MSc in Data Science

Deep Learning

Face verification system

Φώτης Παπαδήμας
AM: **2022202204022**
email: ***dit2222dsc@go.uop.gr***

Γιώργος Φασάκης
AM: **2022202204011**
email: ***dit2211dsc@go.uop.gr***

Introduction

The aim of our project was to create a face verification system. We split the problem in 3 parts.

1)Firstly we created the videotoframes.py file. This is a python file that is used to get photos from the person we want our models to be able to detect.

2) We created files such as transfer_learning-custom_cnn.py and triplet_loss_online_training_clean.py. In these files the models are trained to detect a person.

3)Thirdly in the Face_Browser.py we open the camera of the computer ,use the models we trained previously and if the model detects the correct person it opens a steam in the browser.

To train the networks we used 4 different techniques. The first was to create a custom CNN network and train it from scratch. Then we used transfer learning from resnet 18 and compared the differences. The next way we tried was to create a triplet loss network. The first two techniques are similar. The CNN networks try to classify an image based on data that we have proved them. The triplet loss models embed the images we provide as vectors. By comparing the vectors using some similarity measure they can be used to classify images.

For the project we created a virtual environment in our local machines. The requirement file of the virtual environment is uploaded in the GitHub repository of the project.

Some technical difficulties arose. One was the path used in the files. While we tried to use relative paths the virtual environment interpreter from Visual studio code would always start from the user file. We have initiated the paths used in the GitHub repository from the deeep learning file but it is possible to not work so this could need some tweaking. Another possible problem that can happen is the use of the model_weights.pth file in the Face_Browser.py. During the training of the model we used a gpu in order to speed up the training. If the computer loading the model does not have cuda the weights may not load. Finally because in the training the camera of the computer,if used with another camera it could be that its performance is not satisfactory.

Custom CNN / transfer learning

As stated before for our CNN's we used a custom and a model based on resnet18 with transfer learning.

Data Problems:

1)The first problem we came across was the data where to find them and how many. For models such as CNN's in order to train them from scratch and achieve high accuracy a lot more data than those we had available were required. However we used on the CNN the data we had available and considering how limited they were, we got satisfactory results.

2) Random people data: We had to get data in our hands from various people in order to train our model to be able to differentiate between the person we want to identify and a random stranger. We downloaded images from face image datasets from the web as it is pretty easy to do so.

3) A problem we came across was that we had much less data from the person we wanted to identify that the random strangers. The persons used in the context of the project were our own selves. We took photos of ourselves in different conditions in order to help the model to generalize better. Then we limited the random people data to match the data of ourselves so that the dataset is not imbalanced.

4)Another problem that we came across early was that the models could easily distinguish between the photos from our cameras and the photos from the web. In order to overcome this obstacle we used various friends and family members photos that were put along the photos of the people we got from the web. This had good results and helped the model to generalize better. After using these data too the models were able to distinguish between other people and ourselves pretty well.

5)The color of the images was also another problem we came across. When we created the system and used it was possible for the image to pass as the person in the question if the colors were akin to the skin color of the person that we wanted the system to recognize. In order to alleviate this problem we put some random hue, tone transformations in our dataset.

6) We resized the images to keep the center of each image. This helped with computing the model

7)We applied a random horizontal flip in order to help the model better identify humans. We could use random rotations but this had better results.

8)Finally we normalized the images before passing them to our model to help the model converge.

We named in our dataset the class **person** for the person to be recognized and **humans** the class with all the other people.

Networks:

Custom Network :

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3,6,5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 53 * 53, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 2)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

As our model we used the above model it has two convolutional layers, two pooling layers and 3 fully connected layers that end up in the two classes we want to predict **person** or **humans.**

Transfer learning :

The transfer learning model is just the resnet 18 but we have frozen all the layers except the last two that we will retrain.

```
layers = 18 #18 layer exei to resnet18
layercount = 0
for param in model_conv.parameters():
    if layercount == layers - 3 :
        break
    param.requires_grad = False
    layercount+=1

num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)
```

Optimizer used : Stochastic gradient descend, Learning rate : 0.001
Loss function : Cross Entropy loss ,Training epoch : 25, Batch size : 4

The hope here is that the model in the previous layers has learnt to distinguish humans and by retraining the last layers we can make it distinguish between people.

For both the models we used the Cross entropy loss as the loss function. As optimizer the stochastic gradient descent was chosen with a learning rate of 0.001.

Results:

The ways face identification systems are assesed in the literature is

False Match Rate (FMR) or known as False positive rate(FPR):

FMR = F P/N = F P/(F P + T N)

False Non-Match Rate (FNMR) we use in our graphs below (FNR):

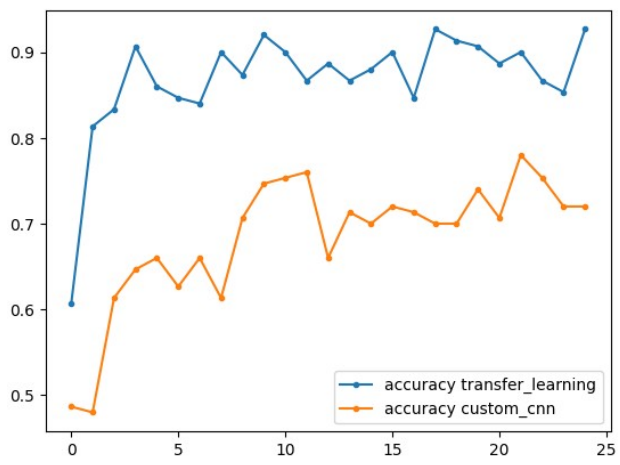
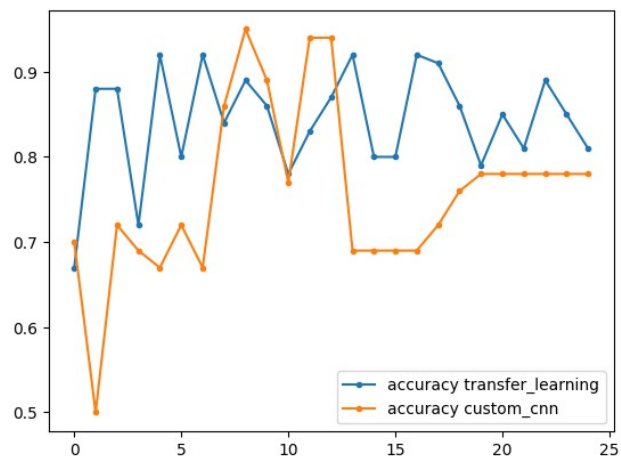
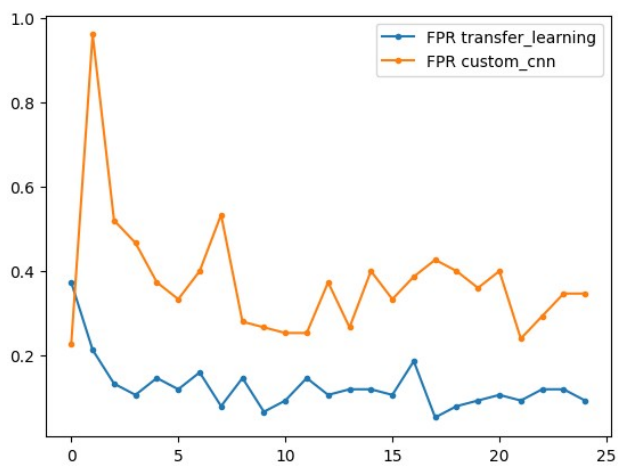
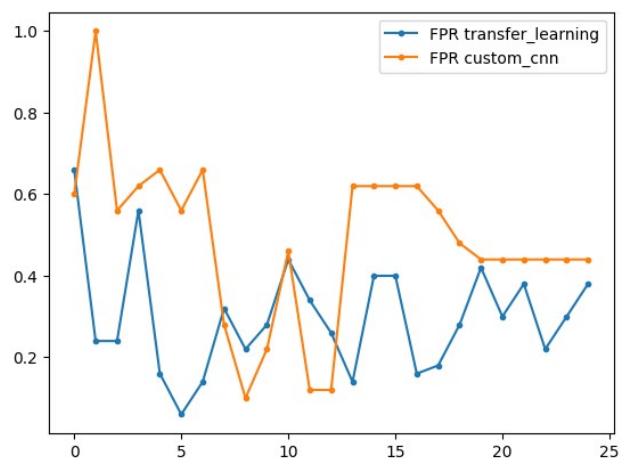
FNMR = F N/(F N + T P)

