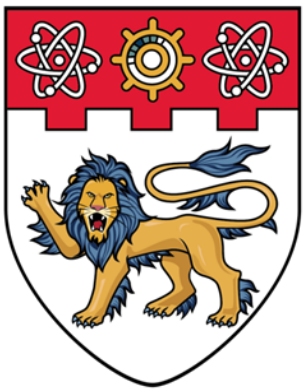

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4041 Machine Learning Project: Northeastern SMILE Lab - Recognizing Faces in the Wild

GROUP 44:

FAN YUPEI (U2120835A)

SANKAR SAMIKSHA (U2021021D)

TABASSUM NAZIR (U2023161K)

TAN JIA MIN (U2021019L)

XING KUN (U2023452E)

A project submitted to the School of Computer Science and Engineering as part of the CZ4041 Machine Learning Requirement

**All members contributed equally to the codes and the project.*

Table of Contents

1 IMPORTANT DEFINITIONS 3

2 EVALUATION SCORE AND RANKING POSITION OF PREDICTION RESULTS 3

3 PROBLEM STATEMENT 4

4 CHALLENGES OF THE PROBLEM 4

5 PROPOSED SOLUTION 5

5.1 OVERVIEW OF PROPOSED SOLUTION 5
5.2 DATA PREPROCESSING 5
5.2.1 Image Set Exploration 5
5.2.2 Randomizing image selection 6
5.2.3 Balancing of Relationship and Non-Relationship Data 6
5.2.4 Data Augmentation 7
5.2.5 Varying Augmentation per Epoch 9
5.3 THE MODEL 10
5.3.1 FaceNet 10
5.4 SIAMESE NETWORK 11
5.4.1 Classification Head 12
5.4.2 Running the Epochs 13
5.5 VALIDATION SET 13

6 EXPERIMENTS 14

6.1 RESNET101 VS FACENET 14
6.2 TUNING HYPERPARAMETERS 14
6.2.1 Hyperparameter 1: Learning rate λ 14
6.2.2 Hyperparameter 2: Dropout rate 14
6.2.3 Hyperparameter 4: Image Sizes 15
6.2.4 Hyperparameter 4: Batch size 15
6.3 USING DLIB 15
6.4 CHANGING THE SUBMISSION.CSV FORMAT AND DATASET 16
6.4.1 Selection of Relationship and Non-Relationship Pairs 16
6.4.2 Dataset Size 16
6.5 DATA AUGMENTATION 17
6.6 FINAL CHANGES 17

7 CONCLUSION 18

8 REFERENCES 19

1 Important Definitions

Term	Definition
Relationship Data	A pair of images, where the individuals are related
Non-Relationship Data	This means the two images provided are not kin. A pair of images where the individuals are not related

2 Evaluation Score and Ranking Position of Prediction Results

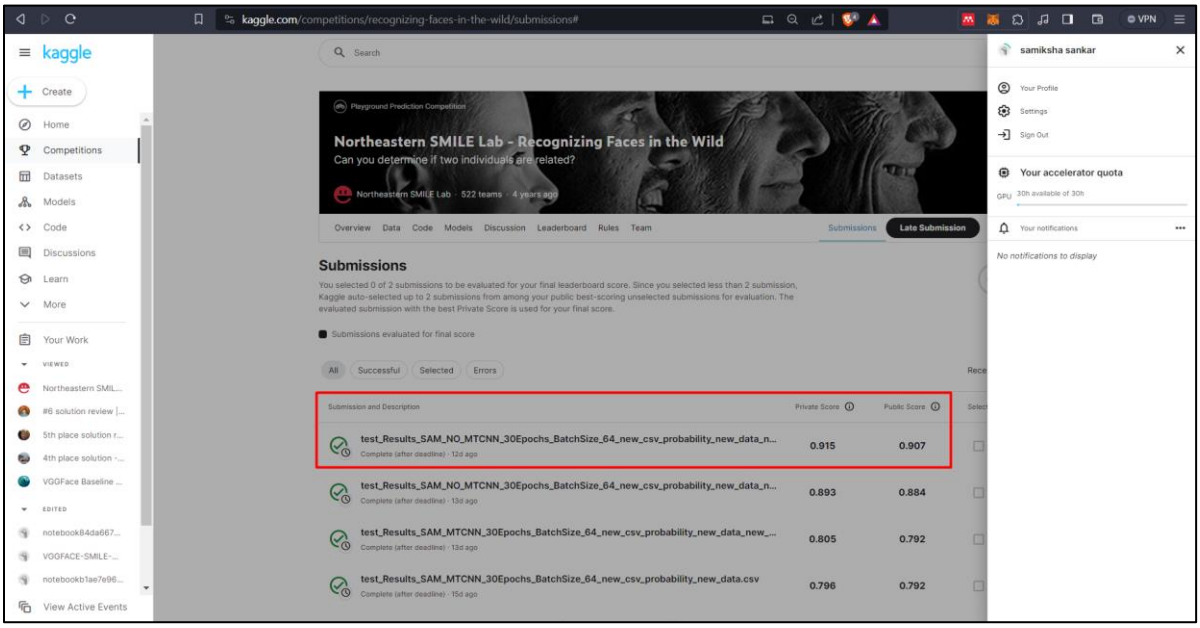

Figure 1: Kaggle Evaluation Score

The team managed to obtain a score of 0.907 and 0.915 in the public and private leaderboard respectively.

