Note: The answers for some of the numerical parts and graphs may vary because of the shuffling used in valid sometimes to prevent overtraining.

[Q1] What is the correct kernel size in the average pooling layer? Explain your answer.

Assuming that there is a flattening operation after the average pooling layer, the kernel size of the average pooling layer should be with stride as 1. The formula can be used to find the corresponding size of the output after each layer is applied. So, for example, after the 1st convolutional layer is applied, with the help of our formula, the output is of size but there are 32 of those. We do the similar thing after every layer, and after max pool, the output becomes of size and remains that way until we flatten it and apply average pooling with kernel size and stride = 1 to get output of size .

[Q2] What is dropout in deep learning? Why does it work? Explain it briefly.

Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly. This means that their contribution to the activation of downstream neurons is temporarily removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.
As a neural network learns, neuron weights settle into their context within the network. Weights of neurons are tuned for specific features providing some specialization. Neighboring neurons start to rely on this specialization, which if taken too far can result in a fragile model too specialized to the training data or in other words oversampling. This reliance on context for a neuron during training is referred to complex co-adaptations.

It works because if neurons are randomly dropped out of the network during training, then other neurons will have to step in and handle the representation required to make predictions for the missing neurons. This results in multiple independent internal representations being learned by the network. The effect is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data.

[Q3] What is the total number of trainable parameters in this Siamese network model?

The total number of trainable parameters used in the model are 1842465.

[Q4] If we replace the absolute difference aggregation function by concatenation (i.e., concate- nating the hidden state vectors of the two input images into one before feeding into the fully connected layers), what is the change in the total number of trainable parameters when compared to the original setting?

The total number of trainable parameters when replacing the aggregation function by concatenation becomes 2629921. The change in the total number of parameters is That is the number of parameters increases by 787456 which makes sense because there is a concatenation instead of difference for comparison. This is when d_in = 1024 and the two fully connected layers are fc1 = nn.Linear(1024,1024) and fc2 = nn.Linear(1024,1). The answer may vary if something like fc1 = nn.Linear(1024,512) and fc2 = nn.Linear(512,1) is used.

[Q5] Paste the screenshot of the training and validation loss curves and report the final training and validation losses obtained. According to your plot, when should you stop training? Why?

One important thing to be noted is that before calculating the loss and training, the shuffle in train and valid is turned true for better results.

In Figure 1, it is shown that the training loss decreases as the epochs increase and the training loss per epoch represented by individual lines is calculated per 10 steps. It looks like an ideal graph since the training loss decreases after each epoch as intuitively imagined.

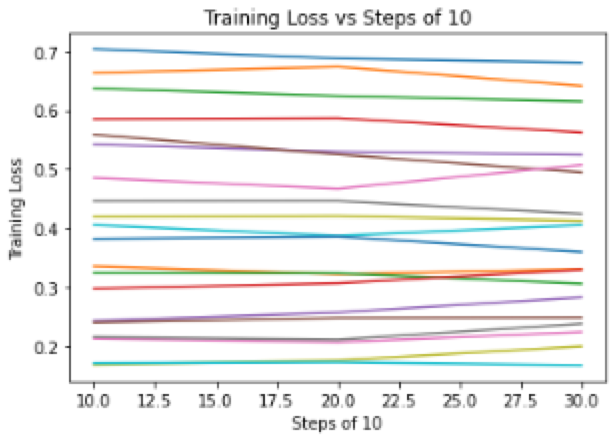


Figure 1

In Figure 2, the validation loss is calculated per 1 step. Each line represents validation loss for different epochs similar to the previous explanation. Again, the general trend is that validation loss decreases over time but it's not very stable in the steps which is just the way the model works and predicts. For sanity check, the model was also implemented for a small dataset to see if it overfits and it did. Meaning that the model in itself does not have any problem.

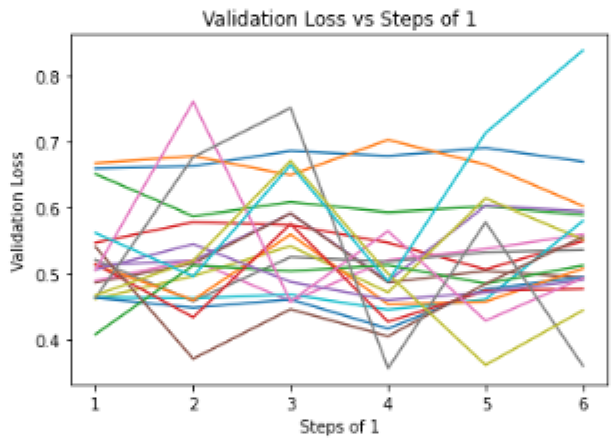


Figure 2

In Figure 3, all the epochs from figure 1 are concatenated together and just like in 1, the step size is 10.

