)] [Train Acc (all
64%), (positive: 54%), (negatives: 74%), Loss 0.005391]
epoch 39. [Val Acc (all 68%), (positives 78%), (negatives: 57%)] [Train Acc (all
72%), (positive: 83%), (negatives: 61%), Loss 0.005387]
epoch 40. [Val Acc (all 70%), (positives 76%), (negatives: 63%)] [Train Acc (all
71%), (positive: 76%), (negatives: 66%), Loss 0.005382]
epoch 41. [Val Acc (all 69%), (positives 78%), (negatives: 59%)] [Train Acc (all
71%), (positive: 77%), (negatives: 66%), Loss 0.005376]
epoch 42. [Val Acc (all 74%), (positives 82%), (negatives: 65%)] [Train Acc (all
71%), (positive: 85%), (negatives: 56%), Loss 0.005370]
epoch 43. [Val Acc (all 74%), (positives 80%), (negatives: 67%)] [Train Acc (all
73%), (positive: 86%), (negatives: 61%), Loss 0.005365]
epoch 44. [Val Acc (all 70%), (positives 86%), (negatives: 53%)] [Train Acc (all
75%), (positive: 89%), (negatives: 60%), Loss 0.005360]
epoch 45. [Val Acc (all 75%), (positives 90%), (negatives: 61%)] [Train Acc (all
75%), (positive: 94%), (negatives: 56%), Loss 0.005349]
saving epoch 45 as best epoch
epoch 46. [Val Acc (all 75%), (positives 82%), (negatives: 69%)] [Train Acc (all
75%), (positive: 89%), (negatives: 61%), Loss 0.005337]
saving epoch 46 as best epoch
epoch 47. [Val Acc (all 73%), (positives 90%), (negatives: 55%)] [Train Acc (all
75%), (positive: 93%), (negatives: 58%), Loss 0.005325]
epoch 48. [Val Acc (all 75%), (positives 96%), (negatives: 55%)] [Train Acc (all
74%), (positive: 94%), (negatives: 55%), Loss 0.005315]
epoch 49. [Val Acc (all 75%), (positives 96%), (negatives: 53%)] [Train Acc (all
74%), (positive: 95%), (negatives: 54%), Loss 0.005296]
epoch 50. [Val Acc (all 75%), (positives 96%), (negatives: 55%)] [Train Acc (all
74%), (positive: 95%), (negatives: 54%), Loss 0.005283]
epoch 51. [Val Acc (all 75%), (positives 92%), (negatives: 57%)] [Train Acc (all
76%), (positive: 93%), (negatives: 59%), Loss 0.005255]
```

```
epoch 52. [Val Acc (all 77%), (positives 96%), (negatives: 59%)] [Train Acc (all
76%), (positive: 95%), (negatives: 58%), Loss 0.005237]
saving epoch 52 as best epoch
epoch 53. [Val Acc (all 72%), (positives 94%), (negatives: 49%)] [Train Acc (all
75%), (positive: 94%), (negatives: 55%), Loss 0.005210]
epoch 54. [Val Acc (all 75%), (positives 98%), (negatives: 53%)] [Train Acc (all
75%), (positive: 96%), (negatives: 55%), Loss 0.005186]
epoch 55. [Val Acc (all 77%), (positives 94%), (negatives: 61%)] [Train Acc (all
76%), (positive: 94%), (negatives: 59%), Loss 0.005147]
epoch 56. [Val Acc (all 74%), (positives 92%), (negatives: 55%)] [Train Acc (all
77%), (positive: 96%), (negatives: 58%), Loss 0.005112]
epoch 57. [Val Acc (all 75%), (positives 80%), (negatives: 71%)] [Train Acc (all
78%), (positive: 90%), (negatives: 65%), Loss 0.005073]
epoch 58. [Val Acc (all 75%), (positives 96%), (negatives: 55%)] [Train Acc (all
75%), (positive: 96%), (negatives: 54%), Loss 0.005035]
epoch 59. [Val Acc (all 82%), (positives 96%), (negatives: 69%)] [Train Acc (all
75%), (positive: 95%), (negatives: 55%), Loss 0.004983]
saving epoch 59 as best epoch
epoch 60. [Val Acc (all 74%), (positives 98%), (negatives: 49%)] [Train Acc (all
75%), (positive: 95%), (negatives: 54%), Loss 0.004942]
epoch 61. [Val Acc (all 79%), (positives 96%), (negatives: 63%)] [Train Acc (all
75%), (positive: 93%), (negatives: 56%), Loss 0.004898]
epoch 62. [Val Acc (all 74%), (positives 86%), (negatives: 61%)] [Train Acc (all
75%), (positive: 90%), (negatives: 61%), Loss 0.004834]
epoch 63. [Val Acc (all 79%), (positives 98%), (negatives: 61%)] [Train Acc (all
74%), (positive: 94%), (negatives: 54%), Loss 0.004777]
epoch 64. [Val Acc (all 75%), (positives 92%), (negatives: 57%)] [Train Acc (all
74%), (positive: 92%), (negatives: 57%), Loss 0.004708]
epoch 65. [Val Acc (all 75%), (positives 88%), (negatives: 63%)] [Train Acc (all
74%), (positive: 91%), (negatives: 57%), Loss 0.004624]
epoch 66. [Val Acc (all 74%), (positives 90%), (negatives: 57%)] [Train Acc (all
75%), (positive: 93%), (negatives: 56%), Loss 0.004638]
epoch 67. [Val Acc (all 73%), (positives 90%), (negatives: 55%)] [Train Acc (all
74%), (positive: 93%), (negatives: 56%), Loss 0.004517]
epoch 68. [Val Acc (all 75%), (positives 90%), (negatives: 61%)] [Train Acc (all
76%), (positive: 94%), (negatives: 58%), Loss 0.004462]
epoch 69. [Val Acc (all 74%), (positives 88%), (negatives: 59%)] [Train Acc (all
76%), (positive: 92%), (negatives: 59%), Loss 0.004454]
epoch 70. [Val Acc (all 76%), (positives 90%), (negatives: 63%)] [Train Acc (all
75%), (positive: 91%), (negatives: 60%), Loss 0.004340]
epoch 71. [Val Acc (all 79%), (positives 98%), (negatives: 61%)] [Train Acc (all
76%), (positive: 94%), (negatives: 58%), Loss 0.004349]
epoch 72. [Val Acc (all 77%), (positives 90%), (negatives: 65%)] [Train Acc (all
75%), (positive: 92%), (negatives: 59%), Loss 0.004208]
epoch 73. [Val Acc (all 81%), (positives 92%), (negatives: 71%)] [Train Acc (all
77%), (positive: 94%), (negatives: 59%), Loss 0.004167]
epoch 74. [Val Acc (all 79%), (positives 90%), (negatives: 69%)] [Train Acc (all
75%), (positive: 94%), (negatives: 56%), Loss 0.004169]
```

```
epoch 75. [Val Acc (all 75%), (positives 96%), (negatives: 55%)] [Train Acc (all
76%), (positive: 95%), (negatives: 58%), Loss 0.004077]
epoch 76. [Val Acc (all 75%), (positives 88%), (negatives: 61%)] [Train Acc (all
77%), (positive: 93%), (negatives: 61%), Loss 0.004113]
epoch 77. [Val Acc (all 78%), (positives 98%), (negatives: 59%)] [Train Acc (all
77%), (positive: 97%), (negatives: 56%), Loss 0.004018]
epoch 78. [Val Acc (all 75%), (positives 78%), (negatives: 73%)] [Train Acc (all
79%), (positive: 92%), (negatives: 66%), Loss 0.004001]
epoch 79. [Val Acc (all 79%), (positives 96%), (negatives: 63%)] [Train Acc (all
76%), (positive: 95%), (negatives: 56%), Loss 0.004002]
epoch 80. [Val Acc (all 82%), (positives 98%), (negatives: 67%)] [Train Acc (all
77%), (positive: 96%), (negatives: 57%), Loss 0.003975]
epoch 81. [Val Acc (all 74%), (positives 80%), (negatives: 67%)] [Train Acc (all
79%), (positive: 94%), (negatives: 63%), Loss 0.003908]
epoch 82. [Val Acc (all 79%), (positives 96%), (negatives: 63%)] [Train Acc (all
78%), (positive: 97%), (negatives: 58%), Loss 0.003867]
epoch 83. [Val Acc (all 77%), (positives 80%), (negatives: 75%)] [Train Acc (all
80%), (positive: 95%), (negatives: 65%), Loss 0.003850]
epoch 84. [Val Acc (all 75%), (positives 96%), (negatives: 55%)] [Train Acc (all
79%), (positive: 97%), (negatives: 61%), Loss 0.003770]
epoch 85. [Val Acc (all 80%), (positives 92%), (negatives: 69%)] [Train Acc (all
80%), (positive: 96%), (negatives: 64%), Loss 0.003725]
epoch 86. [Val Acc (all 77%), (positives 84%), (negatives: 71%)] [Train Acc (all
80%), (positive: 96%), (negatives: 65%), Loss 0.003682]
epoch 87. [Val Acc (all 82%), (positives 96%), (negatives: 69%)] [Train Acc (all
78%), (positive: 98%), (negatives: 59%), Loss 0.003686]
epoch 88. [Val Acc (all 78%), (positives 80%), (negatives: 76%)] [Train Acc (all
82%), (positive: 94%), (negatives: 69%), Loss 0.003721]
epoch 89. [Val Acc (all 80%), (positives 94%), (negatives: 67%)] [Train Acc (all
79%), (positive: 96%), (negatives: 62%), Loss 0.003754]
epoch 90. [Val Acc (all 81%), (positives 84%), (negatives: 78%)] [Train Acc (all
81%), (positive: 96%), (negatives: 66%), Loss 0.003613]
epoch 91. [Val Acc (all 76%), (positives 84%), (negatives: 69%)] [Train Acc (all
80%), (positive: 96%), (negatives: 63%), Loss 0.003560]
epoch 92. [Val Acc (all 81%), (positives 92%), (negatives: 71%)] [Train Acc (all
79%), (positive: 97%), (negatives: 62%), Loss 0.003449]
epoch 93. [Val Acc (all 83%), (positives 86%), (negatives: 80%)] [Train Acc (all
82%), (positive: 94%), (negatives: 71%), Loss 0.003635]
saving epoch 93 as best epoch
epoch 94. [Val Acc (all 75%), (positives 90%), (negatives: 61%)] [Train Acc (all
81%), (positive: 97%), (negatives: 64%), Loss 0.003555]
epoch 95. [Val Acc (all 77%), (positives 88%), (negatives: 67%)] [Train Acc (all
81%), (positive: 96%), (negatives: 66%), Loss 0.003481]
epoch 96. [Val Acc (all 78%), (positives 90%), (negatives: 67%)] [Train Acc (all
83%), (positive: 96%), (negatives: 69%), Loss 0.003454]
epoch 97. [Val Acc (all 79%), (positives 88%), (negatives: 71%)] [Train Acc (all
81%), (positive: 95%), (negatives: 67%), Loss 0.003500]
epoch 98. [Val Acc (all 78%), (positives 90%), (negatives: 67%)] [Train Acc (all
```