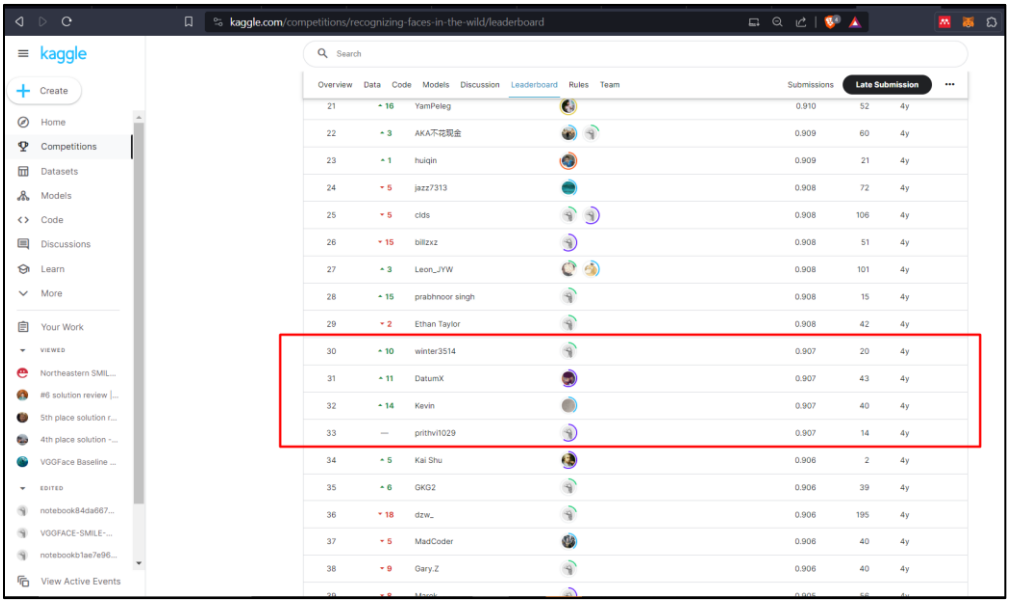

Figure 2: Kaggle Ranking

A score of 0.907 allowed the team to place within the 30-33 rank among the 523 participants in the public leaderboard. (Top 6%)

3 Problem Statement

In this project, our group chose to challenge the Kaggle competition: [Northeastern SMILE Lab - Recognizing Faces in the Wild], where we embarked on developing a deep learning model capable of determining if individuals are related by blood, simply from their facial images.

Our team's plan involved choosing an appropriate architecture, such as VGGface, ResNet or FaceNet as the backbone of our deep learning model. Subsequently, we conducted fine-tuning using the selected model pretrained on a suitable dataset. Additionally, we made strategic modifications to the training dataset and model parameters to enhance our model's ability to recognise kinship.

4 Challenges of the problem

1. Managing the complex relationship between kinship and facial similarity. The general idea behind this task is that people who are related likely share similar facial features while strangers are less likely to resemble each other. However, in reality, there are many flaws to this assumption. For example, siblings may not resemble each other if one inherited trait from their mother and the other from their father. Strangers can look alike by chance. Other factors such as age, gender, plastic surgery add even more complexities for the model to learn.
2. Deciding the optimal model complexity to prevent overfitting, while retaining the ability to capture subtle kinship features.

3. Balancing the training dataset to prevent bias towards more frequently represented demographics.
4. Selecting the appropriate model, loss function and optimiser etc., as well as effective tuning of hyperparameters
5. Conducting suitable data preprocessing to combat issues due to a poor-quality dataset, such as noise.
6. Designing models with shorter running time of each epoch due to the limited GPU source and time.

5 Proposed Solution

Platform: Google Colaboratory

Language: Python

5.1 Overview of Proposed Solution

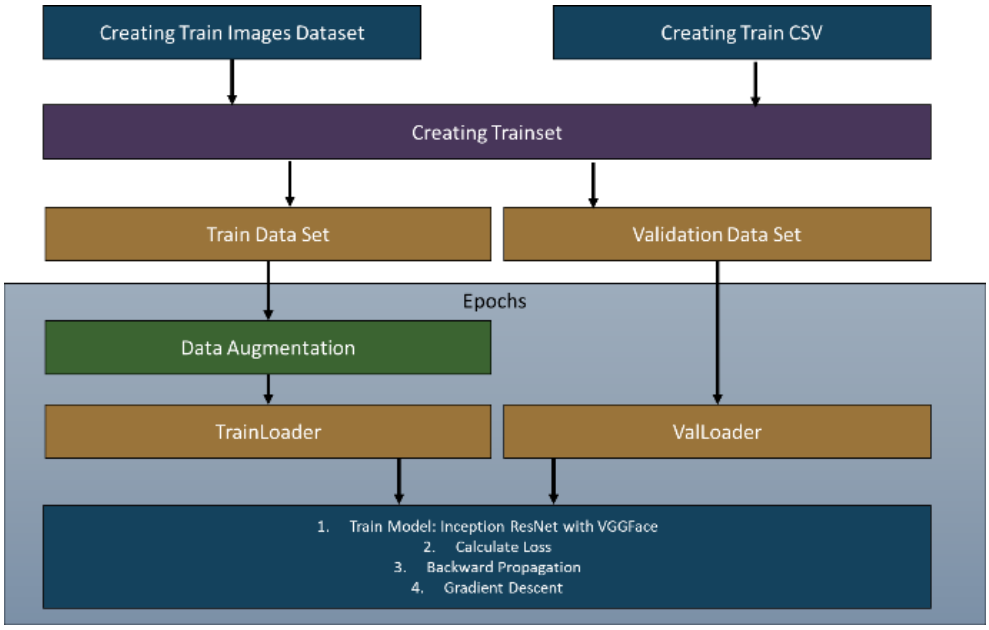

Figure 3: The flow of the model

5.2 Data preprocessing

5.2.1 Image Set Exploration

Various methods were explored to curate the dataset for fine-tuning. The 'train.zip' file containing roughly 15,000 images was the suggested training set to use, but showed poor performance, likely because it was not representative of the final testing set.