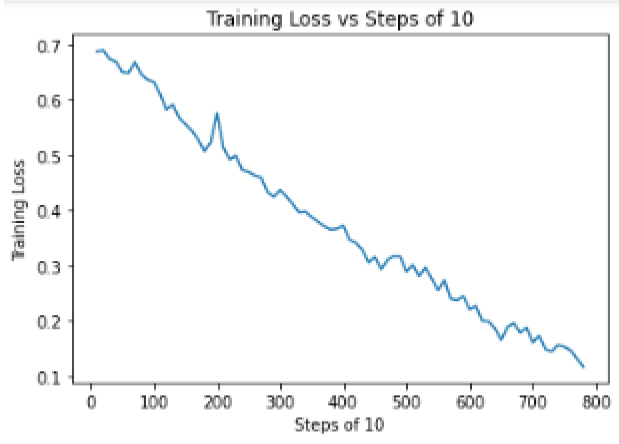


Figure 3

In Figure 4, all the epochs from figure 2 are concatenated together and just like in 2, the step size is 1.

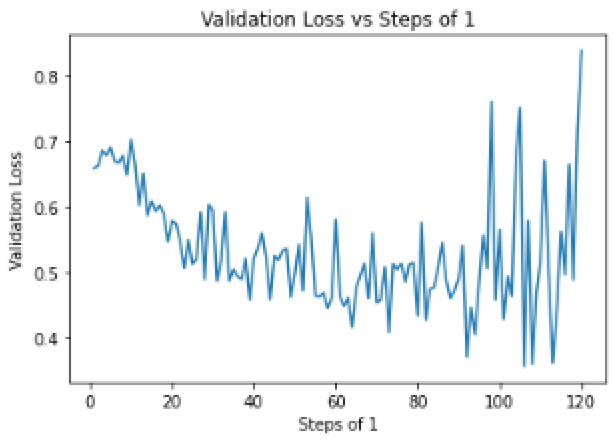


Figure 4

Figure 5 gives us a wholesome picture of how our algorithm is working. The orange curve represents the training loss and the blue represents the validation curve. Validation loss here is taken after every 10 steps unlike the validation in previous figures and as is training which is the same as figure 3. Their loss is aggregated over 10 steps and then averaged after every 10 steps. In total, there are around 780 steps taken.

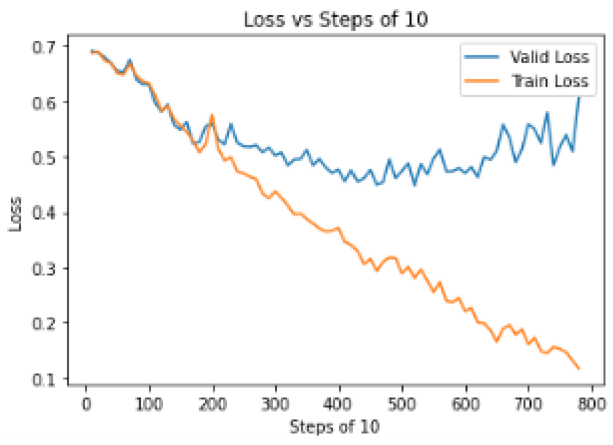


Figure 5

According to the plots, it would be better if the training is stopped early at around 410 steps. As we can see from figure 5, the loss for the validation set starts deviating from the training loss. This indicates that the training set is starting to memorize its instances and oversampling. That is not something that is ideal or we want for our model. Therefore, as the training loss decreases, the validation loss increases which almost nullifies if not deters the work that is put in further training. Below is figure 6, that describes how valid and train accuracy fluctuates as the model goes through training and validation. After about 13 epochs, the valid accuracy stabilizes but due to more episodes of training, the training accuracy keeps increasing without any change in its ability to predict output.

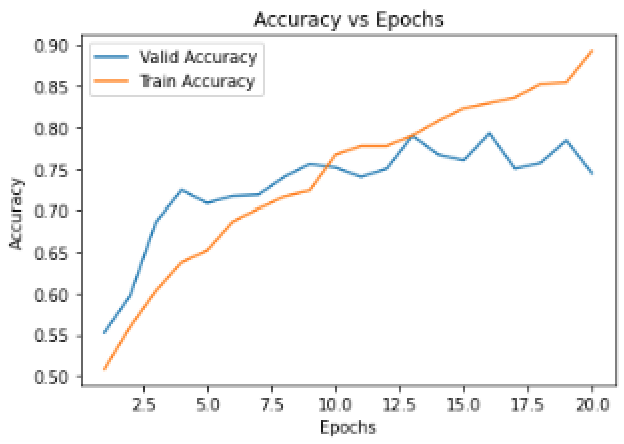
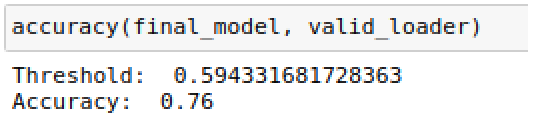


Figure 6

[Q6] Report the optimal threshold θ and the corresponding validation accuracy.