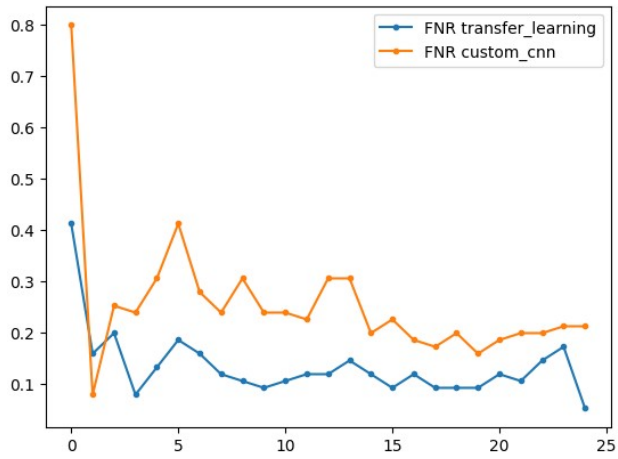
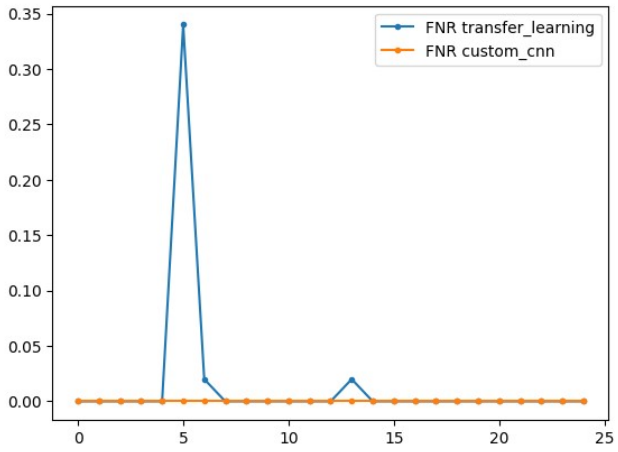
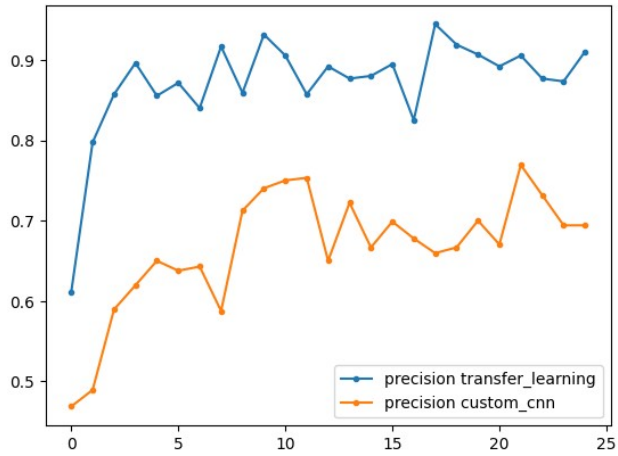
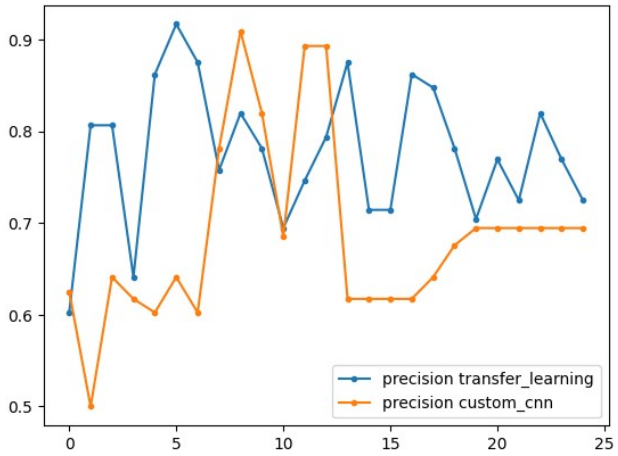
and the accuracy:

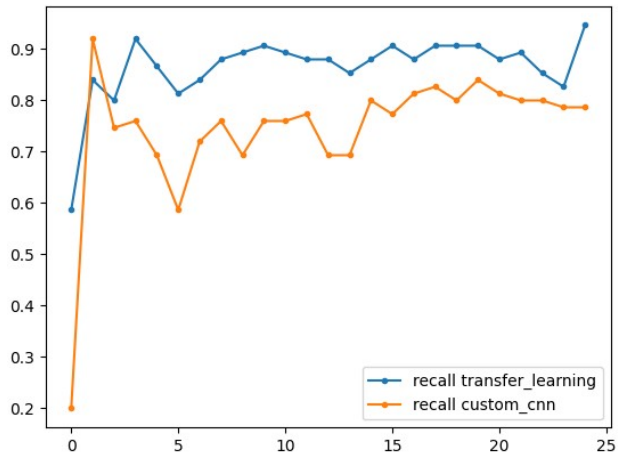
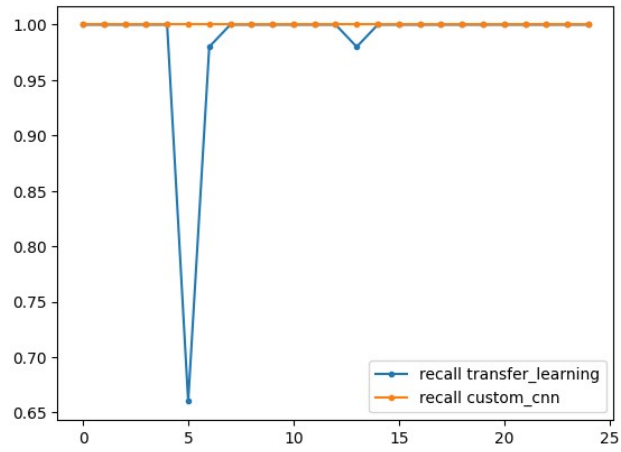
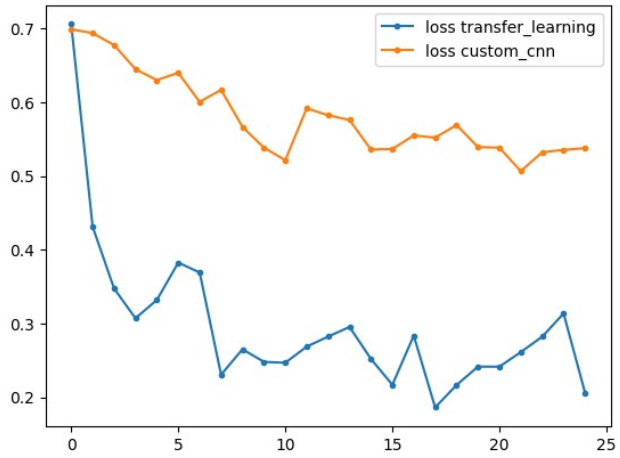
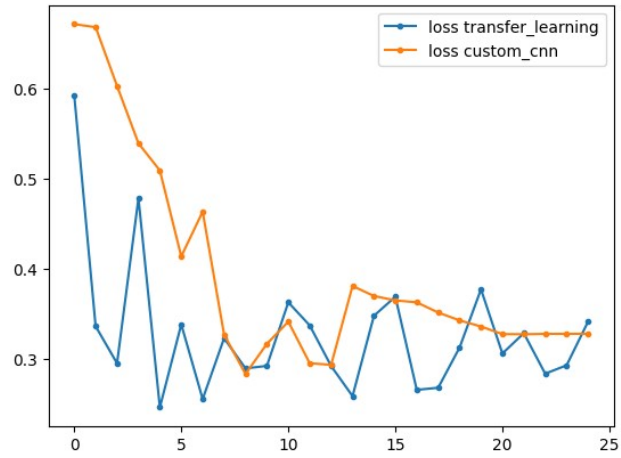
Acc = (T P + T N)/(T P + T N + F P + F N

along with these we checked precision and recall for the test and train phases.

We run the two models and we got these results across the 25 epochs we trained them.

Train	Test
	
	

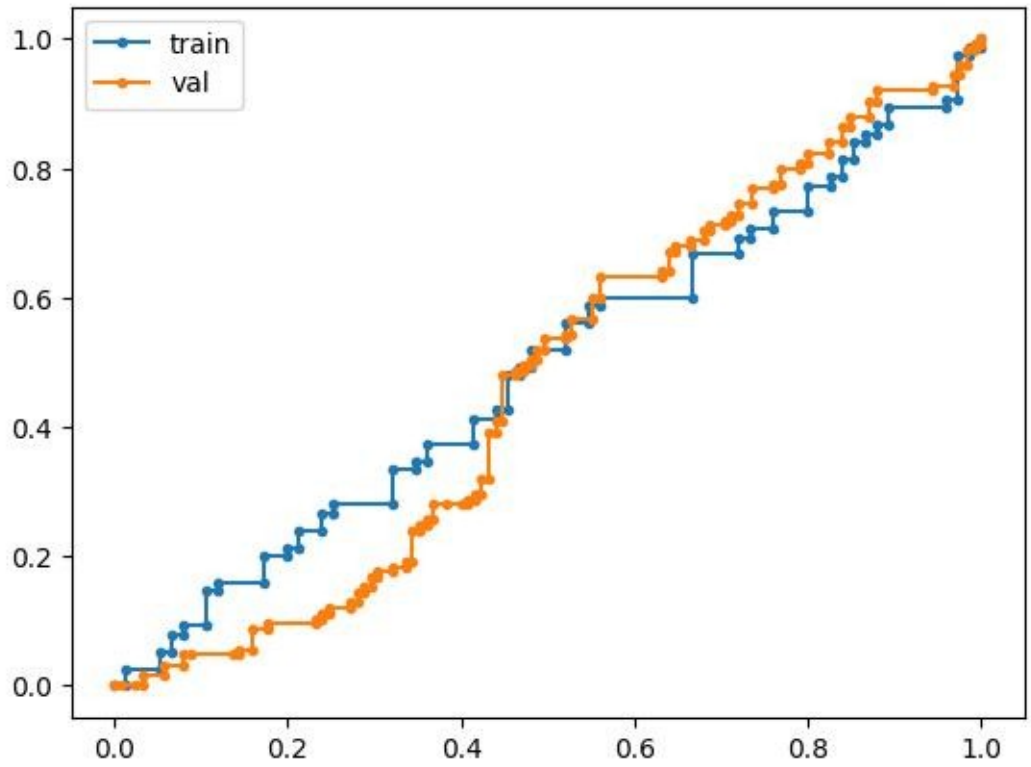
Train	Test
 FNR transfer_learning FNR custom_cnn	 FNR transfer_learning FNR custom_cnn
 precision transfer_learning precision custom_cnn	 precision transfer_learning precision custom_cnn

Train	Test
	
	

From the above we see that in the training phase the transfer learning model has a better performance. However in the test phase the two models do not have that big of a difference. The accuracy of the transfer learning model is higher than the custom CNN's especially in the first epochs but then the CNN starts to catch up. Where there is difference thought and it matters is in the False positives. The transfer learning model has better FPR in average even though for two cases the CNN has bette FPR. FPR having less false positives means the system is more secure. For this reason we used the Transfer learning model in the Face_Browser.py file to open the browser.