Subsequently, we experimented with 'train-faces.zip' and 'test-public-faces.zip,' each with approximately 25,000 images, but each contained a different collection of families and individuals. Ultimately, 'test-public-faces.zip' emerged as the most effective choice.

The relationships were obtained by combining the various CSVs in ‘test-public-lists.zip’.

5.2.2 Randomizing image selection

The dataset contained thousands of relationship pairs, each linking two individuals through kinship. As there are multiple candidate photos per person. should we have included every pair of images for every relationship, the dataset would have been too large. This is excluding the exponentially larger number of possible non-relationships.

We used a randomized selection of images (Figure 4), as opposed to a fixing the image used for each individual due to its better performance. This is further elaborated in Section 6 (Experiments).

```
while relatedCount<targetRelatedCount:
  for k,v in relationshipDict.items():
     for relation in v:
      i2 = random.randint(0, len(personPathFile[k])-1) #random photo of person1
      i3 = random.randint(0, len(personPathFile[relation])-1) #random photo of person2
      trainData.append((personPathFile[k][i2], personPathFile[relation][i3], 1))
      relatedCount+=1
      if relatedCount>=targetRelatedCount:
          break
     if relatedCount>=targetRelatedCount:
        break
```

Figure 4: Randomized Selection of Images for the Relationships

Randomizing the dataset allows the model to see larger variety of images and capture features important for generalization.

5.2.3 Balancing of Relationship and Non-Relationship Data

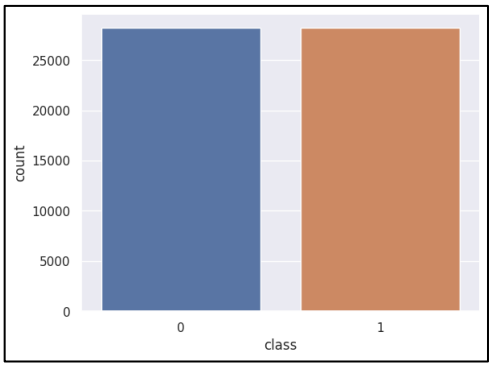

Figure 5: Balancing of Classes

There are two classes – relationship and non-relationship. Relationship is represented by the Boolean value of 1 and non-relationship is represented by the Boolean value of 0.

It is essential that the model receives a balanced dataset of both 0s and 1s. This is to prevent the model from being biased towards one class merely due to excess of one class.

5.2.4 Data Augmentation

Data Augmentation not only increases the variation in the images. Hence, the model can get more rich data allowing it to perform better and more accurately. While we know theoretically, it would help improve the model, the exact augmentation to be used must be experimented. This is explained further in Section 6.

In this section, the augmentations used is explained and supported with evidence on why it was used.

Random Crop

It was realized that certain faces in the test folder were cropped, had side profiles, or were obscured by accessories.

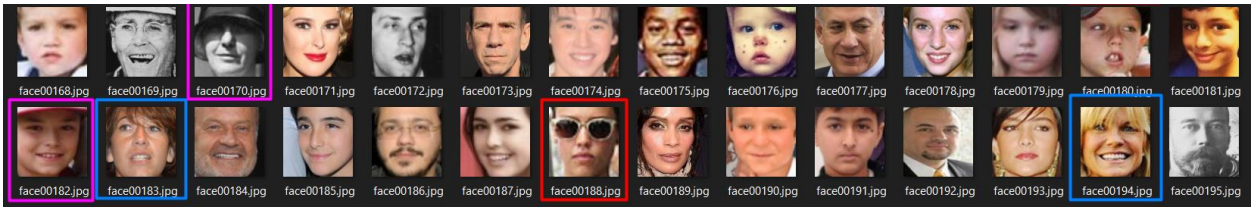

Figure 6: Obscured by Sunglasses(Red Boxes), Obscured by hat (Pink Boxes), Obscured by hair (Blue Boxes)

Figure 7: Side profiles

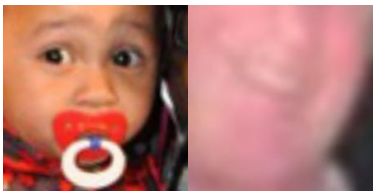

Figure 8: Obscured by bib and forehead is mostly cropped (face00774.jpg) [left]; Highly Blur image and cropped to only see the lower face (face00790.jpg) [right]

Based on Figure 8, it can be deduced that the test images contained not only randomly cropped images but highly blurred images as well.

Gaussian Blurs

Similarly, some images were not only cropped, side profiled or obscured by but also were blurred.