```
81%), (positive: 96%), (negatives: 66%), Loss 0.003391]
epoch 99. [Val Acc (all 76%), (positives 92%), (negatives: 61%)] [Train Acc (all
79%), (positive: 97%), (negatives: 61%), Loss 0.003332]
epoch 100. [Val Acc (all 82%), (positives 90%), (negatives: 75%)] [Train Acc
(all 82%), (positive: 96%), (negatives: 68%), Loss 0.003267]
epoch 101. [Val Acc (all 81%), (positives 92%), (negatives: 71%)] [Train Acc
(all 80%), (positive: 97%), (negatives: 64%), Loss 0.003375]
epoch 102. [Val Acc (all 81%), (positives 90%), (negatives: 73%)] [Train Acc
(all 82%), (positive: 96%), (negatives: 68%), Loss 0.003415]
epoch 103. [Val Acc (all 78%), (positives 92%), (negatives: 65%)] [Train Acc
(all 81%), (positive: 95%), (negatives: 66%), Loss 0.003371]
epoch 104. [Val Acc (all 77%), (positives 86%), (negatives: 69%)] [Train Acc
(all 82%), (positive: 95%), (negatives: 68%), Loss 0.003275]
epoch 105. [Val Acc (all 82%), (positives 90%), (negatives: 75%)] [Train Acc
(all 83%), (positive: 96%), (negatives: 69%), Loss 0.003267]
epoch 106. [Val Acc (all 78%), (positives 88%), (negatives: 69%)] [Train Acc
(all 82%), (positive: 95%), (negatives: 69%), Loss 0.003294]
epoch 107. [Val Acc (all 83%), (positives 92%), (negatives: 75%)] [Train Acc
(all 82%), (positive: 95%), (negatives: 69%), Loss 0.003304]
epoch 108. [Val Acc (all 77%), (positives 90%), (negatives: 65%)] [Train Acc
(all 82%), (positive: 98%), (negatives: 66%), Loss 0.003230]
epoch 109. [Val Acc (all 76%), (positives 84%), (negatives: 69%)] [Train Acc
(all 84%), (positive: 94%), (negatives: 73%), Loss 0.003293]
epoch 110. [Val Acc (all 80%), (positives 92%), (negatives: 69%)] [Train Acc
(all 83%), (positive: 97%), (negatives: 68%), Loss 0.003274]
epoch 111. [Val Acc (all 78%), (positives 90%), (negatives: 67%)] [Train Acc
(all 84%), (positive: 94%), (negatives: 73%), Loss 0.003238]
epoch 112. [Val Acc (all 79%), (positives 92%), (negatives: 67%)] [Train Acc
(all 81%), (positive: 98%), (negatives: 65%), Loss 0.003086]
epoch 113. [Val Acc (all 78%), (positives 84%), (negatives: 73%)] [Train Acc
(all 85%), (positive: 92%), (negatives: 78%), Loss 0.003210]
epoch 114. [Val Acc (all 82%), (positives 92%), (negatives: 73%)] [Train Acc
(all 83%), (positive: 97%), (negatives: 68%), Loss 0.003233]
epoch 115. [Val Acc (all 78%), (positives 84%), (negatives: 73%)] [Train Acc
(all 86%), (positive: 95%), (negatives: 77%), Loss 0.003062]
epoch 116. [Val Acc (all 79%), (positives 92%), (negatives: 67%)] [Train Acc
(all 83%), (positive: 96%), (negatives: 69%), Loss 0.002982]
epoch 117. [Val Acc (all 77%), (positives 84%), (negatives: 71%)] [Train Acc
(all 84%), (positive: 94%), (negatives: 73%), Loss 0.003170]
epoch 118. [Val Acc (all 77%), (positives 90%), (negatives: 65%)] [Train Acc
(all 83%), (positive: 97%), (negatives: 68%), Loss 0.003127]
epoch 119. [Val Acc (all 84%), (positives 86%), (negatives: 82%)] [Train Acc
(all 85%), (positive: 92%), (negatives: 78%), Loss 0.003172]
saving epoch 119 as best epoch
epoch 120. [Val Acc (all 76%), (positives 92%), (negatives: 61%)] [Train Acc
(all 81%), (positive: 97%), (negatives: 66%), Loss 0.003182]
epoch 121. [Val Acc (all 78%), (positives 86%), (negatives: 71%)] [Train Acc
(all 84%), (positive: 93%), (negatives: 75%), Loss 0.003097]
```

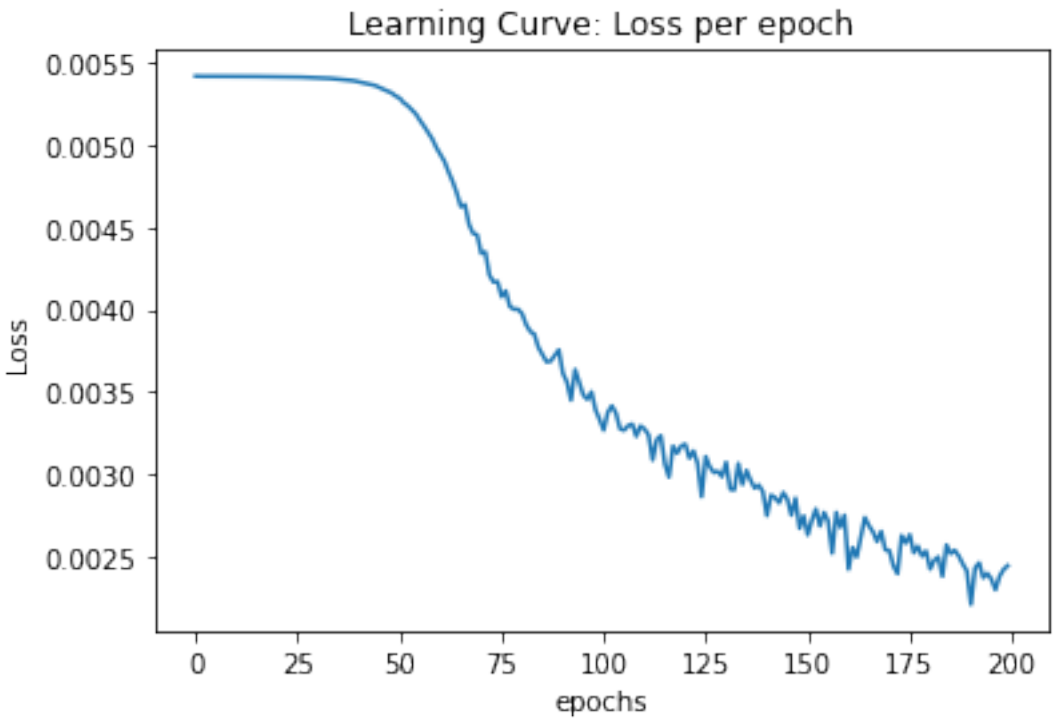
```
epoch 122. [Val Acc (all 83%), (positives 88%), (negatives: 78%)] [Train Acc
(all 83%), (positive: 95%), (negatives: 72%), Loss 0.003141]
epoch 123. [Val Acc (all 79%), (positives 90%), (negatives: 69%)] [Train Acc
(all 85%), (positive: 94%), (negatives: 76%), Loss 0.003067]
epoch 124. [Val Acc (all 79%), (positives 92%), (negatives: 67%)] [Train Acc
(all 81%), (positive: 96%), (negatives: 66%), Loss 0.002864]
epoch 125. [Val Acc (all 81%), (positives 88%), (negatives: 75%)] [Train Acc
(all 86%), (positive: 95%), (negatives: 77%), Loss 0.003107]
epoch 126. [Val Acc (all 78%), (positives 86%), (negatives: 71%)] [Train Acc
(all 83%), (positive: 95%), (negatives: 70%), Loss 0.003044]
epoch 127. [Val Acc (all 78%), (positives 88%), (negatives: 69%)] [Train Acc
(all 82%), (positive: 93%), (negatives: 71%), Loss 0.003010]
epoch 128. [Val Acc (all 82%), (positives 86%), (negatives: 78%)] [Train Acc
(all 83%), (positive: 93%), (negatives: 74%), Loss 0.003017]
epoch 129. [Val Acc (all 77%), (positives 88%), (negatives: 67%)] [Train Acc
(all 83%), (positive: 96%), (negatives: 70%), Loss 0.002985]
epoch 130. [Val Acc (all 80%), (positives 86%), (negatives: 75%)] [Train Acc
(all 86%), (positive: 94%), (negatives: 79%), Loss 0.003072]
epoch 131. [Val Acc (all 79%), (positives 92%), (negatives: 67%)] [Train Acc
(all 83%), (positive: 96%), (negatives: 70%), Loss 0.002909]
epoch 132. [Val Acc (all 80%), (positives 86%), (negatives: 75%)] [Train Acc
(all 86%), (positive: 94%), (negatives: 79%), Loss 0.002905]
epoch 133. [Val Acc (all 80%), (positives 86%), (negatives: 75%)] [Train Acc
(all 85%), (positive: 94%), (negatives: 77%), Loss 0.003065]