In our selection of model we chose the model which had the best test accuracy and it was for the epoch 6. The model had accuracy 0.92, FPR 0.14 , FNR: 0.02.

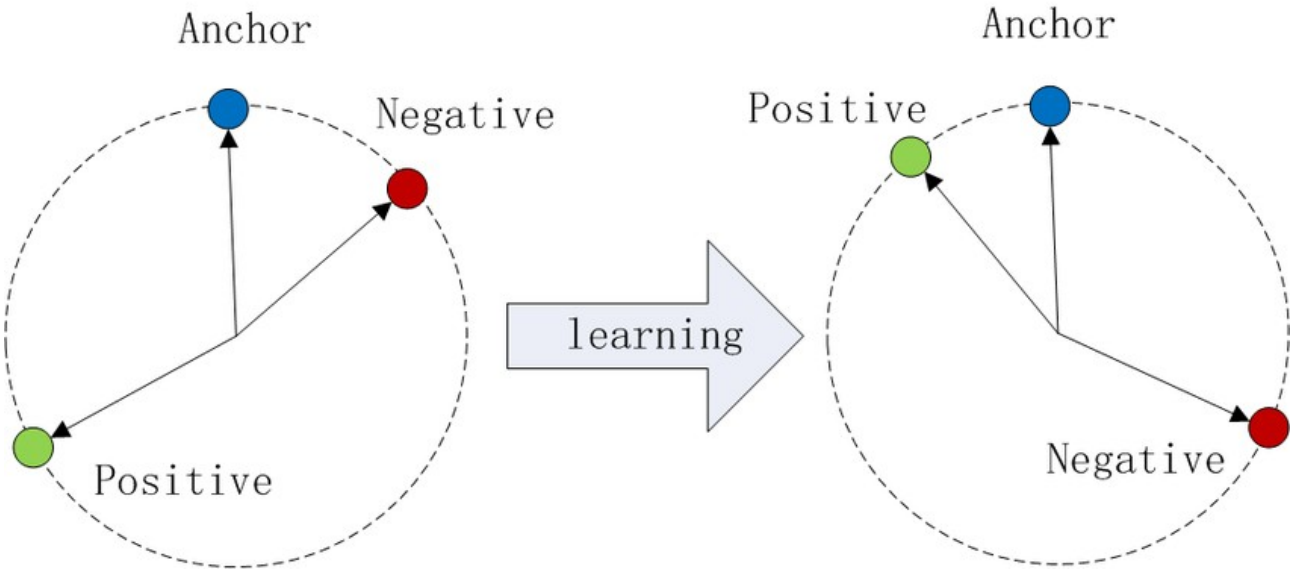
Below is the ROC curve of the system.



After training and selecting our model we saved it. Then we used it to recognize our faces. The Face_Browser.py takes 10 pictures loads the model we saved and decides for each photo in which class it belongs. If for the 70% of the photos the answer is that it is the person to be identified it opens the browser. The 70% was selected with the fact that classifying something not seen before by the model had a 50% chance to be put in either class. As such a 70% threshold should stop a random images from opening the program.

Triplet loss

We tried to create a triplet loss model as well. The idea here is that the Neural network works as an embedding machine. It embed each photo to a vector. We aim to minimize the distance between pairs of vectors of the same class and maximize the distance between pairs of vectors of different class. As such it would be easy to classify each image by which cluster of vectors it belongs to.



The way this is achieved is by a loss function that takes as input three vectors. The anchor, a positive vectors and a negative vector. The loss function aim is for the anchor-positive distance to be smaller than the anchor-negative distance. In order for the model to not output trivial results (f(anything) = 0) a constant a is used.

$$\mathcal{L}(A, P, N) = \max(\| \mathrm{f}(A) - \mathrm{f}(P) \|_2 - \| \mathrm{f}(A) - \mathrm{f}(N) \|_2 + \alpha, 0)$$

Choosing triplets in random results in very large datasets. If we have n data originally we end up with a n*n*n dataset. For this reason techniques for triplet selection are implemented. We used the **FaceNet** approach of semihard online triplet mining and the one used in **A Conditional Triplet Loss for Few-shot Learning and its Application to Image Co-Segmentation.**

The first approach works in each epoch and in each batch and each class in the batch. There a point is selected as anchor. Based on the distance from it the hard positive and hard negative points can be defined. The hard positive point is the argmax(euclidian_distance(anchor,poisitives)). This means that it is the the point that is the same class as the anchor and is the furthest from it. The hard negative is the point that is clossest to the anchor from the data that are not in the same class as the anchor. It is argmin(euclidian_distance(anchor,negatives)). The triplets are created by using the anchor,hard positive and hard negatives. This is by taking as anchor eacah point in each class. Because the hard negatives and hard positives can lead to bad local minima the semi-hard negatives are used usually instead for the hard negatives. The semi hard negatives are the points that are in a different class from the anchor and fullfill

$$\|f(x_i^a) - f(x_i^p)\|_2^2 < \|f(x_i^a) - f(x_i^n)\|_2^2 \ .$$

The second approach selects triplets in random and applies penalties to bad triplets and rewards to good triplets.

Xa = anchor points
Xp = points in the same class as anchor
Xn = points in a different class from anchor
a = hyperparameter we choose
m = margin
k = real in (0,1)

The **penalty** is put for the triplets that satisfy:

$$\text{dis}\left(x_a^z, x_p^z\right) > \text{dis}\left(x_a^z, x_n^z\right) + m$$

And the loss function becomes

$$\text{loss}\left(\mathcal{TW}_z\right) = \underbrace{\text{Max}\left[\text{dis}\left(x_a^z, x_p^z\right) - \text{dis}\left(x_a^z, x_n^z\right) + m, 0\right]}_{\text{loss}} + \underbrace{\alpha \times \left[\frac{\text{dis}\left(x_a^z, x_p^z\right) + \text{dis}\left(x_a^z, x_n^z\right)}{2}\right]}_{\text{penalty}}$$

The **reward** is put for triplets that satisfy:

$$\varepsilon < \text{dis}\left(x_a^z, x_p^z\right) - \text{dis}\left(x_a^z, x_n^z\right) + m \leq 2\varepsilon \qquad \epsilon = km \text{ where } 0<k<1.$$

And the loss function becomes

$$\text{loss}\left(\mathcal{TB}_z\right) = \underbrace{\text{Max}\left[\text{dis}\left(x_a^z, x_p^z\right) - \text{dis}\left(x_a^z, x_n^z\right) + m, 0\right]}_{\text{loss}} - \underbrace{\alpha \times \left[\frac{\text{dis}\left(x_a^z, x_n^z\right) - \text{dis}\left(x_a^z, x_p^z\right)}{2}\right]}_{\text{reward}}$$

The metrics we used to assess the system are the ones from the Face net paper.