Hence, Gaussian Blur is applied in our project to which simulates real-world image blur by averaging nearby pixel values, emphasizing central pixels, which softens edges and reduces detail. This aligns training images with common test conditions like motion blur or focus issues, improving model generalization and predictive accuracy on blurred inputs. [1]

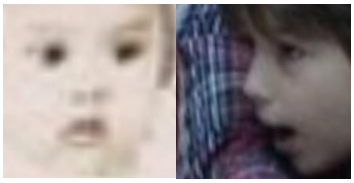

Figure 9: Blurred and Greyscaled image (face02454.jpg) [left]; Blurred and Side Profiled (face02678.jpg) [right]

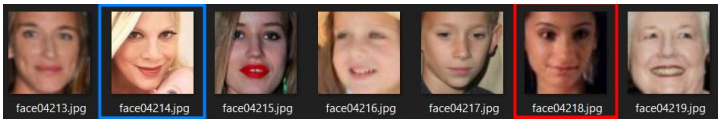

Figure 10: Blurred Images (Red Boxes) and Clearer Images (Blue Boxes)

Greyscale

Some images were greyscale as well. So, we applied grayscale conversion is to our dataset to reflect the varying photographic conditions and devices that often capture images in black and white. This technique removes the color information, forcing the model to focus on structural and textural details rather than color cues, which enhances its ability to discern facial features and relationships. [2]

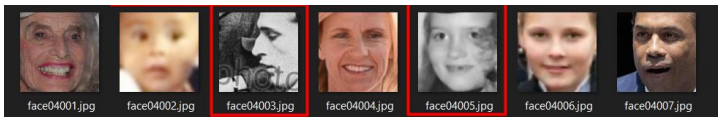

Figure 11: Examples of Greyscaled images

Color Jitter

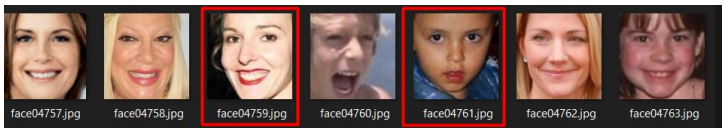

Figure 12: Brightness and Contrast Differences

In the above picture, some images are brighter (face04759.jpg) and some images are dimly lit (face04764.jpg & face04761.jpg). face04779.jpg seems to be naturally lit whereas face04786.jpg seems to be taken with lights.

For the code submitted to Kaggle, only the brightness and contrast was changed. The hue was not changed.

Horizontal Flip

Faces are not entirely symmetrical, hence employing random horizontal flips introduced additional variation during training.

5.2.5 Varying Augmentation per Epoch

Our team implemented different augmentations strategies for each epoch to introduce more diversity into the training process, preventing the model from over fitting.

```
#Change Data Augmentation every other epoch
if epoch % 10 == 0 or epoch %10 == 1 or epoch %10 == 2 or epoch %10 == 3:
  print("Data Augmentation: None")
  trainloader = createTrain([transforms.Resize((IMG_SIZE,IMG_SIZE)),transforms.ToTensor()])
elif epoch %10 == 4 or epoch %10 == 5:
   print("Data Augmentation: RandomGrayScale(0.5)")
   trainloader = createTrain([transforms.Resize((IMG_SIZE,IMG_SIZE)),transforms.RandomGrayscale(p=0.5),transforms.ToTensor()])
elif epoch %10 == 6 or epoch %10 == 7:
  print("Data Augmentation: RandomCrop((90,90)),RandomGrayScale(0.8), RandomHorizontalFlip, GaussianBlur(kernel_size = 5,
      sigma=(0.1, 3.0)")
  trainloader =
      createTrain(
      [transforms.RandomCrop((80,80)),transforms.Resize((IMG_SIZE,IMG_SIZE)),transforms.RandomGrayscale(p=0.8),transforms
      .RandomHorizontalFlip(),transforms.GaussianBlur(kernel_size = 5, sigma=(0.1, 3.0)),transforms.ToTensor()])
elif epoch %10 == 8 or epoch %10 == 9:
  print("Data Augmentation: RandomGrayScale(0.8), RandomHorizontalFlip, ColorJitter(brightness=0.7, contrast=0.3),")
  trainloader =
      createTrain(
      [transforms.Resize((IMG_SIZE,IMG_SIZE)),transforms.RandomGrayscale(p=0.5),transforms.RandomHorizontalFlip(),transforms
      .ColorJitter(brightness=0.7, contrast=0.3),transforms.ToTensor()])
else:
  print("Data Augmentation: None")
  trainloader = createTrain([transforms.Resize((IMG_SIZE,IMG_SIZE)),transforms.ToTensor()])
```

Figure 13: Varying Augmentations every epoch

The validation set's accuracy stagnated after multiple runs, and the corresponding Kaggle score was not great. This could be due to overfitting. To improve the model's prediction, random data augmentations were introduced between epochs and simulated training with different datasets, enhancing its ability to predict in diverse test environments.

The code is provided in Figure 13. The first 4 epochs only resizes the images. The next 2 epochs adds random greyscaling. The next 2 epochs does greyscaling, horizontal flipping and gaussian blurring. The following 2 epochs does greyscaling, horizontal flipping, increasing brightness and increasing contrast.

Originally, all the epochs had used only one dataset and with the same data augmentation, i.e., no varying augmentation every epoch. The latter provided better results and generalizability than the former version.

5.3 The Model

The final proposed solution uses FaceNet. The team also tried Resnet101. However, FaceNet produced better results. Resnet101 will be explained further in Section 6.

The figure below shows a diagrammatic overview of the original Inception Resnet V1 Model and the Model that the team built specifically for this project.