epoch 134. [Val Acc (all 83%), (positives 88%), (negatives: 78%)] [Train Acc
(all 86%), (positive: 95%), (negatives: 76%), Loss 0.002938]
epoch 135. [Val Acc (all 80%), (positives 88%), (negatives: 73%)] [Train Acc
(all 86%), (positive: 96%), (negatives: 77%), Loss 0.003023]
epoch 136. [Val Acc (all 79%), (positives 88%), (negatives: 71%)] [Train Acc
(all 84%), (positive: 94%), (negatives: 73%), Loss 0.002963]
epoch 137. [Val Acc (all 78%), (positives 88%), (negatives: 69%)] [Train Acc
(all 86%), (positive: 94%), (negatives: 78%), Loss 0.002915]
epoch 138. [Val Acc (all 80%), (positives 92%), (negatives: 69%)] [Train Acc
(all 85%), (positive: 97%), (negatives: 72%), Loss 0.002934]
epoch 139. [Val Acc (all 78%), (positives 88%), (negatives: 69%)] [Train Acc
(all 84%), (positive: 95%), (negatives: 72%), Loss 0.002894]
epoch 140. [Val Acc (all 81%), (positives 86%), (negatives: 76%)] [Train Acc
(all 84%), (positive: 94%), (negatives: 74%), Loss 0.002749]
epoch 141. [Val Acc (all 77%), (positives 88%), (negatives: 67%)] [Train Acc
(all 84%), (positive: 95%), (negatives: 72%), Loss 0.002873]
epoch 142. [Val Acc (all 79%), (positives 90%), (negatives: 69%)] [Train Acc
(all 83%), (positive: 96%), (negatives: 71%), Loss 0.002858]
epoch 143. [Val Acc (all 80%), (positives 88%), (negatives: 73%)] [Train Acc
(all 86%), (positive: 94%), (negatives: 78%), Loss 0.002826]
epoch 144. [Val Acc (all 79%), (positives 88%), (negatives: 71%)] [Train Acc
(all 86%), (positive: 95%), (negatives: 77%), Loss 0.002888]
epoch 145. [Val Acc (all 80%), (positives 88%), (negatives: 73%)] [Train Acc
(all 86%), (positive: 94%), (negatives: 79%), Loss 0.002849]
```

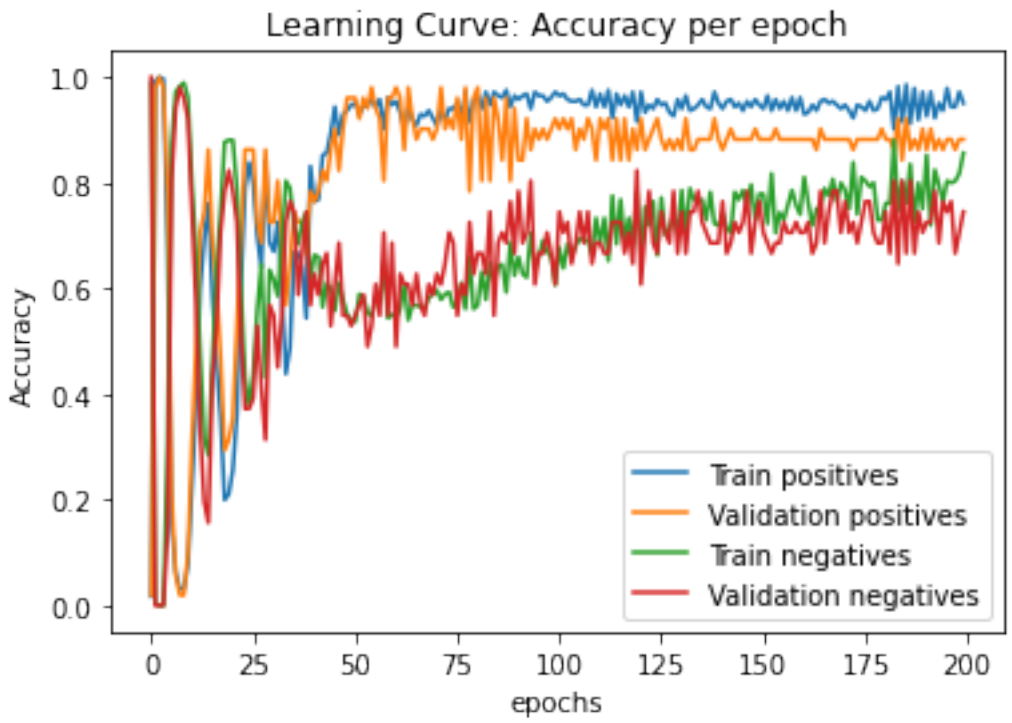
epoch 146. [Val Acc (all 83%), (positives 88%), (negatives: 78%)] [Train Acc (all 86%), (positive: 95%), (negatives: 76%), Loss 0.002751]
epoch 147. [Val Acc (all 77%), (positives 88%), (negatives: 67%)] [Train Acc (all 87%), (positive: 94%), (negatives: 80%), Loss 0.002855]
epoch 148. [Val Acc (all 81%), (positives 90%), (negatives: 73%)] [Train Acc (all 84%), (positive: 96%), (negatives: 73%), Loss 0.002671]
epoch 149. [Val Acc (all 82%), (positives 88%), (negatives: 76%)] [Train Acc (all 86%), (positive: 95%), (negatives: 78%), Loss 0.002747]
epoch 150. [Val Acc (all 79%), (positives 88%), (negatives: 71%)] [Train Acc (all 86%), (positive: 94%), (negatives: 78%), Loss 0.002632]
epoch 151. [Val Acc (all 78%), (positives 88%), (negatives: 69%)] [Train Acc (all 84%), (positive: 96%), (negatives: 72%), Loss 0.002718]
epoch 152. [Val Acc (all 77%), (positives 88%), (negatives: 67%)] [Train Acc (all 88%), (positive: 94%), (negatives: 82%), Loss 0.002789]
epoch 153. [Val Acc (all 79%), (positives 90%), (negatives: 69%)] [Train Acc (all 83%), (positive: 96%), (negatives: 71%), Loss 0.002685]
epoch 154. [Val Acc (all 78%), (positives 88%), (negatives: 69%)] [Train Acc (all 84%), (positive: 94%), (negatives: 74%), Loss 0.002767]
epoch 155. [Val Acc (all 80%), (positives 88%), (negatives: 73%)] [Train Acc (all 84%), (positive: 95%), (negatives: 73%), Loss 0.002716]
epoch 156. [Val Acc (all 79%), (positives 88%), (negatives: 71%)] [Train Acc (all 85%), (positive: 94%), (negatives: 76%), Loss 0.002520]
epoch 157. [Val Acc (all 79%), (positives 88%), (negatives: 71%)] [Train Acc (all 87%), (positive: 96%), (negatives: 79%), Loss 0.002768]
epoch 158. [Val Acc (all 80%), (positives 88%), (negatives: 73%)] [Train Acc (all 85%), (positive: 94%), (negatives: 76%), Loss 0.002678]
epoch 159. [Val Acc (all 79%), (positives 88%), (negatives: 71%)] [Train Acc (all 85%), (positive: 96%), (negatives: 74%), Loss 0.002751]
epoch 160. [Val Acc (all 79%), (positives 88%), (negatives: 71%)] [Train Acc (all 87%), (positive: 93%), (negatives: 81%), Loss 0.002423]
epoch 161. [Val Acc (all 80%), (positives 88%), (negatives: 73%)] [Train Acc (all 85%), (positive: 93%), (negatives: 76%), Loss 0.002555]
epoch 162. [Val Acc (all 77%), (positives 88%), (negatives: 67%)] [Train Acc (all 83%), (positive: 95%), (negatives: 72%), Loss 0.002497]
epoch 163. [Val Acc (all 77%), (positives 86%), (negatives: 69%)] [Train Acc (all 84%), (positive: 94%), (negatives: 75%), Loss 0.002621]
epoch 164. [Val Acc (all 84%), (positives 90%), (negatives: 78%)] [Train Acc (all 85%), (positive: 95%), (negatives: 75%), Loss 0.002737]
epoch 165. [Val Acc (all 78%), (positives 88%), (negatives: 69%)] [Train Acc (all 87%), (positive: 96%), (negatives: 79%), Loss 0.002687]
epoch 166. [Val Acc (all 81%), (positives 88%), (negatives: 75%)] [Train Acc (all 86%), (positive: 95%), (negatives: 78%), Loss 0.002650]
epoch 167. [Val Acc (all 83%), (positives 88%), (negatives: 78%)] [Train Acc (all 88%), (positive: 95%), (negatives: 81%), Loss 0.002591]
epoch 168. [Val Acc (all 83%), (positives 88%), (negatives: 78%)] [Train Acc (all 86%), (positive: 95%), (negatives: 76%), Loss 0.002653]
epoch 169. [Val Acc (all 79%), (positives 88%), (negatives: 71%)] [Train Acc (all 87%), (positive: 94%), (negatives: 80%), Loss 0.002541]