Psame are the pairs of vectors that are of the same class.
Pdiff are the pairs of vectors that are of different class.
D(xi,xj) is the distance between the two points.
And d is the distance we choose.

The points that were correctly classified are :

$$\mathrm{TA}(d) = \{(i,j) \in \mathcal{P}_{\text{same}}, \text{with } D(x_i, x_j) \leq d\} .$$

The points that were incorectly classified are:

$$\mathrm{FA}(d) = \{(i,j) \in \mathcal{P}_{\text{diff}}, \text{with } D(x_i, x_j) \leq d\}$$

The paper defined the validation rate VAL(d) and the false accept rate
FAR(d) for a given face distance d are then defined as

$$\mathrm{VAL}(d) = \frac{|\mathrm{TA}(d)|}{|\mathcal{P}_{\text{same}}|} , \quad \mathrm{FAR}(d) = \frac{|\mathrm{FA}(d)|}{|\mathcal{P}_{\text{diff}}|} .$$

Between the methods used we saw that the one described in **A Conditional Triplet Loss for Few-shot Learning and its Application to Image Co-Segmentation.** Had better results. This could be due to that triplet mining is slower to converge. Another reason could be that in the second method we created randomly data. Instead of creating n*n*n pairs we created batches of anchor,positive,negative triplets chosen at random that was of size 6*n. The 6*n size was chosen arbitrarily. On the other hand in the first method a lot of triplets are ruled out by the definition of hard negatives/semi-hard negatives and hard positives. So in order to get a good accuracy more sampling of the data is required and as such more epochs.

Our setup: In both cases we used the same net for the embeddings

```
class SiameseNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3,6,5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.conv3 = nn.Conv2d(16, 32, 5)
        self.conv4 = nn.Conv2d(32, 64, 5)

        #self.fc1 = nn.Linear(64 * 2 * 2, 120)
        #self.fc1 = nn.Linear(64 * 8 * 8, 120)
        self.fc1 = nn.Linear(64 * 15 * 15, 256)
        self.fc2 = nn.Linear(256, 128)

    def forward(self, arg):
        output = self.pool(F.relu(self.conv1(arg)))
        #print(output.size())
        output = self.pool(F.relu(self.conv2(output)))
        #print(output.size())
        output = self.pool(F.relu(self.conv3(output)))
        #print(output.size())
        output = self.pool(F.relu(self.conv4(output)))
        #print(output.size())
        output = torch.flatten(output, 1) # flatten all dimensions except batch
        #print(output.size())
        output = F.relu(self.fc1(output))
        output = F.relu(self.fc2(output))
        return output
```

The optimizer used was Adam

We trained for 75 epochs

The learning rate was 0.005

In the FaceNet model the TripletMarginLoss loss function was used

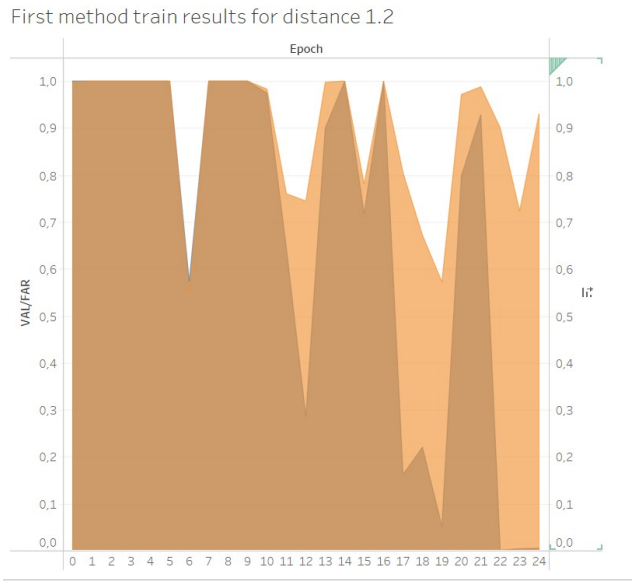
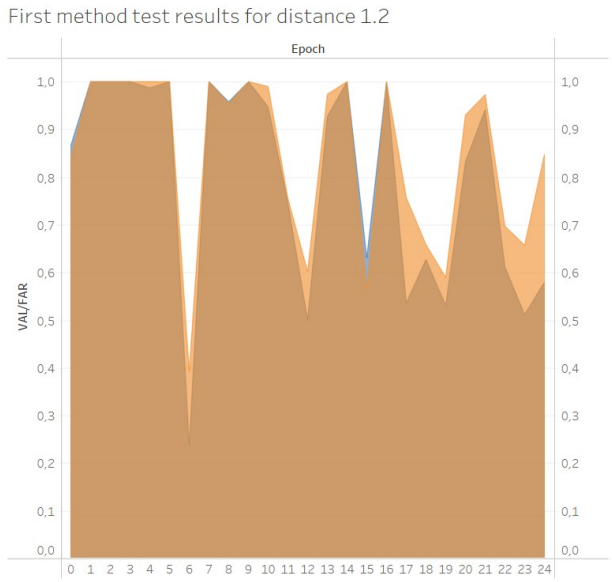
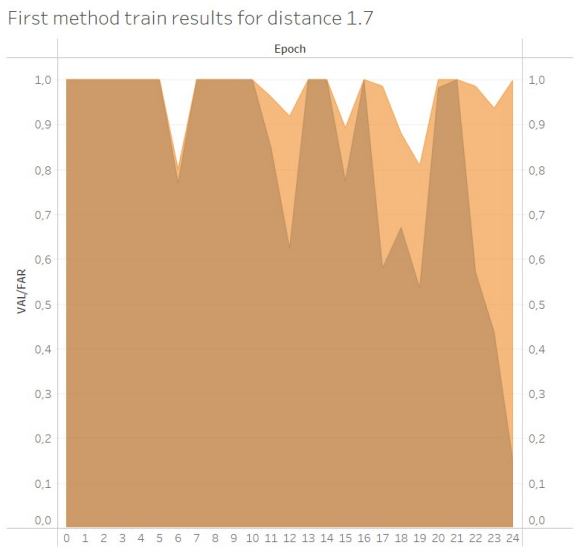
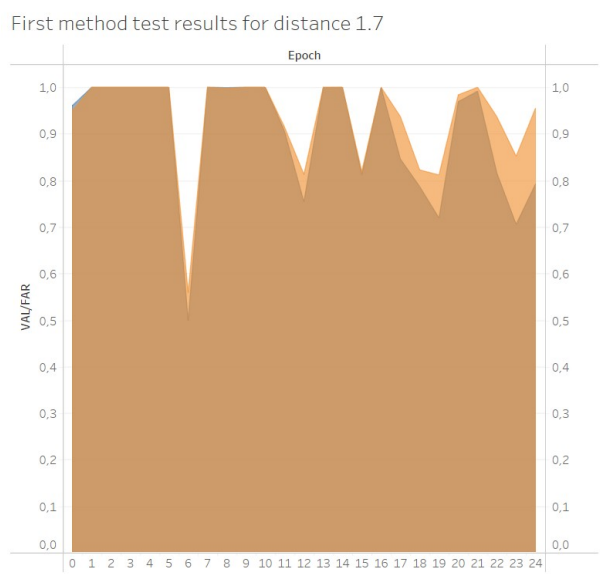
In the A Conditional Triplet Loss for Few-shot Learning and its Application to Image Co-Segmentation try we created the required loss function

```
class TripletLoss(nn.Module):
    def __init__(self, margin=1.0,a=0.5):
        super(TripletLoss, self).__init__()
        self.margin = margin
        self.alpha = a

    def calc_euclidean(self, x1, x2):
        return (x1 - x2).pow(2).sum(1)

    def forward(self, anchor: torch.Tensor, positive: torch.Tensor, negative: torch.Tensor) -> torch.Tensor:
        distance_positive = self.calc_euclidean(anchor, positive)
        distance_negative = self.calc_euclidean(anchor, negative)
        losses = torch.relu(distance_positive - distance_negative + self.margin)
        ###worst triplets
        if distance_negative>distance_negative+self.margin:
            losses = losses +self.alpha* (distance_positive+distance_negative)/2
        ##best cases
        k = 0.99
        d = distance_positive - distance_negative
        if d>self.margin*(k-1) and d< self.margin*(2*k-1):
            losses = losses -self.alpha* (distance_positive-distance_negative)/2

        return losses.mean()
```