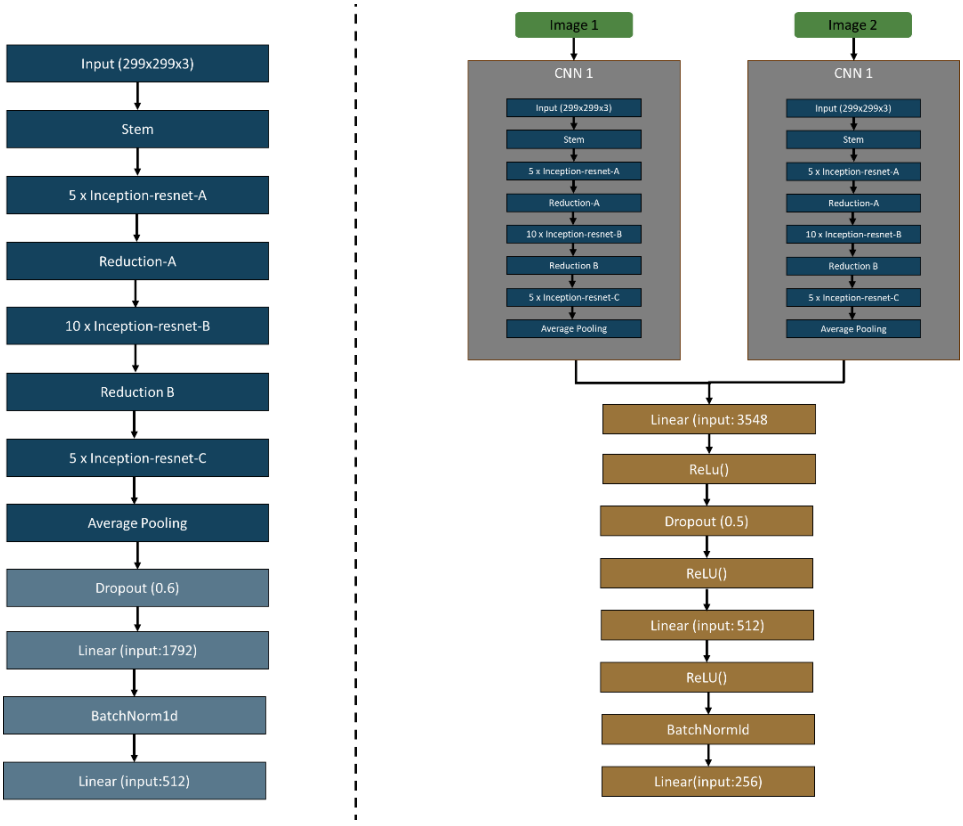

Figure 14: Inception Resnet V1 [Left]; This Project's Model [Right]

5.3.1 FaceNet

Timesler's FaceNet is "a repository for Inception Resnet (V1) models in pytorch, pretrained on VGGFace2 and CASIA-Webface". FaceNet imported the parameters from David Sandberg's Face Recognition model to initialize the PyTorch model weights.

FaceNet was originally developed by Google in 2015 for facial recognition. It proposed ZF-Net and Inception Network followed by a method called triplet loss as the loss function. Later this version underwent multiple improvements. [3] [4] [5] [6, 7, 8]

The "stem" of the Inception network was modified to have three main inception modules named A, B, and C. [6, 7, 8] The three inception modules were implemented in Inception Resnet. The pooling layers from Inception-v4 was replaced by residual connections and an additional 1x1 convolution was added.

This allowed Inception-Resnet-v1 to obtain higher accuracies at lower epochs. This potentially explains why our model could end at epoch 30 and did not have the need to extend to 100 epochs

generally used in deep learning training. The architecture for the Inception-Resnet-V1 Model is provided in Figure 14.

David Sandberg's model used Tensorflow, which utilized the Inception-Resnet-V1. This model was specifically designed for facial recognition. [6, 7, 8] Sandberg offers a pre-trained model of Inception-Resnet-V1 trained using the VGGFace2 dataset. VGGFace2 is an extensive dataset containing only faces and this allows the model to be specifically accurate for detecting faces.

Additionally, Sandberg not just implemented the triplet loss that was utilized in the original FaceNet by Google but implemented the "Softmax activation with cross entropy loss" as well. This theory was used as training with the triplet loss can become difficult [9]. This allowed his model to be more efficient.

This TensorFlow version was converted into a PyTorch Version and is provided by the following repository: https://github.com/timesler/facenet-pytorch [10].

5.4 Siamese Network

Siamese Network is a method of using identical networks for two images to extract similarity features [11]. This model was used in this project.

Siamese Network is a method of using identical networks for two images to extract similarity features [11]. This model was used in this project.

```
def forward(self, input1, input2):
    output1 = self.cnn1(input1)
    output1 = output1.view(output1.size()[0], -1)
    output2 = self.cnn1(input2)
    output2 = output2.view(output2.size()[0], -1)


    output = torch.cat((output1, output2),1)
    output = self.fc(output)
    return output
```

Figure 15: Siamese Network

Two identical copies of FaceNet (PyTorch version) were used. The dropout layer, linear layers, and Batch Norm 1d layers were removed in these identical copies. This is shown in Figure 14.