```
epoch 170. [Val Acc (all 82%), (positives 88%), (negatives: 76%)] [Train Acc
(all 87%), (positive: 95%), (negatives: 79%), Loss 0.002536]
epoch 171. [Val Acc (all 79%), (positives 88%), (negatives: 71%)] [Train Acc
(all 84%), (positive: 94%), (negatives: 75%), Loss 0.002443]
epoch 172. [Val Acc (all 79%), (positives 86%), (negatives: 73%)] [Train Acc
(all 88%), (positive: 92%), (negatives: 84%), Loss 0.002396]
epoch 173. [Val Acc (all 78%), (positives 88%), (negatives: 69%)] [Train Acc
(all 84%), (positive: 95%), (negatives: 74%), Loss 0.002627]
epoch 174. [Val Acc (all 80%), (positives 88%), (negatives: 73%)] [Train Acc
(all 88%), (positive: 94%), (negatives: 81%), Loss 0.002582]
epoch 175. [Val Acc (all 82%), (positives 88%), (negatives: 76%)] [Train Acc
(all 87%), (positive: 94%), (negatives: 80%), Loss 0.002632]
epoch 176. [Val Acc (all 82%), (positives 88%), (negatives: 76%)] [Train Acc
(all 87%), (positive: 95%), (negatives: 79%), Loss 0.002523]
epoch 177. [Val Acc (all 79%), (positives 88%), (negatives: 71%)] [Train Acc
(all 87%), (positive: 94%), (negatives: 80%), Loss 0.002562]
epoch 178. [Val Acc (all 78%), (positives 88%), (negatives: 69%)] [Train Acc
(all 84%), (positive: 94%), (negatives: 73%), Loss 0.002502]
epoch 179. [Val Acc (all 79%), (positives 90%), (negatives: 69%)] [Train Acc
(all 85%), (positive: 96%), (negatives: 73%), Loss 0.002533]
epoch 180. [Val Acc (all 81%), (positives 88%), (negatives: 75%)] [Train Acc
(all 86%), (positive: 96%), (negatives: 76%), Loss 0.002424]
epoch 181. [Val Acc (all 77%), (positives 88%), (negatives: 67%)] [Train Acc
(all 87%), (positive: 97%), (negatives: 76%), Loss 0.002477]
epoch 182. [Val Acc (all 83%), (positives 86%), (negatives: 80%)] [Train Acc
(all 89%), (positive: 90%), (negatives: 88%), Loss 0.002498]
epoch 183. [Val Acc (all 78%), (positives 92%), (negatives: 65%)] [Train Acc
(all 86%), (positive: 98%), (negatives: 75%), Loss 0.002381]
epoch 184. [Val Acc (all 82%), (positives 84%), (negatives: 80%)] [Train Acc
(all 86%), (positive: 92%), (negatives: 80%), Loss 0.002570]
epoch 185. [Val Acc (all 79%), (positives 92%), (negatives: 67%)] [Train Acc
(all 87%), (positive: 99%), (negatives: 76%), Loss 0.002517]
epoch 186. [Val Acc (all 83%), (positives 86%), (negatives: 80%)] [Train Acc
(all 87%), (positive: 91%), (negatives: 84%), Loss 0.002539]
epoch 187. [Val Acc (all 77%), (positives 88%), (negatives: 67%)] [Train Acc
(all 85%), (positive: 98%), (negatives: 73%), Loss 0.002503]
epoch 188. [Val Acc (all 80%), (positives 86%), (negatives: 75%)] [Train Acc
(all 86%), (positive: 92%), (negatives: 80%), Loss 0.002454]
epoch 189. [Val Acc (all 80%), (positives 90%), (negatives: 71%)] [Train Acc
(all 86%), (positive: 97%), (negatives: 75%), Loss 0.002413]
epoch 190. [Val Acc (all 79%), (positives 86%), (negatives: 73%)] [Train Acc
(all 89%), (positive: 93%), (negatives: 85%), Loss 0.002209]
epoch 191. [Val Acc (all 81%), (positives 90%), (negatives: 73%)] [Train Acc
(all 85%), (positive: 97%), (negatives: 72%), Loss 0.002430]
epoch 192. [Val Acc (all 82%), (positives 86%), (negatives: 78%)] [Train Acc
(all 87%), (positive: 92%), (negatives: 82%), Loss 0.002462]
epoch 193. [Val Acc (all 78%), (positives 88%), (negatives: 69%)] [Train Acc
(all 87%), (positive: 95%), (negatives: 80%), Loss 0.002371]
```