RESULTS:

First Method :FaceNet: A Unified Embedding for Face Recognition and Clustering

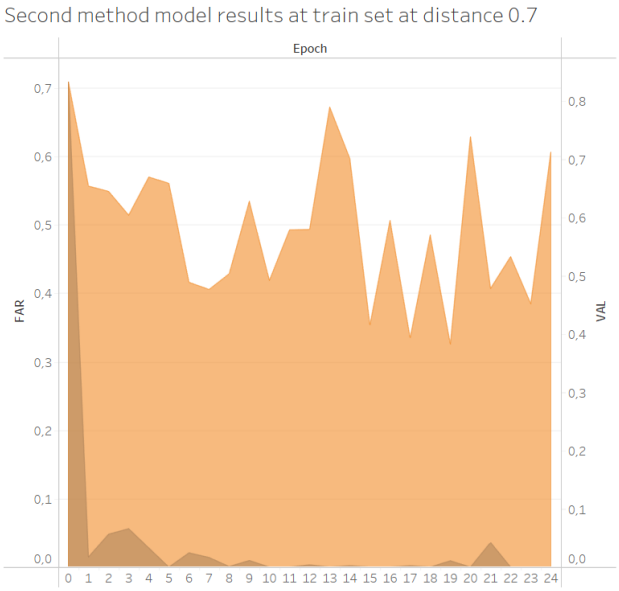
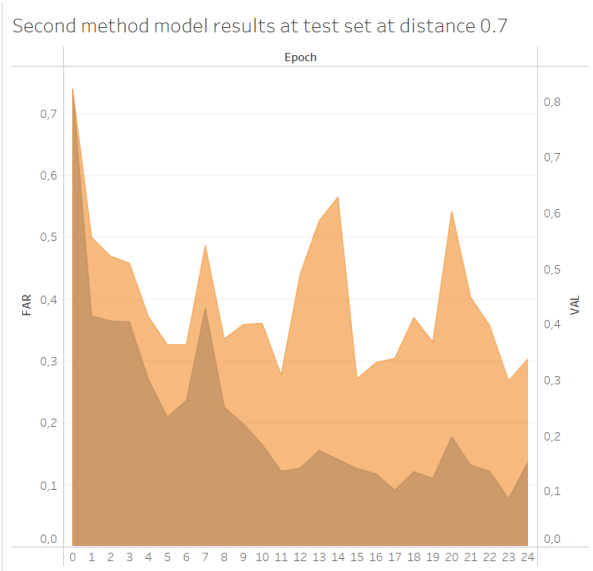
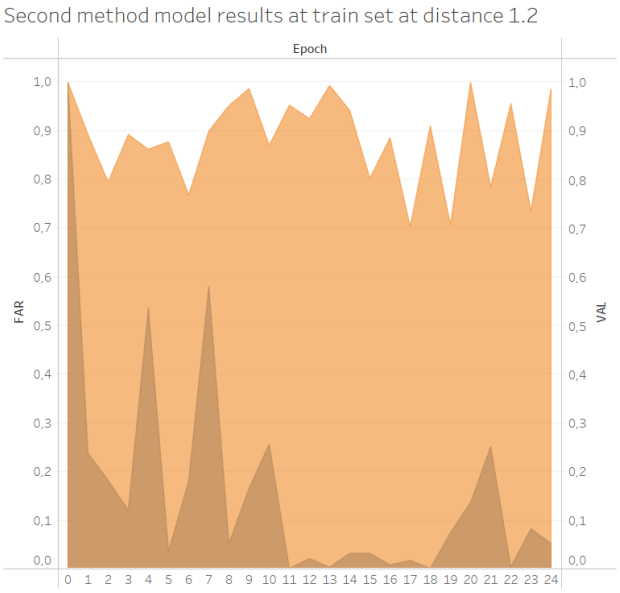
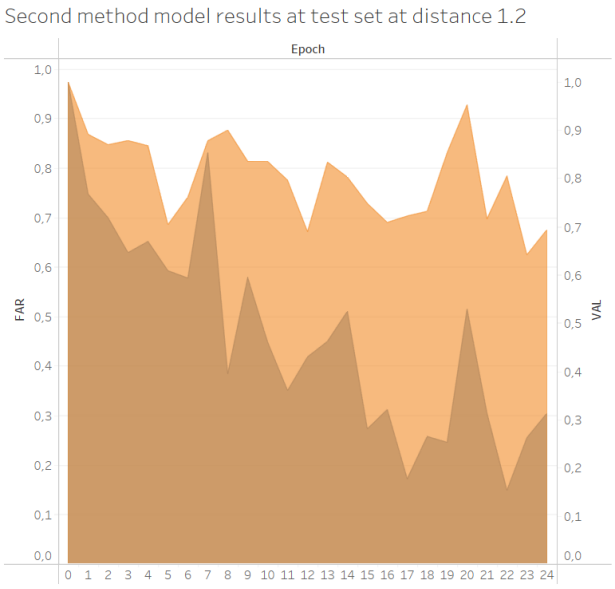
the distance below is the distance that seperates the two classes.