In the pseudocode provided in Figure 15, *image1* and *image2* are passed through the Siamese networks. Subsequently, the two outputs are concatenated to form a 1-dimensional tensor. This is passed through the custom-made fully connected layers to generate the *Output.*

5.4.1 Classification Head

```
model.classifier = nn.Sequential(
    nn.Linear(3584,2048),
    nn.ReLU(),
    nn.Dropout(0.55),  # Add dropout for regularization
    nn.Linear(2048, 512),
    nn.ReLU(),
    nn.Linear(512, 256),
    nn.ReLU(),
    nn.BatchNorm1d(256), # Apply batch normalization
    nn.Linear(256, 2)
)
```

Figure 16: Classification Head

nn.Linear(in, out) represents a fully connected (linear) layer that executes the operation – output equals the input multiplied by weights plus bias. Four successive layers use this operation to progressively decrease feature dimensions, with the final layer being nn.Linear(256,2) for our bianry classification task (related vs not related).

nn.ReLU() stands for Rectified Linear Unit, an activation function crucial for introducing non-linearity into the neural network. This activation function encourages sparsity and potentially reduces overfitting. Compared to other activations like sigmoid or tanh, ReLU is more efficient and helps mitigate the vanishing gradients problem.

nn.Dropout(0.55) serves as a regularization technique during training, randomly nullifying 55% of input units. This reduction in network capacity prevents overfitting and promotes better generalization.

nn. BatchNorm1D(256) utilizes batch normalization techniques tailored for a fully connected (1D) layer containing 256 features. It normalizes input across the batch dimension, aiming to expedite training and diminish internal covariate shift within the network.

We used a pretrained InceptionResnetV1 model, pretrained on the VGGFace2 dataset. VGGFace2 is a large-scale face recognition dataset with an estimated 3.31 million images divided into 9131 classes, which is a much larger dataset than the recognizing-faces-in-the-wild dataset. Such extensive datasets allow us to capture generic features or representations that are transferable to various downstream tasks. By using a pretrained model, we can leverage the useful knowledge gained from the large amount of data to initialize model parameters, which enables faster convergence during our finetuning process, reducing the need for extensive training our smaller task-specific dataset, while possibly improving performance compared to if only the smaller dataset was used.

In our project, we used parameter freezing, such that the weights for the InceptionResnetV1 models will not be updated while training. They will produce the feature maps of each image, capturing the import information of each face, before we feed it into our final fully connected layers, which parameters will be updated during finetuning.

5.4.2 Running the Epochs

```
for i, data in enumerate(trainloader,0):
    img0, img1 , labels = data
    img0, img1 , labels = img0.cuda(), img1.cuda() , labels.cuda()
    optimizer.zero_grad()
    outputs = net(img0,img1)
    loss = criterion(outputs,labels)
    loss.backward()
    optimizer.step()
    total_loss += loss.item()
```

Figure 17: Code for running the epochs

Loss Calculation

The line of code 'loss = criterion(outputs, labels)' represents the computation of the cross-entropy loss function. It computes the loss based on 'outputs' (predicted values) and the ground truth 'labels'. The loss value calculates how much the predicted probability diverges from the actual label. Cross-entropy loss measures the performance of the model with an output value of probability between 0 and 1. In this project, PyTorch is used to implement the loss function through the 'nn.CrossEntropyLoss' class.

Backward Propagation

Backpropagation is the iterative adjustment of weights to minimize the loss function. During forward pass, the model uses input data to produce 'outputs'. The loss is calculated using the cross-entropy loss function. The next step is backward pass ('loss.backward()'). The gradients of error with respect to the output in the output layer are computed and then backpropagated to calculate gradients for weights in each layer. The optimizer is updated using 'optimizer.step()' based on the optimization algorithm. Backpropagation is repeated for several epochs until the loss function is minimized and the point of convergence is reached. The gradient descent optimization algorithm relies on backpropagation.

Gradient Descent

Gradient descent is an optimization algorithm commonly used to train neural networks. It minimizes the loss function by adjusting its parameters until it yields the smallest possible error. Learning rate is a hyperparameter defined as the size in the steps taken by the optimizer to reach convergence. For this project, we have used the Adaptive Moment Estimation (Adam) optimization which is an extension of stochastic gradient descent (SGD). It is based on the adaptive estimation of first and second order moments. While SGD maintains a single learning rate which does not change during training, the Adam optimizer computes individual adaptive learning rates from estimates of the first and second moments of the gradients.

5.5 Validation Set

We used 'train_test_split' function imported from scikit-learn to create the validation set. The function will randomly shuffle and split input data into two subsets: trainSet and valSet. We can set the validation set size using 'test_size'. Since it is set to a value of 0.1, 10% of the training dataset is used for the validation set. The 'random_state' parameter is used to set the random

seed for reproducibility which allows us to achieve the same split every run, thus ensuring consistency across all code runs.

```
# Create the Siamese network
net = SiameseNetwork(InceptionResnetV1(pretrained='vggface2', classify=False)).cuda()
# Define the cross entropy loss
criterion = nn.CrossEntropyLoss()

# Define the optimizer (e.g., Adam)
optimizer = optim.Adam(net.parameters(), lr=0.0005)
```

Figure 18: Chosen Model, Loss and Optimizer

6 Experiments

6.1 Resnet101 vs FaceNet

We first started off by determining which architecture would be most suitable for our task and dataset. Using train.zip and the fixed dataset approach, we decided to test the performance of ResNet101 (pretrained on ImageNet) and Facenet (pretrained on VGGFace2) as both were popular choices for facial recognition. For each, we used the base model with minimal tweaks as both subnetworks of the Siamese Net. Facenet was shown to have a slightly better performance, with a Kaggle Score of 0.61 which we used as a baseline.