```
epoch 194. [Val Acc (all 81%), (positives 86%), (negatives: 76%)] [Train Acc
(all 85%), (positive: 94%), (negatives: 76%), Loss 0.002398]
epoch 195. [Val Acc (all 81%), (positives 88%), (negatives: 75%)] [Train Acc
(all 89%), (positive: 98%), (negatives: 80%), Loss 0.002363]
epoch 196. [Val Acc (all 82%), (positives 88%), (negatives: 76%)] [Train Acc
(all 87%), (positive: 94%), (negatives: 80%), Loss 0.002298]
epoch 197. [Val Acc (all 76%), (positives 86%), (negatives: 67%)] [Train Acc
(all 87%), (positive: 94%), (negatives: 80%), Loss 0.002380]
epoch 198. [Val Acc (all 79%), (positives 88%), (negatives: 71%)] [Train Acc
(all 89%), (positive: 97%), (negatives: 82%), Loss 0.002422]
epoch 199. [Val Acc (all 81%), (positives 88%), (negatives: 75%)] [Train Acc
(all 90%), (positive: 95%), (negatives: 86%), Loss 0.002444]
```

[ ]:
```
#printing the loss-epochs loss and learning curve
plot_learning_curve(*learning_curve_info)

with open('/content/gdrive/My Drive/learning_curve_info_CNN_1.pickle', 'wb') as␣
 ↪handle:
    pickle.dump(learning_curve_info, handle, protocol=pickle.HIGHEST_PROTOCOL)
```





[ ]:
```
# Include the training curves for the two models.
model_CNNChannel = CNNChannel(n=24).to(device)
learning_curve_info = train_model(model_CNNChannel,␣
 ↪training_data=training_data, validation_data=validation_data, batch_size=64,␣
 ↪learning_rate=0.0001, weight_decay=5e-7, epochs=50, checkpoint_path='/
 ↪content/gdrive/My Drive/pytorch_CNNChannel_2_ckpt-{}.pk',␣
 ↪save_best_model=True)
```

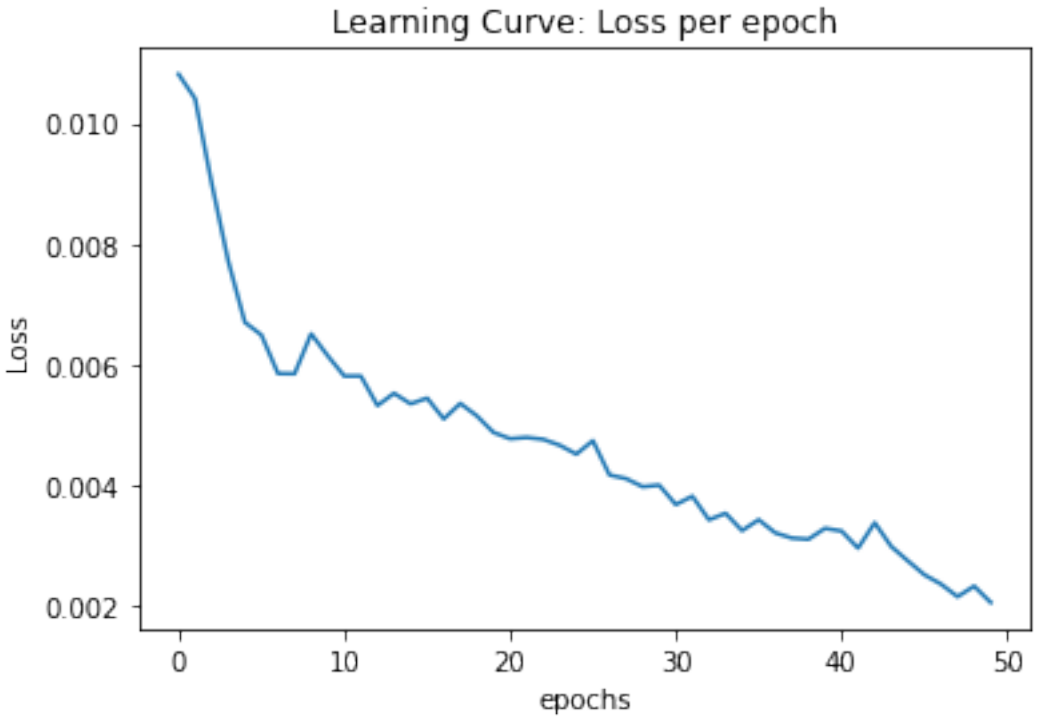
```
epoch 0. [Val Acc (all 50%), (positives 0%), (negatives: 100%)] [Train Acc (all
50%), (positive: 2%), (negatives: 98%), Loss 0.010816]
saving epoch 0 as best epoch
epoch 1. [Val Acc (all 68%), (positives 53%), (negatives: 82%)] [Train Acc (all
68%), (positive: 60%), (negatives: 75%), Loss 0.010409]
saving epoch 1 as best epoch
epoch 2. [Val Acc (all 78%), (positives 94%), (negatives: 63%)] [Train Acc (all
75%), (positive: 93%), (negatives: 58%), Loss 0.009004]
saving epoch 2 as best epoch
epoch 3. [Val Acc (all 81%), (positives 94%), (negatives: 69%)] [Train Acc (all
81%), (positive: 89%), (negatives: 73%), Loss 0.007713]
saving epoch 3 as best epoch
epoch 4. [Val Acc (all 85%), (positives 94%), (negatives: 76%)] [Train Acc (all
84%), (positive: 88%), (negatives: 80%), Loss 0.006700]
saving epoch 4 as best epoch
epoch 5. [Val Acc (all 90%), (positives 94%), (negatives: 86%)] [Train Acc (all
```

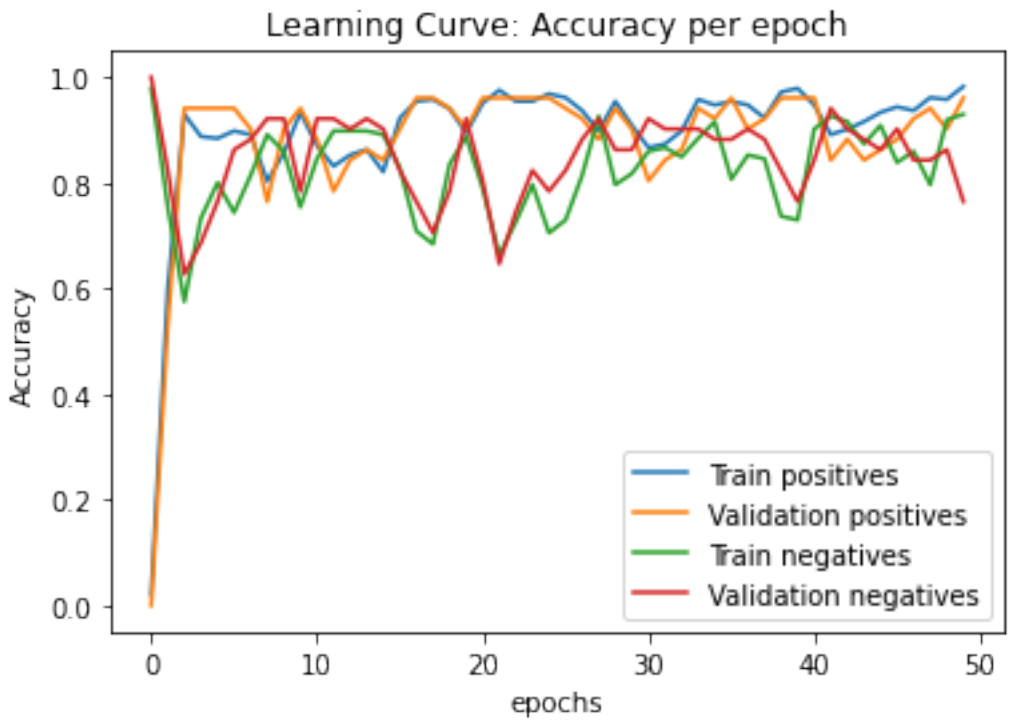
```
82%), (positive: 90%), (negatives: 74%), Loss 0.006489]
saving epoch 5 as best epoch
epoch 6. [Val Acc (all 89%), (positives 90%), (negatives: 88%)] [Train Acc (all
85%), (positive: 89%), (negatives: 81%), Loss 0.005849]
epoch 7. [Val Acc (all 84%), (positives 76%), (negatives: 92%)] [Train Acc (all
85%), (positive: 80%), (negatives: 89%), Loss 0.005846]
epoch 8. [Val Acc (all 91%), (positives 90%), (negatives: 92%)] [Train Acc (all
86%), (positive: 86%), (negatives: 86%), Loss 0.006509]
saving epoch 8 as best epoch
epoch 9. [Val Acc (all 86%), (positives 94%), (negatives: 78%)] [Train Acc (all
84%), (positive: 93%), (negatives: 75%), Loss 0.006144]
epoch 10. [Val Acc (all 90%), (positives 88%), (negatives: 92%)] [Train Acc (all
86%), (positive: 87%), (negatives: 85%), Loss 0.005807]
epoch 11. [Val Acc (all 85%), (positives 78%), (negatives: 92%)] [Train Acc (all
86%), (positive: 83%), (negatives: 90%), Loss 0.005807]
epoch 12. [Val Acc (all 87%), (positives 84%), (negatives: 90%)] [Train Acc (all
88%), (positive: 85%), (negatives: 90%), Loss 0.005318]
epoch 13. [Val Acc (all 89%), (positives 86%), (negatives: 92%)] [Train Acc (all
88%), (positive: 86%), (negatives: 90%), Loss 0.005521]
epoch 14. [Val Acc (all 87%), (positives 84%), (negatives: 90%)] [Train Acc (all
86%), (positive: 82%), (negatives: 89%), Loss 0.005347]
epoch 15. [Val Acc (all 86%), (positives 90%), (negatives: 82%)] [Train Acc (all
88%), (positive: 92%), (negatives: 83%), Loss 0.005441]
epoch 16. [Val Acc (all 86%), (positives 96%), (negatives: 76%)] [Train Acc (all
83%), (positive: 95%), (negatives: 71%), Loss 0.005096]
epoch 17. [Val Acc (all 83%), (positives 96%), (negatives: 71%)] [Train Acc (all
82%), (positive: 96%), (negatives: 68%), Loss 0.005352]
epoch 18. [Val Acc (all 86%), (positives 94%), (negatives: 78%)] [Train Acc (all
89%), (positive: 94%), (negatives: 84%), Loss 0.005146]
epoch 19. [Val Acc (all 91%), (positives 90%), (negatives: 92%)] [Train Acc (all
89%), (positive: 89%), (negatives: 89%), Loss 0.004872]
epoch 20. [Val Acc (all 88%), (positives 96%), (negatives: 80%)] [Train Acc (all
87%), (positive: 95%), (negatives: 79%), Loss 0.004768]
epoch 21. [Val Acc (all 80%), (positives 96%), (negatives: 65%)] [Train Acc (all
82%), (positive: 98%), (negatives: 66%), Loss 0.004789]
epoch 22. [Val Acc (all 85%), (positives 96%), (negatives: 75%)] [Train Acc (all
84%), (positive: 95%), (negatives: 73%), Loss 0.004757]
epoch 23. [Val Acc (all 89%), (positives 96%), (negatives: 82%)] [Train Acc (all
88%), (positive: 95%), (negatives: 80%), Loss 0.004659]
epoch 24. [Val Acc (all 87%), (positives 96%), (negatives: 78%)] [Train Acc (all
84%), (positive: 97%), (negatives: 71%), Loss 0.004512]
epoch 25. [Val Acc (all 88%), (positives 94%), (negatives: 82%)] [Train Acc (all
85%), (positive: 96%), (negatives: 73%), Loss 0.004735]
epoch 26. [Val Acc (all 90%), (positives 92%), (negatives: 88%)] [Train Acc (all
88%), (positive: 94%), (negatives: 81%), Loss 0.004166]
epoch 27. [Val Acc (all 90%), (positives 88%), (negatives: 92%)] [Train Acc (all
91%), (positive: 90%), (negatives: 93%), Loss 0.004105]
epoch 28. [Val Acc (all 90%), (positives 94%), (negatives: 86%)] [Train Acc (all
```