Train	Test
	
	

Initially because the FAR and VAL are around1. This means that the model has collapsed all the vectors to the same point. We see that the model as the time passes starts to separate them especially in the later epochs were the VAL are high and the FAR are low. The VAL are the light colored regions and the Far the dark colored regions. However we see that it cant generalize it to the test set due to the high FAR and thus a lot of false positives.

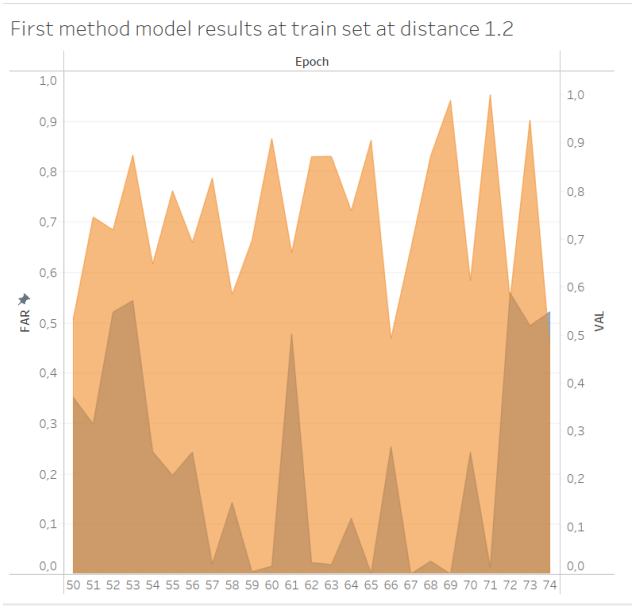
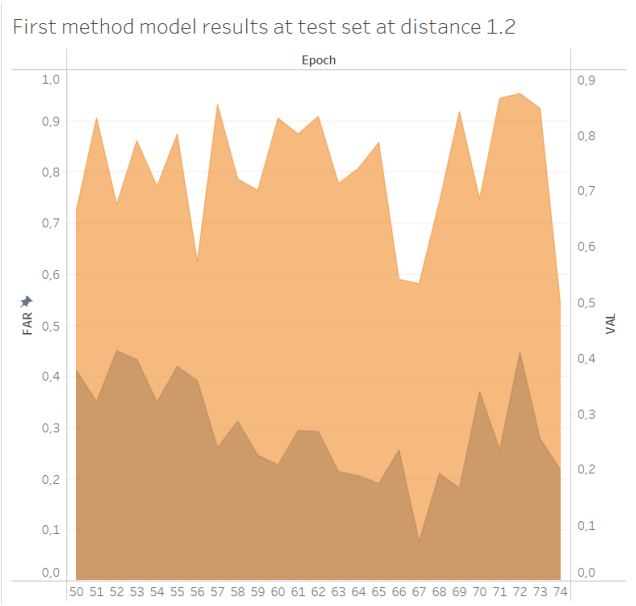
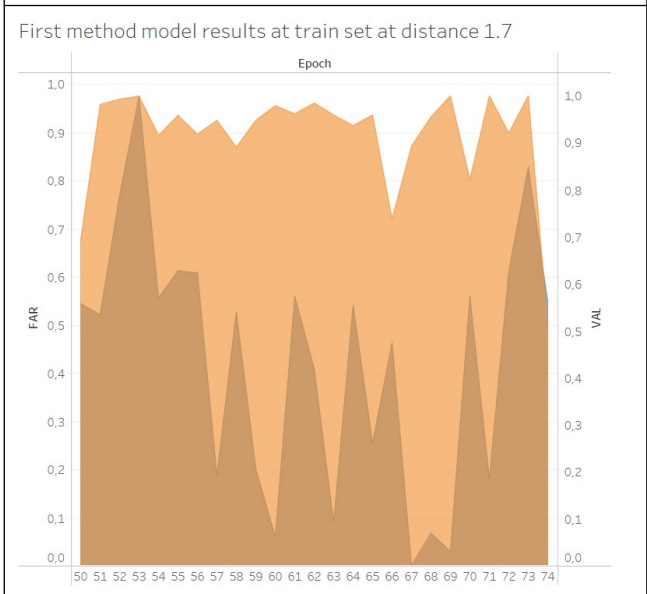
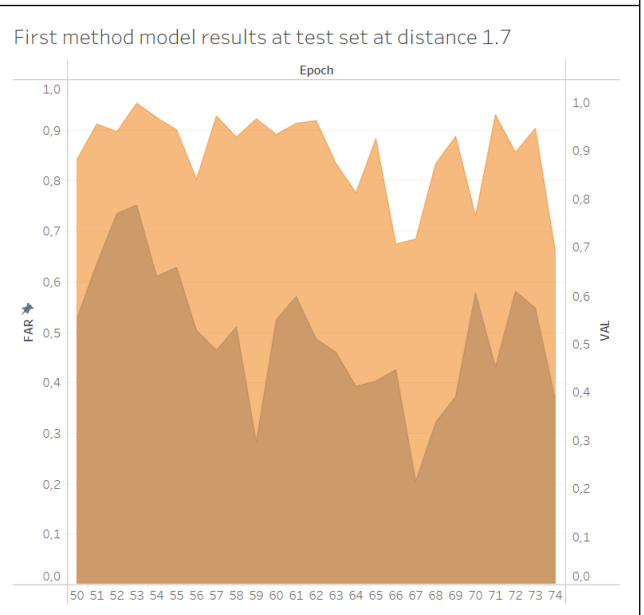
Second Method : A Conditional Triplet Loss for Few-shot Learning and its Application to Image Co-Segmentation

the distance below is the distance that seperates the two classes.

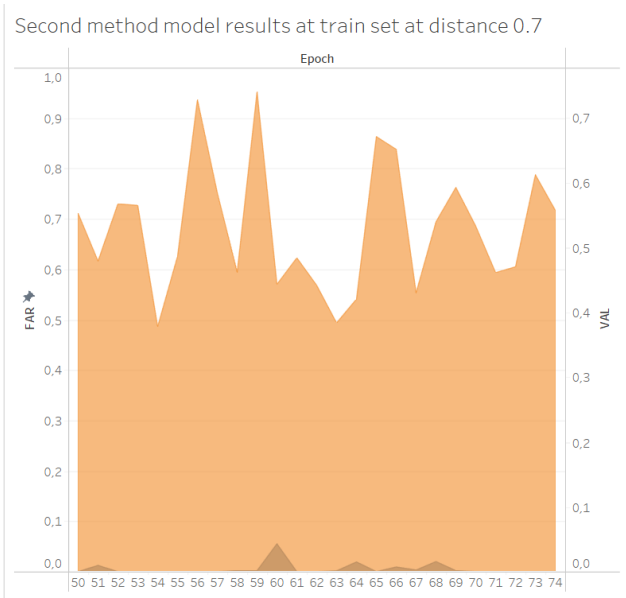
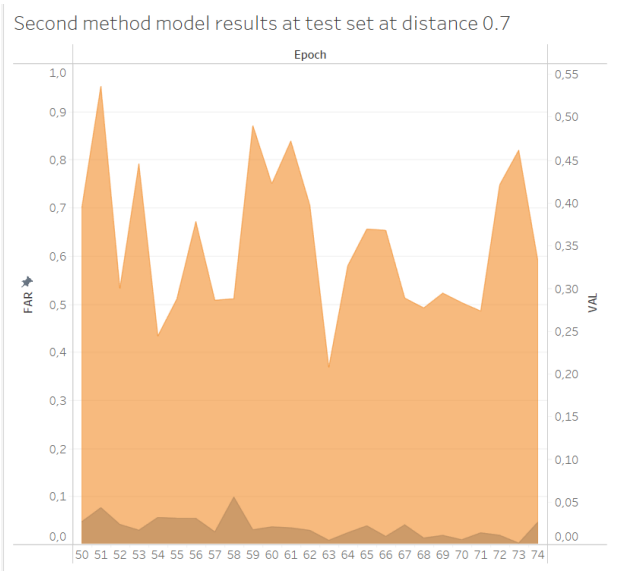
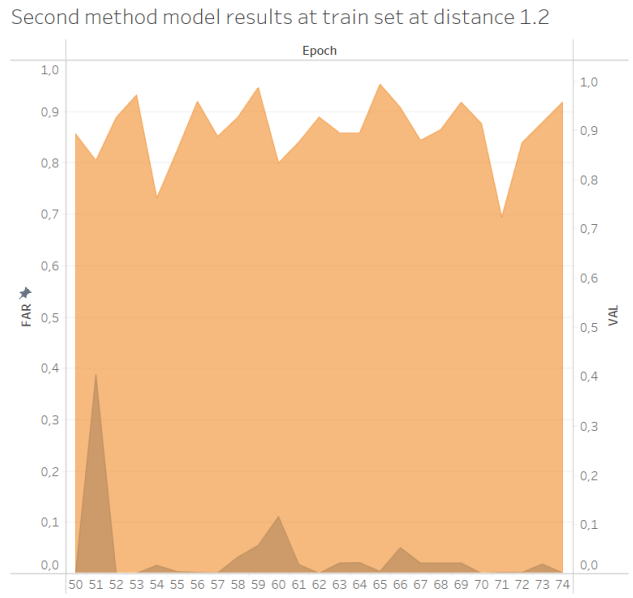
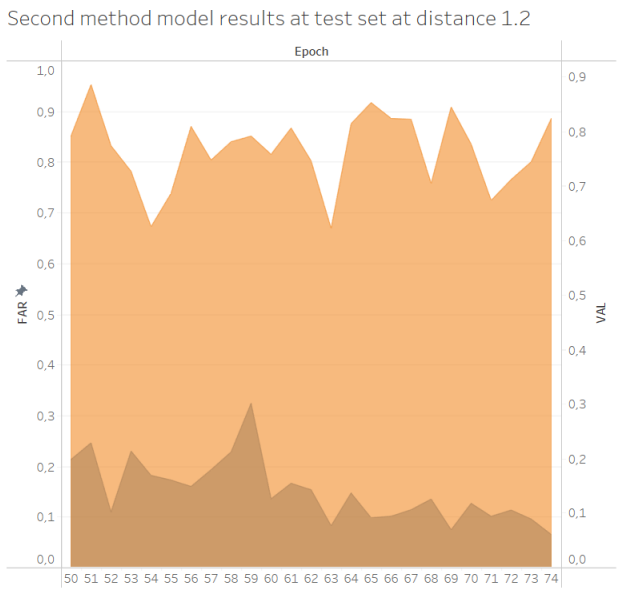
Train	Test
	
	

Here we see that the model false positives drop in both test and train. The VAL is the light colored orange and the FAR is the blue on the background. We also see that for the test set the false positives drop with time. This means that the model has probably a better generalization.
Even still in all cases the False positives are way higher compared to CNN's.

We see that in both models there is a trend for the false positives to fall as the epochs grow larger. So we decided to make the experiments one more for 75 batches in each epoch. The results for the batches 50 -74 are below.
First Method :FaceNet: A Unified Embedding for Face Recognition and Clustering

Train	Test
	
	

Second Method : A Conditional Triplet Loss for Few-shot Learning and its Application to Image Co-Segmentation

Train	Test
	
	

From the above we see that the false positives fall dramatically at the later batches both in test as well as the train set. From the two methods the second gives better results. This can mean that the first method has a slower convergence and maybe more epochs would be required to reach the same resutls. Another thing we see is that at lower distances the results are better this could be because the decision boundary between the two classes has been pushed lower and closer to one which in our training was the the margin we set when training the models.

Conclusions/Observations

We see from our work that the model with the best results were the CNN's. On average the results of the Transfer learning model is better compared to the Custom CNN as expected, but not by much. We had an idea to unfreeze another layer of the transfer learning model to achieve better performance. This however did not work out as well as we hoped. Going in the opposite way and freezing all but the last layer, essentially makes the model only able to determine from which camera the photos came. So unfreezing the two last layers seemed to us as the best solution.
Compared to the triplet loss models the transfer learning model was more reliable when applied for face verification. Due to the low input data the triplet models have trouble with multicolored backgrounds. When the background was akin to the skin color of the person to be verified the triplet loss model had problems verifying him. There are problems with lighting and distance from the camera as well but they can be more general. The distance must be not too far or too close and the lighting not too bright.
While the two methods are not directly comparable comparisons can be made about the time they need. The CNN's were in the range of a 1 to two minutes while the triplet loss took from half an hour to an hour to train. The triplet loss models seem to require quite a lot of epochs to be trained. Also the selection of parameters during their training can be challenging.

Between the two the faster training was the CNN's but in this our code can have affected the speed of the algorithm.