Model	Batch Size	Learning Rate	Kaggle Score
Facenet	8	0.0005	0.61
Resnet101	8	0.0005	0.602

6.2 Tuning Hyperparameters

Next, we started to tune different hyperparameters to increase our Kaggle score.

6.2.1 Hyperparameter 1: Learning rate λ

Learning rate is the rate of a neural network updates its weights in response to the loss gradient for every epoch. It is the most critical hyperparameter in deep learning. A high learning rate will lead to faster convergence but might overshoot the optimal weights. A smaller learning rate might lead to better performance, but it also significantly slows training and risks getting stuck in local minima if excessively low. Thus, we experimented to find a balance between them.

We tested different learning rates from 0.1 to 0.00001 and reached the optimum 0.0005.

6.2.2 Hyperparameter 2: Dropout rate

The dropout rate is the probability of a neuron being dropped during training. Increasing dropout rate reduces the participation of neurons and can overfitting.

We decided to add a dropout layer, as we found that while our training accuracy was high, our test accuracy was much lower, a likely sign of overfitting. We experimented with dropout rates between 0.5 and 0.7 and found the most effective value of to be 0.55.

6.2.3 Hyperparameter 4: Image Sizes

We tried to increase the image size (from 100x100 to 240x240) in hopes that larger images can capture finer details and nuances within the images, which may lead to higher accuracies. However, our Kaggle score dropped from 0.635 to 0.485.

It was not a problem of overfitting as even the training accuracy was lower when the image size used was 240 x 240. We suspect that the larger images significantly increased the complexity of the learning task, making it more challenging for the model to learn essential patterns and features.

Image Size	Model	Hyperparameters	Epochs	Kaggle Score
100	Facenet	Batch Size = 128 Dropout = 0.5	10	**0.635**
224	Facenet	Batch_Size = 128 Dropout = 0.5	10	**0.485**

6.2.4 Hyperparameter 4: Batch size

Batch size determines the degree of variability in the updates to the model's weights. Larger batch sizes can increase stability in the updates and training speed. However, smaller batch sizes can lead to better generalization as it uses a smaller subset of the data. With the following experiments (batch size 64 vs 128), we could not observe a significant difference in performance at low epochs. However, since our finetuning did not take extremely long (2h for 30 epochs), we decided to use the smaller batch sizes which would theoretically lead to better performance.

Model	Batch Size	Learning rate	Dropout rate	Epochs	Kaggle Score
Facenet	**128**	0.0005	0.8	5	0.662
Facenet	**128**	0.0005	0.8	20	0.672
Facenet	**64**	0.0005	0.8	5	0.655
Facenet	**64**	0.0005	0.8	20	0.678

6.3 Using Dlib

Dlib is a C++ library with face detection, facial landmark detection, and image alignment capabilities. We theorised that by aligning all the faces in the images before feeding it into our model, it might be better able to learn important facial features, leading to higher accuracy.

However, we found that Dlib requires a very high computational power especially with our large dataset. This soon led to lack of GPU resources on Google Colab, and the job would often be killed even when iterating through the first few epochs. Using Dlib also increased the time taken for each epoch significantly from around 4 minutes to 50 minutes per epoch. Additionally, the Kaggle score from 0.507 to 0.534, which we deemed insignificant for the challenges we faced. Thus, with the limited time and resources given, we decided to drop Dlib.

6.4 Changing the submission.csv format and dataset

Up until this stage, we were struggling with consistently low Kaggle scores below 0.7. Upon investigation, we identified an issue with our submission.csv format. Initially, we had been outputting predicted labels (0,1) based on the sample_submission.csv format. However, we realized the Kaggle score was evaluated using the area under the ROC curve between predicted probabilities and observed targets, contrary to the format we were following.

At the same time, we recognized the inadequacy of our training dataset to be representative of the test set, as indicated by higher validation accuracies (>0.85) as compared to the corresponding Kaggle Scores. We made two changes to the dataset.

6.4.1 Selection of Relationship and Non-Relationship Pairs

- **“Fixed" dataset:** Initially, for each individual, we selected their first image to represent them in a relation or non-relation. Should we need more datapoints after exhausting this, we continued to use their second image and so on. The rationale was to eliminate other factors by using a consistent image, isolating only the kinship relation for the model to learn. This method showed signs of overfitting as mentioned.
- **Randomized dataset:** For each relationship or non-relationship, for each of the two individuals, we select a random image of them to represent them. This led to much better results, with good performance on unseen data.

6.4.2 Dataset Size

We pivoted to a larger and hence more veried dataset (test-public-faces.zip) and experimented with a range of 10k to 100k datapoints. We observed that roughly 30k instances provided a satisfactory performance within a reasonable training time.

Contrary to the common notion that larger datasets lead to improved results, we noticed that excessively large datasets did not improve performance. This is in line with another competitor who obtained better results by randomly selecting 30,000 image pairs, compared to using all 300k possible image pairs. This might be due to the imbalance in the quantity of images per individual. Using all available images of each person to generate pairs appears to bias the model towards the specific characteristics of those individuals, resulting in poorer generalization in the test set.

Implementing these changes in submission format and dataset resulted in a significant performance leap (0.618 to 0.795). Notably, the substantial improvement was primarily attributed to the dataset change, as using the new submission format with the old dataset did not show nearly as drastic of change.

6.5 Data Augmentation

To further improve accuracy, we experimented with more data augmentation to account for the noisy samples in the unseen data. We noticed that using Random Crop and Color Jitter saw significant improvements.

Data	Augmentation	Model	Hyperparameters	Epochs	Kaggle Score
Test-public-faces.zip (30K data points)	Random Greyscale, Random Horizontal Flip, Gaussian Blur	Facenet	Batch_Size = 64 Dropout = 0.7	30	0.792
Test-public-faces.zip (30K data points)	Random Greyscale, Random Horizontal Flip, Gaussian Blur, **Random Crop(75,75) and Color Jitter.**	Facenet	Batch Size 64 Dropout = 0.7	30	0.884

6.6 Final Changes

To achieve our results, we decreased the dropout rate from (0.7 to 0.5). The rationale was that we no longer observed overfitting and saw advantages of retaining the contributions of more neurons and hence the information learned. This showed as improvement in the Kaggle Score of 0.884 to 0.898.

Lastly, we decided to increase the portion of the image retained by random crop (from 75x75 to 80x80), to match the samples more closely in the test set. Finally, by increasing dropout slightly (from 0.5 to 0.55), we managed to attain our final result of 0.907.

Data	Augmentation	Model	Hyperparameters	Epochs	Kaggle Score
Test-public-faces.zip (30K data points)	With random crop(75,75) and color jitter	Facenet	Batch Size 64 **Dropout = 0.7**	30	**0.884**
Test-public-faces.zip (30K data points)	With random crop(75,75) and color jitter	Facenet	Batch Size 64 **Dropout = 0.5**	30	**0.898**
Test-public-faces.zip (30K data points)	With random crop(80,80) and color jitter	Facenet	Batch Size 64 **Dropout = 0.55**	30	**0.907**

7 Conclusion

In conclusion, for this project, we explored using deep learning for facial recognition and recognizing kinship. Our solution involved transfer learning, using the Facenet model pretrained on the VGGFace2 dataset as subnetworks for our SiameseNet, followed by finetuning our custom classification head on the competition dataset. We also tried other novel techniques, such as using Dlib.

From our experiments, we have made some unexpected discoveries. For example, while a much larger dataset typically results in better performance, we observed the opposite for this task. Variation in the dataset was more important than its size. Hence, we introduced more image diversity with purposeful techniques like using a different dataset, implementing data augmentation, and randomizing the relationship and non-relationship pairs.

From this project, we learnt just how task-dependent such deep learning projects are. There are many facial recognition papers and projects available, but we could not simply use the same approaches or hyperparameters. Rather we had to delve deep to tailor a solution that works best for our specific dataset through trial and error.

In this project, we also are enlightened by the capabilities of such technology, being able to recognize kinship from two simple images with upwards of 90% accuracy. This also brings up ethical concerns of such tech, relating to the public's privacy.

8 References

[1] V. Vignesh, "Gaussian Blurring — A Gentle Introduction," *Towards AI,* 2023.

[2] V. R. M. a. P. J. B. H. A. K. Bobak, "A grey area: how does image hue affect unfamiliar face matching?," *Cognitive Research: Principles and Implications,* 2019.

[3] V. Rajput, "Face Detection and Recognition capable of beating humans using FaceNet," Analytics Vidhya, 26 June 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/face-detection-and-recognition-capable-of-beating-humans-using-facenet/. [Accessed 18 November 2023].

[4] GeeksForGeeks, "FaceNet – Using Facial Recognition System," GeeksForGeeks, 12 June 2022. [Online]. Available: https://www.geeksforgeeks.org/facenet-using-facial-recognition-system/. [Accessed 18 November 2023].

[5] S. Ghimire, "What is 'FaceNet' and how does facial recognition system work?," Medium, 22 May 2021. [Online]. Available: https://medium.com/analytics-vidhya/what-is-facenet-and-how-does-facial-recognition-system-work-d7c1eb6e2800. [Accessed 18 November 2023].

[6] B. Raj, "A Simple Guide to the Versions of the Inception Network," Medium, 30 May 2018. [Online]. Available: https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202. [Accessed 18 November 2023].

[7] N. K. Kankam, "Understanding Inception-ResNet V1 architecture," [Online]. Available: https://iq.opengenus.org/inception-resnet-v1/. [Accessed 18 November 2023].

[8] M. Deore, "FaceNet Architecture," Medium, 4 April 2019. [Online]. Available: https://medium.com/analytics-vidhya/facenet-architecture-part-1-a062d5d918a1. [Accessed 18 November 2023].

[9] O. M. Parkhi, A. Vedaldi and A. Zisserman, "Deep Face Recognition," University of Oxford, 2015. [Online]. Available: https://www.robots.ox.ac.uk/~vgg/publications/2015/Parkhi15/parkhi15.pdf. [Accessed 18 November 2023].

[10] timesler, "facenet-pytorch," 2023. [Online]. Available: https://github.com/timesler/facenet-pytorch#conversion-of-parameters-from-tensorflow-to-pytorch. [Accessed 18 November 2023].

[11] A. Dutt, "Siamese Networks Introduction and Implementation," Medium, 12 March 2021. [Online]. Available: https://towardsdatascience.com/siamese-networks-introduction-and-implementation-2140e3443dee. [Accessed 18 November 2023].

[12] D. Sandberg, "facenet," 17 April 2018. [Online]. Available: https://github.com/davidsandberg/facenet. [Accessed 18 November 2023].