```
88%), (positive: 95%), (negatives: 80%), Loss 0.003971]
epoch 29. [Val Acc (all 88%), (positives 90%), (negatives: 86%)] [Train Acc (all
86%), (positive: 91%), (negatives: 82%), Loss 0.003997]
epoch 30. [Val Acc (all 86%), (positives 80%), (negatives: 92%)] [Train Acc (all
86%), (positive: 87%), (negatives: 86%), Loss 0.003675]
epoch 31. [Val Acc (all 87%), (positives 84%), (negatives: 90%)] [Train Acc (all
87%), (positive: 87%), (negatives: 87%), Loss 0.003814]
epoch 32. [Val Acc (all 88%), (positives 86%), (negatives: 90%)] [Train Acc (all
87%), (positive: 90%), (negatives: 85%), Loss 0.003422]
epoch 33. [Val Acc (all 92%), (positives 94%), (negatives: 90%)] [Train Acc (all
92%), (positive: 96%), (negatives: 88%), Loss 0.003530]
saving epoch 33 as best epoch
epoch 34. [Val Acc (all 90%), (positives 92%), (negatives: 88%)] [Train Acc (all
93%), (positive: 95%), (negatives: 92%), Loss 0.003243]
epoch 35. [Val Acc (all 92%), (positives 96%), (negatives: 88%)] [Train Acc (all
88%), (positive: 95%), (negatives: 81%), Loss 0.003423]
epoch 36. [Val Acc (all 90%), (positives 90%), (negatives: 90%)] [Train Acc (all
90%), (positive: 95%), (negatives: 85%), Loss 0.003206]
epoch 37. [Val Acc (all 90%), (positives 92%), (negatives: 88%)] [Train Acc (all
88%), (positive: 92%), (negatives: 85%), Loss 0.003119]
epoch 38. [Val Acc (all 89%), (positives 96%), (negatives: 82%)] [Train Acc (all
85%), (positive: 97%), (negatives: 74%), Loss 0.003099]
epoch 39. [Val Acc (all 86%), (positives 96%), (negatives: 76%)] [Train Acc (all
85%), (positive: 98%), (negatives: 73%), Loss 0.003281]
epoch 40. [Val Acc (all 90%), (positives 96%), (negatives: 84%)] [Train Acc (all
92%), (positive: 95%), (negatives: 90%), Loss 0.003237]
epoch 41. [Val Acc (all 89%), (positives 84%), (negatives: 94%)] [Train Acc (all
91%), (positive: 89%), (negatives: 93%), Loss 0.002952]
epoch 42. [Val Acc (all 89%), (positives 88%), (negatives: 90%)] [Train Acc (all
91%), (positive: 90%), (negatives: 92%), Loss 0.003374]
epoch 43. [Val Acc (all 86%), (positives 84%), (negatives: 88%)] [Train Acc (all
89%), (positive: 92%), (negatives: 87%), Loss 0.002979]
epoch 44. [Val Acc (all 86%), (positives 86%), (negatives: 86%)] [Train Acc (all
92%), (positive: 93%), (negatives: 91%), Loss 0.002742]
epoch 45. [Val Acc (all 89%), (positives 88%), (negatives: 90%)] [Train Acc (all
89%), (positive: 94%), (negatives: 84%), Loss 0.002510]
epoch 46. [Val Acc (all 88%), (positives 92%), (negatives: 84%)] [Train Acc (all
90%), (positive: 94%), (negatives: 86%), Loss 0.002356]
epoch 47. [Val Acc (all 89%), (positives 94%), (negatives: 84%)] [Train Acc (all
88%), (positive: 96%), (negatives: 80%), Loss 0.002147]
epoch 48. [Val Acc (all 88%), (positives 90%), (negatives: 86%)] [Train Acc (all
94%), (positive: 96%), (negatives: 92%), Loss 0.002321]
epoch 49. [Val Acc (all 86%), (positives 96%), (negatives: 76%)] [Train Acc (all
96%), (positive: 98%), (negatives: 93%), Loss 0.002050]
```

[ ]:
```
plot_learning_curve(*learning_curve_info)
```

```
with open('/content/gdrive/My Drive/learning_curve_info_pytorch_CNNChannel_2.
 ↪pickle', 'wb') as handle:
    pickle.dump(learning_curve_info, handle, protocol=pickle.HIGHEST_PROTOCOL)
```



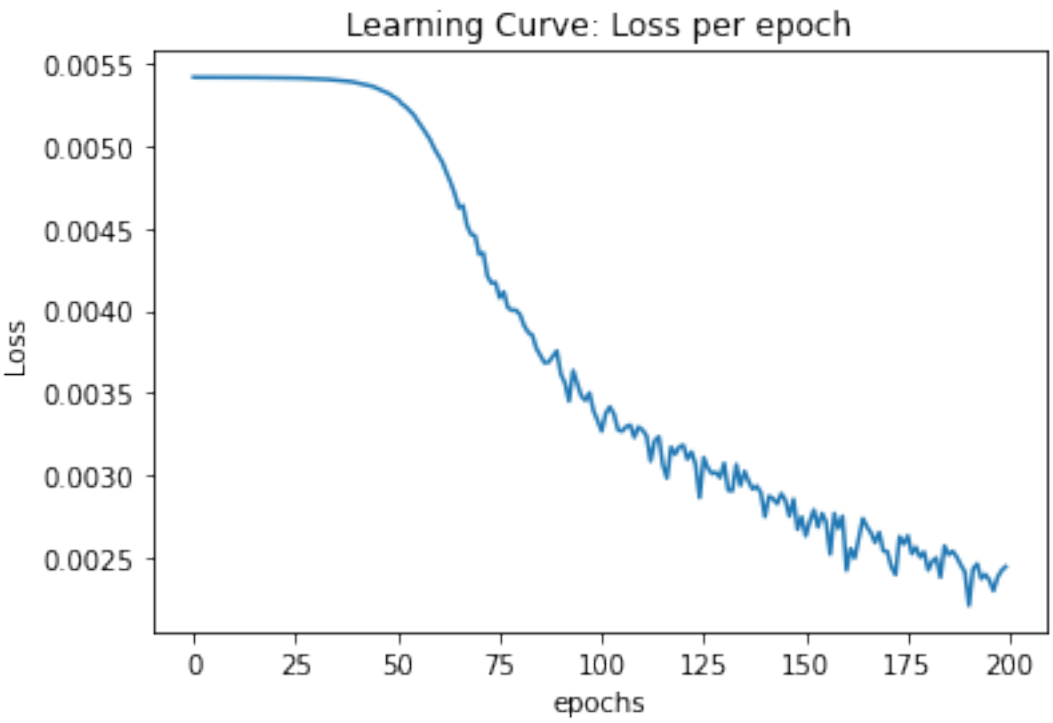


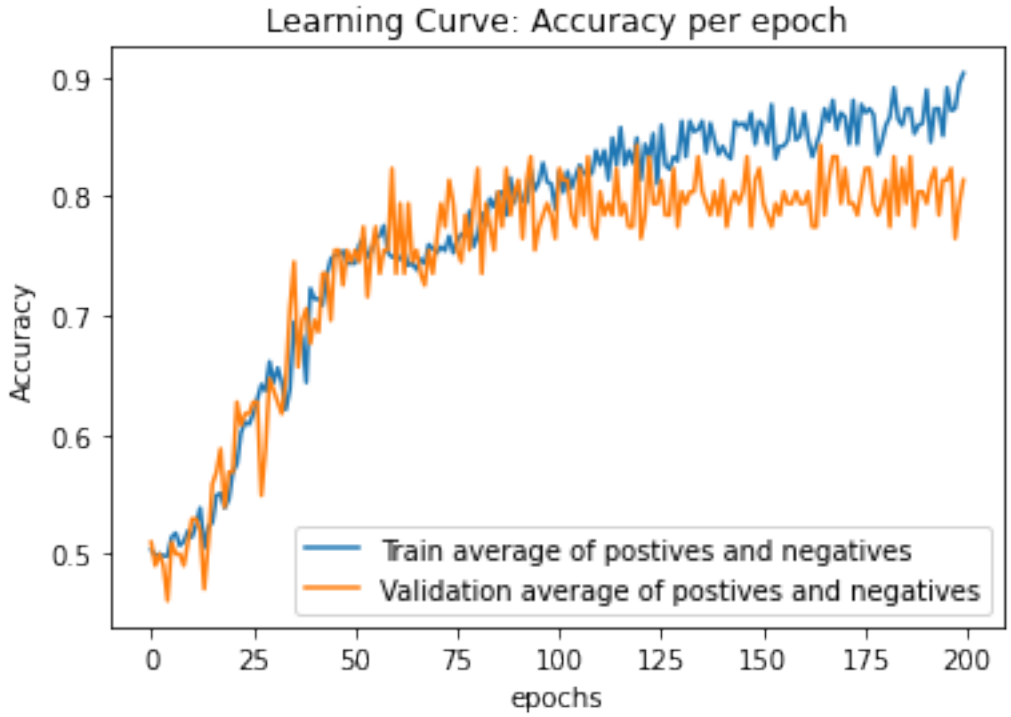
1.5.4 Part (d) -- 4%

Include your training curves for the **best** models from each of Q2(a) and Q2(b). These are the models that you will use in Question 4.

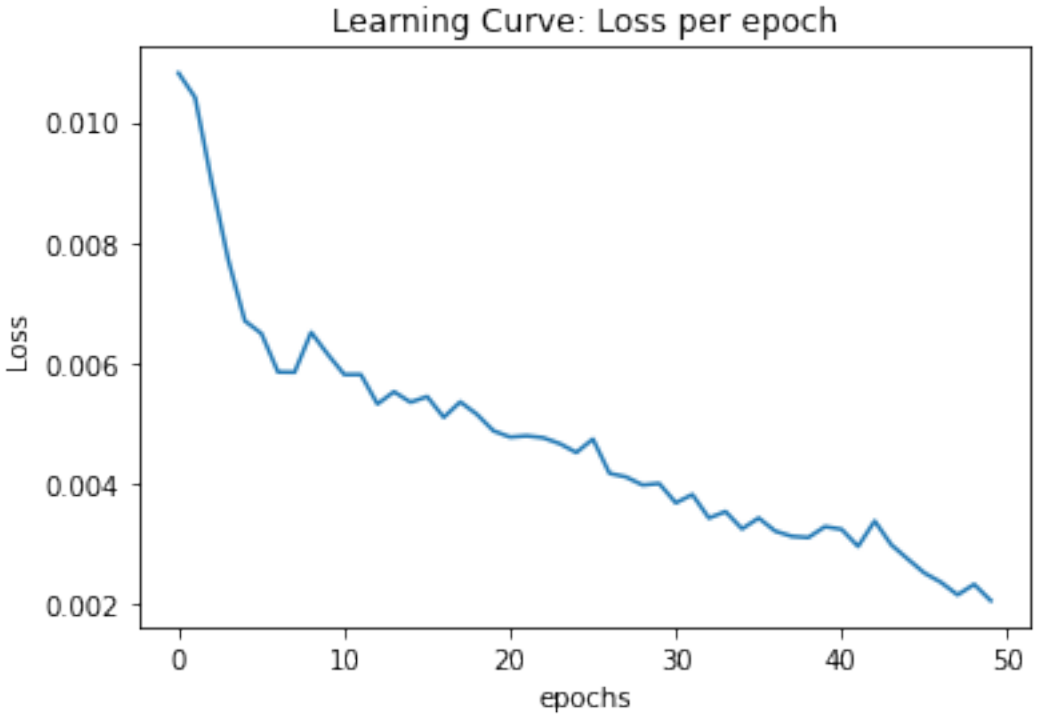
[ ]:
```
# Include the training curves for the two models.
# In our train_model function we include the option average that used here for␣
 ↪averaging the positive and the negative accuracy
print("The CNN model - we averaged the positive and negative accuracy to make␣
 ↪the coverage process more noticeable")
with open('/content/gdrive/My Drive/learning_curve_info_CNN_1.pickle', 'rb') as␣
 ↪handle:
    learning_curve_info = pickle.load(handle)
plot_learning_curve(*learning_curve_info, average=True)