The optimal threshold θ is 0.59 and the corresponding validation accuracy for that threshold is 76.



```
accuracy(final_model, valid_loader)
```

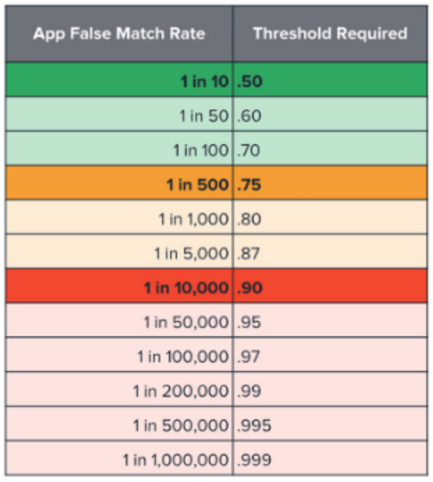
```
Threshold:  0.594331681728363
Accuracy:  0.76
```

[Q7] Suppose you are given a photo application used to identify similar-looking family members and a face authentication application used for mobile banking. Should you set a relatively high or low threshold for each of the applications? Why?

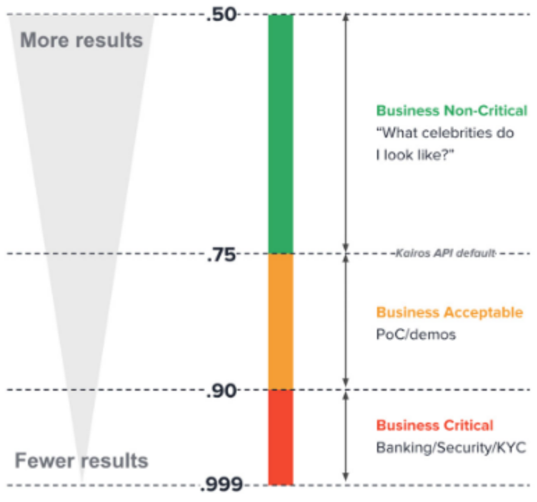
For face-authentication used for mobile banking, security is the most important factor. It should be only the user and no one else should be able to enter the application, not even the family members who might resemble the user otherwise it violates cybersecurity's CIA principle i.e. confidentiality, integrity and availability. For all these reasons, the threshold should be relatively high. Here, we define a high threshold as the system being more sure and accurate about the match between two faces. This relatively high threshold will validate that the person is the actual user and not someone identical or resembles the user.

On the other hand, in case of detection of similarity among family members, the threshold can be put relatively to the lower end of the spectrum since the application is used to determine how similar 2 family members or 2 people are from each other. If their features match, then the application returns that those two people are the same. For this reason, a relatively low threshold can be used.

Below is the figure to explain visually the usage of high and low threshold in practical uses.



App False Match Rate	Threshold Required
1 in 10	**.50**
1 in 50	.60
1 in 100	.70
1 in 500	**.75**
1 in 1,000	.80
1 in 5,000	.87
1 in 10,000	**.90**
1 in 50,000	.95
1 in 100,000	.97
1 in 200,000	.99
1 in 500,000	.995
1 in 1,000,000	.999



(figure from kairos website)

[Q8] Describe your changes and compare the validation performance with the original setting.

For the changes, I have chosen part A and C i.e. Optimization and Data augmentation to improve my model. The goal of the model is to produce the higher validation accuracy compared to the model constructed in previous tasks while keeping in mind that we do not overtrain/undertrain the model.
In this scenario, when using the data loader, the parameter shuffle is set to True to avoid over/under training and make our model more robust. Following will be the major changes made to the model in the corresponding sections to improve the model.
Only changed parameters are numbered.

A. Optimization:

Model Used: Final_Siamese_Network() (same as one used in the previously constructed model)

Changed:

1. num_epochs = 20 (although 27 also works)

Optimizer used:

Changed:

2. AdamW (a variant of Adam proposed in Decoupled Weight Decay Regularization).

Parameters:

Final_Siamese_Network().parameters(),

Changed/Added:

3. lr = 0.0006
4. weight_decay=0.00003
5. amsgrad=True

After adding this optimization, the accuracy of the model was increased to a whopping 78% from 76% as mentioned in Q[6].

Once run, all these will be saved in the corresponding "model1_set{setting_num}{num}_log{model_num}.pt" and can be cross examined.

Parameters for Model A

Model1: (with default settings from the previous network)

To be noted that upon the calculation of these models, the validation shuffle was true.

1. num_epochs: 15

   Threshold: 0.48876954317092896

   Accuracy: 0.7514285714285714
2. num_epochs = 25

   Threshold: 0.4274408221244812

   Accuracy: 0.7671428571428571
3. num_epochs = 30

   Threshold: 0.3690468370914459

   Accuracy: 0.7757142857142857

Model2: (with lr = 0.0006 and weight_decay = 0.00003)

1. Optimizer = Adam

   Threshold: 0.5730909109115601

   Accuracy: 0.5485714285714286
2. Optimizer = AdamW

   Threshold: 0.5875911116600037

   Accuracy: 0.5571428571428572
3. Optimizer = SGD

   Threshold: 0.49105361104011536

   Accuracy: 0.5171428571428571

Model3: (with default settings from the previous network)

1. Learning rate = 0.002

   Threshold: 0.6136685609817505

   Accuracy: 0.5528571428571428
2. Learning rate = 0.004

   Threshold: 0.6082062721252441

   Accuracy: 0.5214285714285715
3. Learning rate = 0.0006

   Threshold: 0.5966874361038208

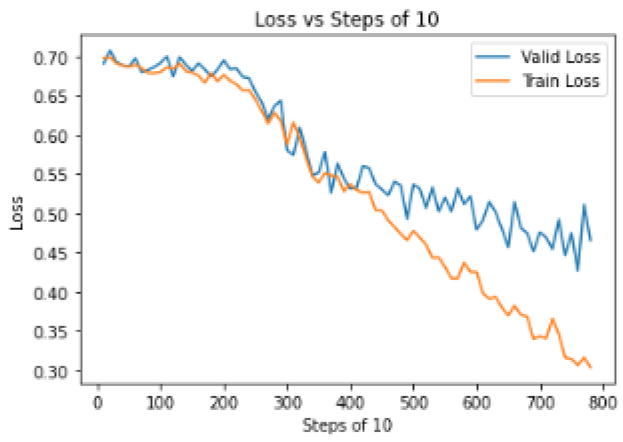
   Accuracy: 0.5585714285714286

Best of settings 1, 2, 3 and a few other parameters combined, we get optimizer for our model 1 as:

$$optimizer = optim.AdamW(final\_model.parameters(),\ lr = 0.006,$$
$$weight\_decay = 0.00003, amsgrad = True)$$

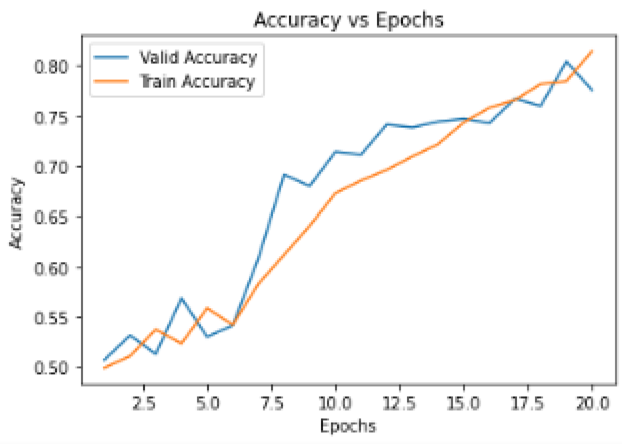
Threshold: 0.594331681728363

Accuracy: 0.77



Model A Train vs Validation Loss

This represents that there is perfect almost collaboration between validation and training loss unlike our previous model.



Parameters for Model C

1.

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Grayscale(num_output_channels=1),
    transforms.Normalize((0.5, ), (0.5, )),
    transforms.RandomPerspective(), #
    transforms.CenterCrop((32,32))
 ])
```

2.

```
transforms.functional.adjust_saturation(img1, saturation_factor=1.5)
#Adjusts the saturation of the images
```

3.

```
transforms.functional.adjust_sharpness(img2, sharpness_factor=1.5)
#Adjust the sharpness of the images
```

These are the transformations applied to the model to help improve its accuracy. Now, I have to address that the Random Perspective parameter might perform a little randomly as its name might suggest, however because sometimes we train for a high number of epochs, we need this randomness to dial down and not overtrain. Also, the architecture of the Siamese_Network has been changed in Siamese_Network_Modified where the transformations are actually called. Further, centercrop crops the picture from the center, so only the face is visible and being compared rather than sometimes also the surroundings in the previous network. On top of that, to make it clearer between dark and light coloured people, the sharpness, saturation and brightness has also been increased to the images. All these factors combined moves up our accuration of validation set by 1% i.e. from 76% to almost 77%.

The final model used to predict the results is mainly model A, although model B also gives out good results, it seemed better to use some part of B and A to calculate the prediction of the set.