print("The CNNChannel model - we averaged the positive and negative accuracy to␣
 ↪make the coverage process more noticeable")
with open('/content/gdrive/My Drive/learning_curve_info_pytorch_CNNChannel_2.
 ↪pickle', 'rb') as handle:
    learning_curve_info = pickle.load(handle)
plot_learning_curve(*learning_curve_info, average=True)
```

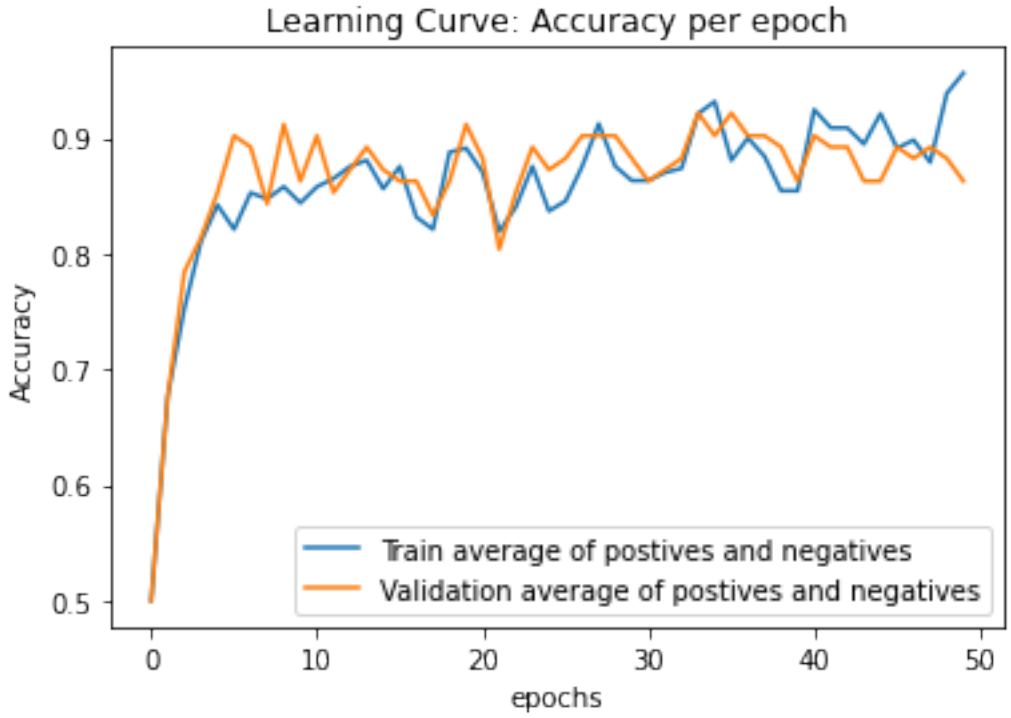
```
The CNN model - we averaged the positive and negative accuracy to make the
coverage process more noticeable
```


Learning Curve: Loss per epoch
Loss
0.0055
0.0050
0.0045
0.0040
0.0035
0.0030
0.0025
0
25
50
75
100
125
150
175
200
epochs


Learning Curve: Accuracy per epoch
Accuracy
0.9
0.8
0.7
0.6
0.5
Train average of postives and negatives
Validation average of postives and negatives
0
25
50
75
100
125
150
175
200
epochs

The CNNChannel model - we averaged the positive and negative accuracy to make the coverage process more noticeable





1.6 Question 4. Testing (15%)

1.6.1 Part (a) -- 7%

Report the test accuracies of your **single best** model, separately for the two test sets. Do this by choosing the model architecture that produces the best validation accuracy. For instance, if your model attained the best validation accuracy in epoch 12, then the weights at epoch 12 is what you should be using to report the test accuracy.

The best model based on the validation set was CNNChannel. It achieved in epoch 33 the best perfomace:

We chose our best model based on the average accuracy of negative and positive samples and it was the CNN channel in epoch 33

epoch 33. [Val Acc (all 92%), (positives 94%), (negatives: 90%)] [Train Acc (all 92%), (positive: 96%), (negatives: 88%), Loss 0.003530] saving epoch 33 as best epoch

[ ]:
```
# Write your code here. Make sure to include the test accuracy in your report
model_CNNChannel = CNNChannel(n=24).to(device)
model_CNNChannel.load_state_dict(torch.load('/content/gdrive/My Drive/
 ↪pytorch_CNNChannel_2_ckpt-32.pk'))

test_acc_pos_m, test_acc_neg_m = get_accuracy(model_CNNChannel, test_m,␣
 ↪batch_size=64)
print(f"Test men set: Total accuracy:{str(((test_acc_pos_m+test_acc_neg_m)/2) *␣
 ↪100)[:5]}, positive samples:{str(test_acc_pos_m * 100)[:5]}, negative␣
 ↪samples:{str(test_acc_neg_m * 100)[:5]}")

test_acc_pos_w, test_acc_neg_w = get_accuracy(model_CNNChannel, test_w,␣
 ↪batch_size=64)
print(f"Test woman set: Total accuracy:{str(((test_acc_pos_w+test_acc_neg_w)/2)␣
 ↪* 100)[:5]}, positive samples:{str(test_acc_pos_w * 100)[:5]}, negative␣
 ↪samples:{str(test_acc_neg_w * 100)[:5]}")
```

```
Test men set: Total accuracy:81.66, positive samples:83.33, negative
samples:80.0
Test woman set: Total accuracy:83.33, positive samples:90.0, negative
samples:76.66
```

1.6.2 Part (b) -- 4%

Display one set of men's shoes that your model correctly classified as being from the same pair.

If your test accuracy was not 100% on the men's shoes test set, display one set of inputs that your model classified incorrectly.

[ ]:
```
model_CNNChannel.eval()
data_pos = generate_same_pair(test_m)      # should have shape [n * 3, 448,␣
 ↪224, 3]
```

```
data_neg = generate_different_pair(test_m) # should have shape [n * 3, 448,␣
 ↪224, 3]
```

[ ]:
```
print("We take the 1th pair in the the positive set (same pairs set)")
xs = torch.Tensor(np.expand_dims(data_pos[0], axis=0)).transpose(1, 3)
xs = xs.to(device)
zs = model_CNNChannel(xs)
pred = zs.max(1, keepdim=True)[1]
print("it should be resulted in prediction of True, and indeed it resulted with␣
 ↪True")
print(pred)
plt.imshow(((data_pos[0] + 0.5) * 255).astype(np.uint8)) # should show 2 shoes␣
 ↪from different pairs
plt.show()
```

```
We take the 1th pair in the the positive set (same pairs set)
it should be resulted in prediction of True, and indeed it resulted with True
tensor([[1]], device='cuda:0')
```

[ ]:
```
print("We take the 18th pair in the the negative set (different_pair set)")
xs = torch.Tensor(np.expand_dims(data_neg[17], axis=0)).transpose(1, 3)
xs = xs.to(device)
zs = model_CNNChannel(xs)
pred = zs.max(1, keepdim=True)[1]
print("it should be resulted in prediction of False, but the model predict True␣
 ↪which is wrong")
```

```
print(pred)
plt.imshow(((data_neg[17] + 0.5) * 255).astype(np.uint8)) # should show 2 shoes␣
 ↪from different pairs
plt.show()
```

```
We take the 18th pair in the the negative set (different_pair set)
it should be resulted in prediction of False, but the model predict True which
is wrong
tensor([[1]], device='cuda:0')
```

1.6.3 Part (c) -- 4%

Display one set of women's shoes that your model correctly classified as being from the same pair.

If your test accuracy was not 100% on the women's shoes test set, display one set of inputs that your model classified incorrectly.

[ ]:
```
model_CNNChannel.eval()
data_pos = generate_same_pair(test_w)      # should have shape [n * 3, 448,␣
 ↪224, 3]
data_neg = generate_different_pair(test_w) # should have shape [n * 3, 448,␣
 ↪224, 3]
```

[ ]:
```
print("We take the 1th pair in the the positive set (same pairs set)")
xs = torch.Tensor(np.expand_dims(data_pos[0], axis=0)).transpose(1, 3)
xs = xs.to(device)
zs = model_CNNChannel(xs)
```

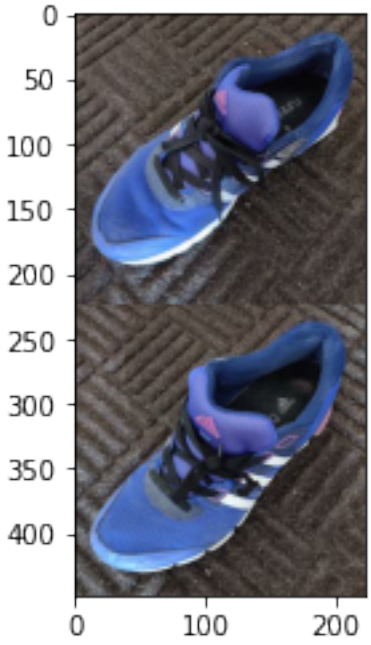
```
pred = zs.max(1, keepdim=True)[1]
print("it should be resulted in prediction of True, and indeed it resulted with␣
 ↪True")
print(pred)
plt.imshow(((data_pos[0] + 0.5) * 255).astype(np.uint8)) # should show 2 shoes␣
 ↪from different pairs
plt.show()
```

```
We take the 1th pair in the the positive set (same pairs set)
it should be resulted in prediction of True, and indeed it resulted with True
tensor([[1]], device='cuda:0')
```

[ ]:
```
print("We take the 9th pair in the the positive set (same pairs set)")
xs = torch.Tensor(np.expand_dims(data_pos[8], axis=0)).transpose(1, 3)
xs = xs.to(device)
zs = model_CNNChannel(xs)
pred = zs.max(1, keepdim=True)[1]
print("it should be resulted in prediction of True, but the model predict False␣
 ↪which is wrong")
print(pred)
plt.imshow(((data_pos[8] + 0.5) * 255).astype(np.uint8)) # should show 2 shoes␣
 ↪from different pairs
plt.show()
```

```
We take the 9th pair in the the positive set (same pairs set)
it should be resulted in prediction of True, but the model predict False which
```

```
is wrong
tensor([[0]], device='cuda:0')
```



[4]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
!pip install pypandoc
from google.colab import drive
drive.mount('/content/gdrive')
!cp gdrive/My\ Drive/Colab\ Notebooks/Assignment3.ipynb ./
# /content/drive/MyDrive/Colab Notebooks/Assignment3.ipynb
!jupyter nbconvert --to=pdf "Assignment3.ipynb"
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
pandoc is already the newest version (1.19.2.4~dfsg-1build4).
texlive is already the newest version (2017.20180305-1).
texlive-latex-extra is already the newest version (2017.20180305-2).
texlive-xetex is already the newest version (2017.20180305-1).
0 upgraded, 0 newly installed, 0 to remove and 37 not upgraded.
Requirement already satisfied: pypandoc in /usr/local/lib/python3.7/dist-
packages (1.7.2)
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
cp: cannot stat 'gdrive/My Drive/Colab Notebooks/Assignment3.ipynb': No such
file or directory
[NbConvertApp] WARNING | pattern u'Assignment3.ipynb' matched no files
```

```
This application is used to convert notebook files (*.ipynb) to various other
formats.

WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
-------

Arguments that take values are actually convenience aliases to full
Configurables, whose aliases are listed on the help line. For more information
on full configurables, see '--help-all'.

--execute
    Execute the notebook prior to export.
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
--no-input
    Exclude input cells and output prompts from converted document.
    This mode is ideal for generating code-free reports.
--stdout
    Write notebook output to stdout instead of files.
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
    relevant when converting to notebook format)
-y
    Answer yes to any questions instead of prompting.
--clear-output
    Clear output of current file and save in place,
    overwriting the existing notebook.
--debug
    set log level to logging.DEBUG (maximize logging output)
--no-prompt
    Exclude input and output prompts from converted document.
--generate-config
    generate default config file
--nbformat=<Enum> (NotebookExporter.nbformat_version)
    Default: 4
    Choices: [1, 2, 3, 4]
    The nbformat version to write. Use this to downgrade notebooks.
--output-dir=<Unicode> (FilesWriter.build_directory)
    Default: ''
    Directory to write output(s) to. Defaults to output to the directory of each
    notebook. To recover previous default behaviour (outputting to the current
```

```
    working directory) use . as the flag value.
--writer=<DottedObjectName> (NbConvertApp.writer_class)
    Default: 'FilesWriter'
    Writer class used to write the  results of the conversion
--log-level=<Enum> (Application.log_level)
    Default: 30
    Choices: (0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL')
    Set the log level by value or name.
--reveal-prefix=<Unicode> (SlidesExporter.reveal_url_prefix)
    Default: u''
    The URL prefix for reveal.js (version 3.x). This defaults to the reveal CDN,
    but can be any url pointing to a copy  of reveal.js.
    For speaker notes to work, this must be a relative path to a local  copy of
    reveal.js: e.g., "reveal.js".
    If a relative path is given, it must be a subdirectory of the current
    directory (from which the server is run).
    See the usage documentation
    (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-
    slideshow) for more details.
--to=<Unicode> (NbConvertApp.export_format)
    Default: 'html'
    The export format to be used, either one of the built-in formats
    ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf',
    'python', 'rst', 'script', 'slides'] or a dotted object name that represents
    the import path for an `Exporter` class
--template=<Unicode> (TemplateExporter.template_file)
    Default: u''
    Name of the template file to use
--output=<Unicode> (NbConvertApp.output_base)
    Default: ''
    overwrite base name use for output files. can only be used when converting
    one notebook at a time.
--post=<DottedOrNone> (NbConvertApp.postprocessor_class)
    Default: u''
    PostProcessor class used to write the results of the conversion
--config=<Unicode> (JupyterApp.config_file)
    Default: u''
    Full path of a config file.

To see all available configurables, use `--help-all`

Examples
--------

    The simplest way to use nbconvert is

    > jupyter nbconvert mynotebook.ipynb
```

```
    which will convert mynotebook.ipynb to the default format (probably HTML).

    You can specify the export format with `--to`.
    Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'rst', 'script', 'slides'].

    > jupyter nbconvert --to latex mynotebook.ipynb

    Both HTML and LaTeX support multiple output templates. LaTeX includes
    'base', 'article' and 'report'.  HTML includes 'basic' and 'full'. You
    can specify the flavor of the format used.

    > jupyter nbconvert --to html --template basic mynotebook.ipynb

    You can also pipe the output to stdout, rather than a file

    > jupyter nbconvert mynotebook.ipynb --stdout

    PDF is generated via latex

    > jupyter nbconvert mynotebook.ipynb --to pdf

    You can get (and serve) a Reveal.js-powered slideshow

    > jupyter nbconvert myslides.ipynb --to slides --post serve

    Multiple notebooks can be given at the command line in a couple of
    different ways:

    > jupyter nbconvert notebook*.ipynb
    > jupyter nbconvert notebook1.ipynb notebook2.ipynb

    or you can specify the notebooks list in a config file, containing::

        c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

    > jupyter nbconvert --config mycfg.py
```

[ ]: