

# CZ4041 GROUP PROJECT: NORTHEASTERN SMILE LAB - RECOGNIZING FACES IN THE WILD

**Leon Guertler**

U2023443J

leon002@e.ntu.edu.sg

**Cao QingTian**

U2020646L

caoq0013@e.ntu.edu.sg

**Chen Peilin**

U2040284H

chen1411@e.ntu.edu.sg

**George Rahul**

U2023835C

rahul020@e.ntu.edu.sg

**Chen Xingyu**

U2021140F

chen1399@e.ntu.edu.sg

## 1 INTRODUCTION

The rapid advancements in the field of facial recognition and machine learning have opened doors to myriad applications, ranging from security and surveillance to social media and entertainment. One intriguing and relatively unexplored domain within facial recognition is kinship verification – the process of determining familial relationships based on facial features. The implications of accurate kinship verification can be profound, impacting areas such as genetics research, ancestry tracing, and even solving missing persons cases.

Kaggle, a prominent platform for machine learning competitions, introduced the SMILE Kinship Prediction Challenge to stimulate research in this niche area. The challenge aimed to push the boundaries of current kinship verification techniques by providing a comprehensive dataset of parent-child facial images and tasking participants to create algorithms capable of accurately predicting these relationships. The significance of such a challenge lies not just in its technical intricacy, but also in its potential to shed light on the genetic and environmental factors influencing facial resemblance among family members.

This report delves into our approach to the SMILE Kinship Prediction Challenge, exploring the methodologies we employed, the challenges we encountered, and the insights we gleaned from the process. Through our analysis, we aim to contribute to the growing body of knowledge surrounding kinship verification and highlight the potential implications of our findings for both the academic and practical realms.

The remainder of this report is structured as follows: Section 2 will in more detail outline the specifics of the SMILE Kinship Prediction Challenge, Section 3 will explore the provided data, Section 4 will introduce the various architectures we have benchmarked, Section 5 will show the performances of these architectures, and finally, we will conclude in Section 7.

## 2 PROBLEM DEFINITION

The SMILE Kinship Prediction Challenge, at its core, presents a binary classification task. Participants are provided with pairs of facial images and are tasked with determining whether the individuals in the images share a kinship relationship or not.

### 2.1 INPUT DATA

The input is a pair of two distinct images. These images are drawn from a diverse range of ethnicities, ages, and genders to ensure a comprehensive representation.

### 2.2 OUTPUT PREDICTION

For each image pair, participants are expected to predict a binary outcome:

- **1** indicating that the individuals in the image pair share a kinship relationship.
- **0** indicating that there is no kinship between the individuals.

The objective of the challenge is to maximize the AUC (Area under the Receiver Operator Curve; See Equation 1) score of these predictions, ensuring that true familial relationships are correctly identified while minimizing the false positives. Presumably, the organizers chose the AUC as a metric of success because of the slight imbalance in the data (see Section 3) Majority of the families have 3 individuals. For each input individual, the number of positive samples is limited as compared to its number of negative samples, which can be individuals from any other families.

$$\text{AUC} = \int_0^1 \text{TPR}(t) dt \quad (1)$$

Where:

- $\text{TPR}(t)$  is the True Positive Rate at a given threshold  $t$ .
- The integral bounds (0 to 1) represent the full range of possible False Positive Rates.

To achieve a high AUC score in this task, it's essential to extract and focus on subtle facial features and patterns that might be indicative of genetic resemblance, without being distracted by image noise (such as the angle, lightning conditions etc.). Furthermore, the challenge lies in differentiating between genuine genetic similarities and similarities that might arise due to environmental factors or mere coincidences.

### 3 EXPLORATORY DATA ANALYSIS

#### 3.1 DATA OVERVIEW

The training dataset provides a total of 12,389 images in train.zip, representing 470 families and 2,316 individuals. By exploring the train relationships file, there are 3,598 relationships involving all 470 families in the dataset. Through systematic sampling, where one family is selected out of every 20 families, it was observed that images of an individual may capture various perspectives, including different angles, various age stages, and the presence or absence of accessories such as glasses. As an illustration, consider F0908/MID1:



**Figure 1:** F0908/MID1

#### 3.2 DATA DISTRIBUTION

The distribution of individuals per family is approximately normal, spanning from 1 to 41. The majority of families exhibit a concentration of 2 to 10 individuals, as depicted in Figure 2. Notably, the most prevalent scenario involves around 120 families with precisely 3 individuals.

Similarly, the distribution of images per individual showcases a left-skewed normal pattern, ranging from 1 to 95. A significant portion of individuals possesses 1 to 13 images, as visualized in Figure 3. This distribution underscores the imbalance in the number of images per person, particularly notable due to a substantial proportion of individuals having only 1 training image.

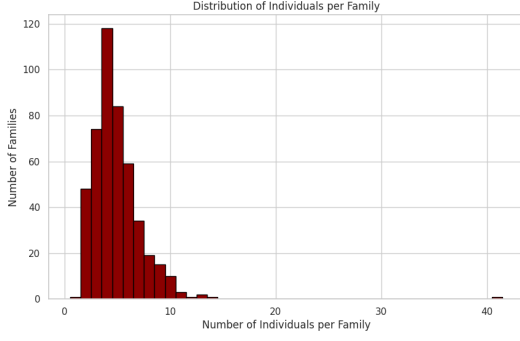


Figure 2: Individuals per Family

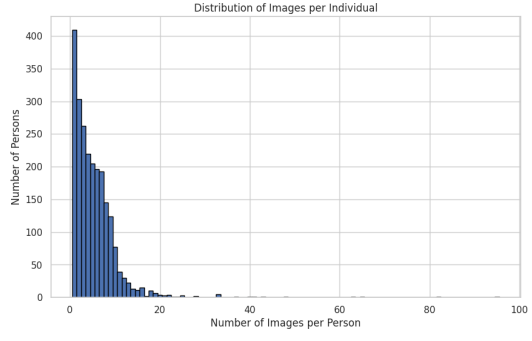


Figure 3: Images per Individual

### 3.3 CHOICE OF VALIDATION DATA

By extracting the top 30 families that exhibit the highest frequency in train relationships and the top 30 families with the greatest number of images, we pinpointed the commonalities as the sampling pool for the validation dataset. From the families that appeared in both sets, we randomly selected one family, which is F009, to serve as the validation family during training.

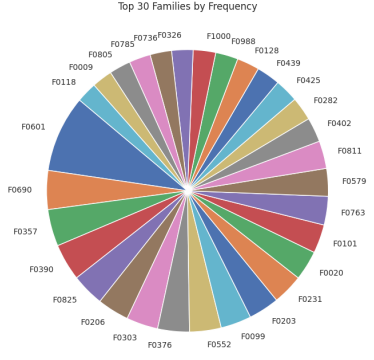


Figure 4: Top 30 Families by Frequency in Train Relationships

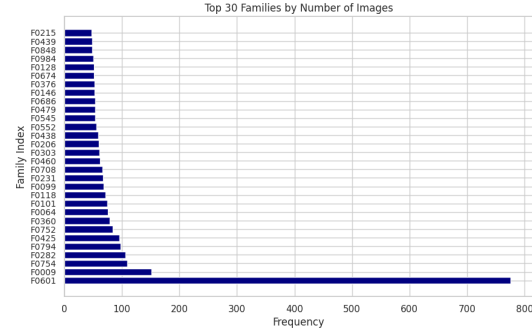


Figure 5: Top 30 Families by Number of Images

## 4 ARCHITECTURES AND APPROACHES

In this section we will outline, in detail, all of the architectures we used for our experiments. Though the individual architectures differ when it comes to loss function used, techniques for combining features, pretraining methods, neural architectures and hyperparameters, do they also share a number of similarities, namely: all architectures follow a siamese's-esque set-up, where two encoders (usually with shared weights) are used to encode both provided images, these encodings are merged (which can be done via simple concatenation, standard operations, or more complex methods), and this merged features are passed through a classification head.

The remainder of this section is organized as follows: in Section 4.1, we introduce the siamese network, and general architecture for our use case (Subsection 4.1.1), and Loss functions used (Subsection 4.1.2). Subsequently, we introduce the different ensemble strategies used in Section 4.2

### 4.1 SIAMESE NETWORKS

A siamese neural network is an artificial neural network that uses the same weights on two different input vectors to compute comparable output vectors. In our case, it could be simplified to using the same encoder on two different images, followed by an optional auxiliary classifier depending on the type of losses we use.

#### 4.1.1 ARCHITECTURES

Our architecture is shown below (Figure 6). For losses such as contrastive loss, we only need the embeddings from the encoder. For binary cross entropy loss, we will post-process the embeddings via

some form of mathematical operations and concatenation, then we convert the “difference” feature into a probability on whether they are related through a few linear layers.

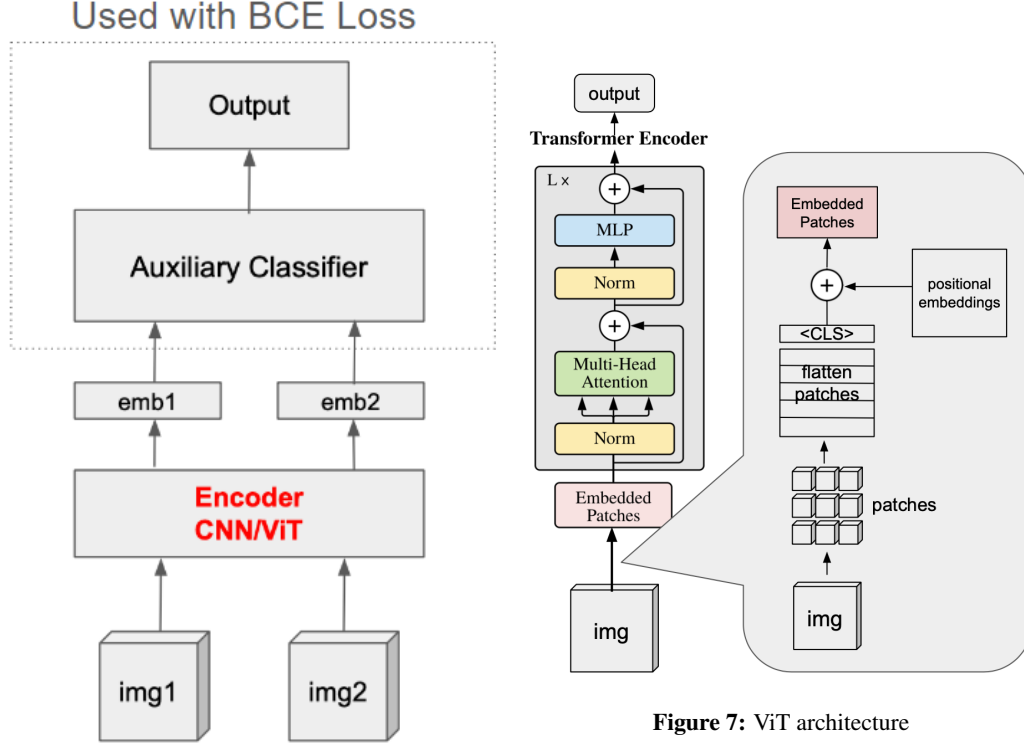


Figure 7: ViT architecture

Figure 6: General Configuration

**Convolutional Neural Network (CNN)** In the field of Computer Vision, Convolutional Neural Networks (CNNs) are essential structures that have had a significant impact on image processing tasks Liu et al. (2022). These specialized neural networks excel at capturing intricate spatial hierarchies and local patterns within images, making them a cornerstone in various computer vision applications.

The fundamental operation of CNNs involves the application of convolutional layers to input images. Unlike traditional fully connected layers, CNNs utilize filters or kernels that slide across the input image, extracting localized features. This process allows the network to progressively learn and combine hierarchical representations, starting from simple textures and edges to more complex structures.

The input image, denoted as  $\mathbf{x} \in \mathbb{R}^{H \times W \times B}$ , undergoes a series of convolutions, producing feature maps that highlight relevant patterns. The network’s ability to preserve spatial relationships between pixels is crucial for tasks such as object recognition, segmentation, and image classification.

Unlike transformer-based approaches, CNN encoders do not treat image elements as tokens; instead, they operate directly on pixel values, leveraging spatial information for robust feature extraction. This characteristic makes CNNs particularly effective when preserving the inherent spatial structures of images is essential.

**Vision Transformer (ViT)** In Computer Vision, Vision Transformers (ViTs) Dosovitskiy et al. (2021) have achieved state-of-the-art performance in a variety of downstream tasks such as image classification. It comprises a transformer encoder module that accepts the features from various patches as input tokens and converts them into output tokens. It’s worth noting that, for the purposes of our experiments, the focus lies solely on the ViT encoder and thus we will only elaborate on how an image flows through the encoder.

As the architecture shown in Figure 7, the image  $\mathbf{x} \in \mathbb{R}^{H \times W \times B}$  is first partitioned into patches  $\mathbf{p} \in \mathbb{R}^{N \times (P^2 \cdot B)}$ , where  $H$ ,  $W$  and  $B$  are the height, width, and spectral bands of the image,  $(P, P)$  is the

resolution of the patch, and  $N = HW/P^2$  is the number of patches obtained. In our implementation, we set  $P = 16$  and the flattened patches are processed and fed in parallel to the Transformer encoder, in the same way as tokens (words) for NLP applications. Following NLP conventional NLP training strategy, an extra classification token ( $<CLS>$ -token) is added to the beginning of the sequence, whose last hidden states can be used as a classification feature vector.

A multilayer perceptron (MLP) block and a multi-head self-attention (MHSA) block comprise the Transformer encoder. A Layer Norm (LN) is used prior to the MHSA and MLP blocks, with residual connections following each block. The Gaussian Error Linear Unit (GELU) is the activation function utilised for the MLP. The encoder block is represented as:

$$\begin{aligned} z'_l &= MSA(LN(z_{l-1})) + z_{l-1} \\ z_l &= MLP(LN(z'_l)) + z'_l, \end{aligned} \quad (2)$$

where  $z$  is the feature tokens and  $l = 1, 2, \dots, L$ . The performance of the Transformer can be attributed to its self-attention mechanism in the MHSA block which captures the correlation between sequential tokens. This is achieved by linearly mapping the tokens to a Query  $\mathbf{Q}$ , Key  $\mathbf{K}$ , and Value  $\mathbf{V}$  vectors. The attention score measures the strength of the relationship of a token with other tokens in the sequence.

The self-attention is computed as

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_K}}\right)\mathbf{V}, \quad (3)$$

where  $d_K$  is the dimension of  $\mathbf{K}$ . The MHSA module uses the self-attention as multiple heads in parallel and concatenates the outputs followed by a linear projection as

$$MHSA(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = Concat(A_1, A_2, \dots, A_H)\mathbf{W}, \quad (4)$$

where  $H$  is the number of attention heads,  $\mathbf{W} \in \mathbb{R}^{H \times d_k \times N}$  is a learnable parameter matrix and  $N$  is the number of patch tokens. The output obtained by the MHSA module is then fed into the MLP module after the residual connection as follows,

$$z = L(GELU(L(x))) \quad (5)$$

where  $x$  is the layer normalized output of the MSA and  $L(\cdot)$  is a linear transformation operation. At this point we have the image embedding which can be used for our downstream tasks.

#### 4.1.2 LOSS FUNCTIONS

To decide on which loss is the best, we conducted several experiments in the next section. Here we will outline the motivations. For Contrastive loss, Circle loss, (N+1)-Tuplet Loss and L1 Hinge Loss, we can use the embeddings from the encoders. The binary cross entropy loss will require attaching a classifier at the end, which follows roughly the same architecture as mentioned in Figure 6.

**Contrastive loss** The idea of using contrastive loss is bringing images of the same family to a point, while we push images of different families apart. In equation 6,  $y$  is the label of the pair, whereas  $D$  refers to the distance or dissimilarity between the embeddings.

$$L = (1 - y)D^2 + \max(0, m - D)^2 y \quad (6)$$

**Circle loss** Another loss we often use is the circle loss to train the embeddings. In the below formula,  $\gamma$  is a scale factor and  $m$  is a margin for better similarity separation, while  $s_n$  and  $s_p$  are similarities scores of positive and negative samples respectively.

$$L = \log\left[1 + \sum_{i=1}^K \sum_{j=1}^L \exp(\gamma(s_n^j - s_p^i + m))\right] \quad (7)$$

**(N+1)-Tuplet Loss for Multiple Negative Examples** The (N+1)-Tuplet Loss is an extension of the traditional triplet loss. In the standard triplet loss, each training sample is associated with an anchor image, a positive image (from the same class), and a negative image (from a different class). The loss encourages the network to minimize the distance between the anchor and positive images while maximizing the distance between the anchor and negative images. However, it doesn't consider multiple negative samples for a given anchor, which can lead to imbalanced training.

The (N+1)-Tuplet Loss helps address limitation by introducing multiple negative examples (N) for each anchor image. This helps improve the robustness of the learned embeddings and can be especially useful in scenarios where there is a large amount of intra-class variability. In our case, due to the computational limit, we set the number of negative examples to be 15 for each tuplet. The AUC of the model did improve by around 7% as compared to triplet loss. Further increasing the size of negative examples may improve the performance, however, due to GPU resource limit, we are not able to experiment with it.

$$(N+1)\text{-Tuplet Loss} = \log \left( 1 + \sum_i \exp(D(a, n_i) - D(a, p)) \right) \quad (8)$$

where  $D(a, n_i)$  is the angular distance between anchor and each negative example,  $D(a, p)$  is the angular distance between anchor and positive.

**Binary Cross Entropy loss** If we attach an auxiliary classifier behind the encoders, we can utilise binary entropy loss. In this case, if we have N images, we calculate the inaccuracies of their classification by taking the actual label of  $i^{th}$  images times the log of predicted possibility that whether this image belongs to a certain class, and sum all the inaccuracies for the two classes and for all images.

$$\text{BCE Loss} = -\frac{1}{N} \sum_{i=1}^N Y_i \log \hat{Y}_i + (1 - Y_i) \log(1 - \hat{Y}_i) \quad (9)$$

**Weighted Binary Cross Entropy** Weighted Binary Cross-Entropy is a variant of the standard Binary Cross-Entropy loss function. Binary Cross-Entropy measures the dissimilarity between the predicted class probabilities and the actual binary labels.

Weighted Binary Cross-Entropy introduces the concept of class weights to address class imbalance issues. In our case, we utilised BCEWithLogitsLoss from the PyTorch torch.nn module with parameter pos\_weight to be 2.00. This assigns more weight to the positive class, making the model pay more attention to correctly classifying positive samples. This helps increase the Accuracy and AUC of model as compared to standard Binary Cross-Entropy.

**L1 Hinge Loss** L1 Hinge Loss is a loss function designed for margin-based classification tasks, making it suitable for binary classification problems. In our case, we customised this Loss class by combining the Hinge Loss and L1 regularization.

The Hinge Loss encourages a margin between the predicted scores ( $f_x$ ) and the true labels (y). The L1 regularization term encourages sparsity in the weight vector ( $lambda_{reg}$ ) by taking the L1 norm (sum of absolute values) of the weight vector. The combined loss is computed as the mean Hinge Loss plus the product of the L1 regularization term and the L1 norm.

$$\text{L1 Hinge Loss} = \frac{1}{N} \sum_{i=1}^N \max(0, 1 - y_i \cdot f(x_i)) + \lambda \sum_{j=1}^M |w_j| \quad (10)$$

## 4.2 ENSEMBLE METHODS

Since using ensembles has proven to be a good strategy in many Kaggle competitions, we too, will be utilizing them here. In this section, we will first introduce the standard approach for doing so, and subsequently a plethora of additional approaches that we want to try. All of these approaches have in common that they are post-hoc, in that, they utilize the generated confidences for each instance of the test set by each model, rather than combining the actual models via boosting or additional training. For the remainder of this section, we will follow the following notations:

- $y_{i,j} \in [0, 1]$  denotes the prediction for the  $i$ -th test sample, by the  $j$ -th model
- where  $i \in I$  and  $j$  abbreviates  $\Omega_j$  where  $j \in J$

#### 4.2.1 SIMPLE AVERAGING & WEIGHTED AVERAGING

The by far most simplistic and popular techniques for building ensembles, not just in this competition, but in general, is simply taking the average

$$\tilde{y}_i = \frac{1}{|J|} \sum_j y_{i,j} \quad (11)$$

where  $|J|$  denotes the size of the set  $J$  (i.e. the number of models).

or taking the weighted average

$$\tilde{y}_i = \frac{1}{|J| + \sum_j w_j} \sum_j y_{i,j} * w_j \quad (12)$$

where  $w_j$  denotes the weight for the  $j$ -th model. In our experiments, we will use the following options as weights:

**private-score** Since we ultimately measure performance on the public-score, and want to keep good data hygiene, we will use the private-score as the weights.

**private-score-pow(x)** This follows a similar set-up to the ‘private-score’ one from above, with the only exception that we scale the score by taking it to the  $x$ -th power in order to magnify it’s impact:

$$\tilde{y}_i = \frac{1}{|J| + \sum_j w_j^x} \sum_j y_{i,j} * w_j^x \quad (13)$$

**private-score-ranked** Rather than using the actual scores achieved as weights, here, we first rank the performances and then use the integers as weights (where we rank them from worst to best, such that the best has the highest weight).

**correlation** Lastly, we hypothesize that models that perform well, but have a lower correlation, will create better ensembles. With this in mind, we utilize the inversed average correlation score of each model as weights:

$$\tilde{y}_i = \frac{1}{|J| + \sum_j \frac{1}{\text{corr}_j}} \sum_j y_{i,j} * \frac{1}{\text{corr}_j} \quad (14)$$

where  $\text{corr}_j$  is determined using Equation 15

$$\text{corr}_j = \frac{1}{|J|} \sum_k \left( \frac{\sum (y_{i,j} - \bar{y}_j)(y_{i,k} - \bar{y}_k)}{\sqrt{\sum (y_{i,j} - \bar{y}_j)^2 \sum (y_{i,k} - \bar{y}_k)^2}} \right) \quad (15)$$

#### 4.2.2 CLUSTERING-BASED ENSEMBLE

In addition to the traditional ensemble techniques described above, we introduce a unique approach that employs clustering to ensemble models. This strategy is rooted in the hypothesis that there exists an inherent structure among the predictions of different models, and by uncovering this structure, we can improve the combined predictions.

Let’s first introduce the clustering methodology:

Given the predictions of each model for the test instances, we represent them as a matrix. Each row of this matrix corresponds to a test instance, while each column represents a model’s predictions.

We then employ the K-Means clustering algorithm to cluster the models based on their predictions. The number of clusters, denoted by  $n\_clusters$ , is determined as the minimum of either 3 or the number of models ( $|J|$ ).

The clustering assigns each model to a cluster, and for each cluster, we calculate the average prediction for each test instance. These averaged predictions are then combined across all clusters, ensuring that predictions from each cluster contribute equally to the final ensemble prediction.

Mathematically, the combined prediction for the  $i$ -th test sample is represented as:

$$\tilde{y}_i^{cluster} = \frac{1}{n\_clusters} \sum_c^{n\_clusters} \frac{1}{|J_c|} \sum_j^{J_c} y_{i,j} \quad (16)$$

where  $c$  denotes the cluster index,  $J_c$  represents the models in cluster  $c$ , and  $|J_c|$  is the number of models in cluster  $c$ .

This approach ensures a balanced contribution from each cluster to the final ensemble, which, in turn, is expected to provide a robust and diversified prediction. Given the novelty of this approach, further experiments and evaluations would be vital to gauge its effectiveness in various scenarios.

#### 4.2.3 BINNING

For this ensemble, we simply allocated each result into a bin of size .05 and selected the bin with the most ‘votes’.

## 5 EXPERIMENTS

In this section, we will delve into the details of the experiments we conducted using different models as image encoders, in the Siamese Network setup discussed in the previous section. We utilized both Convolutional Neural Network based encoders, pretrained on a mixture of the VGG-Face, Casia-WebFace and ImageNet datasets and a Vision Transformer pretrained on the MS-Celeb-1M dataset. To ensure a comprehensive evaluation, our experimentation included our own versions of these encoders fine tuned on the Faces in the Wild (FIW) dataset provided in the competition. All experiments were implemented in PyTorch, with the utilization of CUDA for accelerated model training.

### 5.1 DATA PREPARATION

Images from the ‘train’ folder were utilized as training data alongside the ‘train\_relationships.csv’ file, which contains pairs of related individuals. Notably, the CSV file only includes related pairs (positive samples). Any pairs not included in the CSV file are considered as non-related (negative samples), which will be created during training to create a comprehensive training dataset with both positive and negative instances. To maintain class balance, each training batch consists of a combination of positive and negative samples. Half of the batch is taken from pairs in the CSV file, while the remaining half is created by randomly selecting individuals from the training folder, forming pairs not present in the CSV file. Subsequently, the images are cropped to match the input size of different encoders. The dataset is further enriched through data augmentation techniques such as RandomHorizontalFlip, ColorJitter, and RandomRotation.

### 5.2 CONVOLUTIONAL NEURAL NETWORKS

We first use CNN-based encoders to experiment with losses. The encoders we use are mainly resnet (He et al. (2015)) and VGG (Simonyan & Zisserman (2015)).

#### 5.2.1 CONTRASTIVE LOSS

Although the loss decreases with training, the AUC score does not necessarily decrease. It is possibly because the training dataset is too small for complicated encoders, i.e. the encoders emphasize too much on similarities, causing the final result to be at around 0.5. Moreover, this loss aims to bring the portraits of the same family together as a point, but in reality, the images of the same family should resemble a cluster instead.

We then switch the metric to euclidean distance, with a single neuron at the end as a classifier. The trend of AUC score is similar to what we attempted with cosine distance, further proving our points.

When we add a mixture of contrastive loss to a network with complicated processing of embeddings, the mixture does not help to improve the final result. The original VGG network using a mixture of entropy loss with Euclidean-evaluated contrastive loss will drop the performance from 0.79 to 0.78. We hypothesise that this is because the Euclidean valuation constrains the way model recognises the difference, thus causing the decrease in model accuracy.



### 5.2.2 CIRCLE LOSS

Circle loss indeed increases accuracy from contrastive loss by a bit (the AUC score can reach to about 0.6). However, it still is no match for entropy loss with a complex classifier attached. The reason behind might be similar to that of contrastive loss: it focuses too much of the similarities without a proper classifier function to separate them apart.

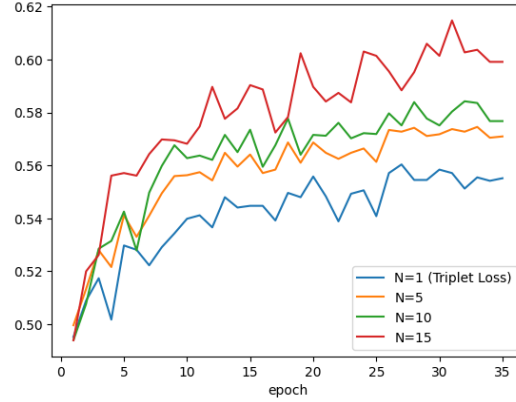
We discover that a mixture of losses also does not improve the final result either. The best AUC score from a mixture of loss is 0.602 compared to 0.675. We hypothesise that this is because circle loss will tend to yield a vastly larger number than entropy loss, and to balance the two losses we will multiply a very small number in front of circle loss, therefore worsening its effect on the overall network training performance.

### 5.2.3 TRIPLET LOSS VS (N+1)-TUPLET LOSS

We experimented with the triplet loss function to train the model for similarity learning, but the results were unsatisfactory, as indicated by a 54% accuracy rate. Upon investigation, we discovered that the triplet loss is constrained by the inadequacy of negative samples generated for each anchor. Since some false classes did not appear in the triplets as negative examples, the model might accidentally get close to them, resulting in inaccurate predictions.

To tackle this issue, we opted to implement the (N+1)-Tuplet Loss, incorporating N negative samples and 1 positive sample for each anchor. By augmenting the negative sample size for each input, we anticipate that the model will better learn to distance itself from numerous negative samples.

From the validation accuracy, we can observe that there is a steady increase of accuracy rate as we increase the number of negative sample size. This suggests that training the model with more negative samples per tuple enhances its ability to distinguish the anchor from a broader spectrum of negative examples. The validation accuracy of N=15 is the highest 61%. This affirms that our previous investigation is valid. However, due to limited computational resource, we were unable to experiment with higher values of N.



**Figure 8:** Validation Accuracy - Triplet Loss vs (N+1) Pair

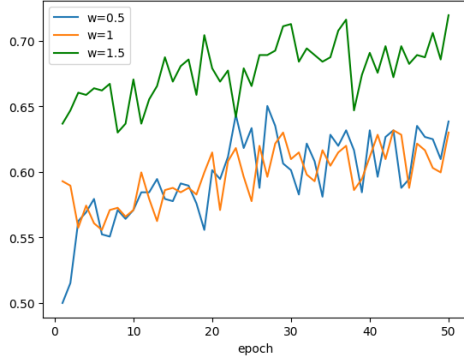
### 5.2.4 BINARY CROSS ENTROPY VS WEIGHTED BINARY CROSS ENTROPY

We discover that Binary Cross Entropy Loss gives the best accuracy among all losses for CNN-based encoders. Its effectiveness depends greatly on the encoder we used as well as the configuration of the classifier network. For the model configuration that process the embedding through ways such as  $X = (X_1^2) \oplus (X_2^2) \oplus (X_1 X_2)$  where  $X_1 = \text{embedding1} \oplus \text{embedding1}$  and  $X_2 = \text{embedding2} \oplus \text{embedding2}$ , a VGG encoder(0.79) performs marginally better than a ResNet encoder(0.77). However, if we modify the middle concatenation part to  $X = (X_1 - X_2) \oplus (X_1^2 + X_2^2) \oplus (X_1 X_2)$ , then ResNet(0.81) will outperform vgg encoder(0.77).

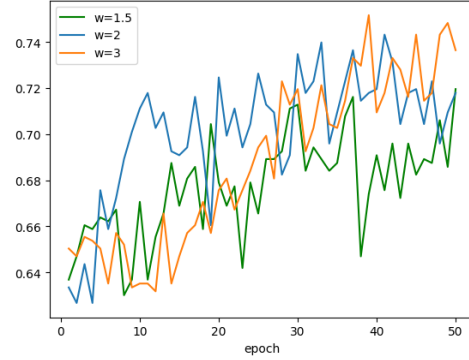
For CNN-based encoders such as ResNet and VGG, the accuracy varies between 0.7-0.8 for a single model.

As illustrated in our EDA section, we observed that majority of the families have 3 individuals, indicating that each of the individuals has only 2 positive classes with limited number of samples. In the other hand, if we consider negative samples, they could be any other individuals from different families, where this number is much greater than that of positives. Hence, on top of the BCEWithLogitsLoss function, we would like to assign weights to positive classes to deal with this imbalanced nature of our dataset. By assigning a higher weight to the positive class, model is told to pay more attention to correctly classifying positive instances.

From Figure 9, we can tell that by increasing the positive weights to 1.5, the validation accuracy is around 10% higher than original BCEWithLogitsLoss with no weights adjusted ( $w=1$ ). We also tried reducing the positive weight to 0.5, the result is poor as expected. Figure 10 shows the other weights we have experimented, there is no significant improvement on the accuracy rate as we increase the positive weights further. Hence, we can conclude that the optimal positive weight value is around 2.00, giving a testing accuracy of 75%.



**Figure 9:** Accuracy with Weights 0.5, 1, 1.5



**Figure 10:** Accuracy with Weights 1.5, 2, 3

However, even though the performance of the model did improve by 10% in terms of accuracy with positive weights adjusted, the AUC is not changing that significantly. The AUC demonstrates a more modest improvement, hovering around 4%, ultimately reaching a value of 83%. This suggests that the model's discrimination ability, as measured by AUC, is relatively robust to the choice of positive weights.

#### 5.2.5 COMPARISON AMONG LOSSES

The table below shows the comparison of how the losses affect the final result.

Loss with the best configuration	score
Contrastive Loss	0.5
Circle Loss	0.6
(N+1)-Tuplet Loss	0.61
Binary Cross Entropy Loss	0.81
Weighted Binary Cross Entropy Loss	0.83

**Table 1:** comparison results between losses

### 5.3 VISION TRANSFORMER ENCODER

Noticing the results from CNN with different losses, we then move on with experiments with vision transformer (ViT). Since the BCELoss showed the best performance on CNN, we generally use a Siamese network with auxiliary classification head with ViT encoder and BCELoss for the following experiments unless otherwise stated. The ViT used is pretrained on MS-Celeb-1M datasets of human faces (Zhong & Deng (2021)).

#### 5.3.1 COMPARISON WITH CNN

Since Vision Transformer (ViT) and the CNN used in the previous section differs significantly in how they process and represent image, we would like to compare their performance specifically to our task of kinship classification.

To ensure a fair comparison, we have maintained consistency in various aspects, including data pre-processing and hyperparameters such as batch size. Additionally, we employ the same set of features (i.e.  $x_1 * x_2$  etc.) that showed the highest private score in CNN models. Two models, one with fixed ViT model weights (m1) and the other one with the last encoder block unfrozen (m2) are trained, the results are shown in Table 2).

Model name	private score
Best CNN score	0.83
ViT Fixed Weights	0.772
ViT Finetuned Last Encoder	0.841

**Table 2:** comparison results

The training and validation accuracy plots are depicted below. The model with fixed weights doesn't show severe overfitting, however, it seems to reach a bottleneck at 0.75 accuracy, proving challenging to enhance further. On the other hand, the model with the last encoder block fine-tuned shows better fitting with higher accuracy. However, after 10 epochs, overfitting issues emerge despite the implementation of higher dropout rates (0.4) compared to the CNN models (0.1; 0.01).

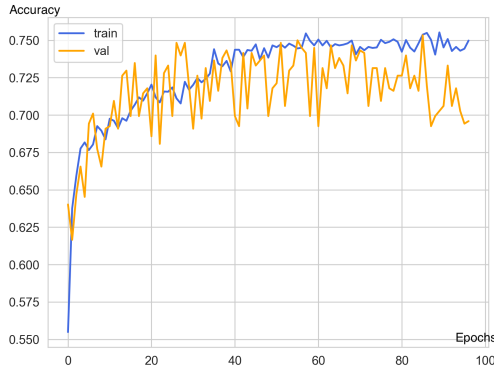


Figure 11: ViT Fixed Weights

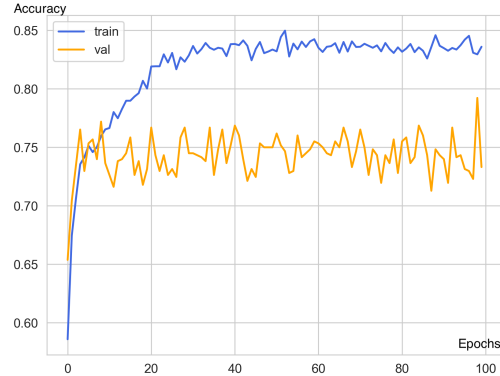


Figure 12: ViT Finetuned Last Encoder

	<b>m2_loss_0.841.csv</b>			
	Complete (after deadline) · now			
	<b>m1_loss_0.772.csv</b>			
	Complete (after deadline) · 1m ago			

<b>0.841</b>	<b>0.834</b>	<input type="checkbox"/>
<b>0.772</b>	<b>0.775</b>	<input type="checkbox"/>

Since the finetuned model showed better performance, we decided to focus on finetuning and solving the overfitting issues, which will be discussed in the next sections.

### 5.3.2 DEALING WITH OVERFITTING

Apart from conventional measures like reducing model complexity and adding dropouts, we explored an innovative approach that unexpectedly yielded the most promising results for a single ViT-like model. While keeping other hyperparameters consistent, we randomly selected one encoder block for fine-tuning every 10 epochs. In total we ran 100 epochs and randomly tuned 10 encoder blocks. Two methods are tried, one (m3) by turning `layer.requires_grad_(False)` in place. The other one (m4) by creating new optimizer for trainable parameters since the layers with 0 gradients might still be updated with Adam optimizer. The code examples and top score for each training strategy are shown below:

```
def gen_trainable(model):
    layer = str(np.random.choice(20))
    trainable_params = []
    for name, param in model.named_parameters():
        if 'encoder' in name:
            if layer in name:
                param.requires_grad_(True)
                trainable_params.append(param)
            else:
                param.requires_grad_(False)
        else:
            param.requires_grad_(True)
            trainable_params.append(param)
    return trainable_params
```

```
def gen_trainable(model):
    layer = str(np.random.choice(20))
    for name, param in model.named_parameters():
        if 'encoder' in name:
            if layer in name:
                param.requires_grad_(True)
            else:
                param.requires_grad_(False)
        else:
            param.requires_grad_(True)
```

Model name	private score
m3 (change <code>requires_grad_</code> )	<b>0.895</b>
m4 (create new optimizer)	0.884

	<b>m4_loss_0.884.csv</b>			
	Complete (after deadline) · now			
	<b>m3_acc_0.895.csv</b>			
	Complete (after deadline) · 23s ago			

<b>0.884</b>	<b>0.872</b>	<input type="checkbox"/>
<b>0.895</b>	<b>0.886</b>	<input type="checkbox"/>

### 5.3.3 ViT WITH ADDITIONAL LINEAR PROJECTION

As the current ViT is trained for classification, it only has self-attention between different patches. In our case, the ability for one image to know which part to pay attention to in the other image would be helpful. For example, if the two images are relatives, we would expect similarities between eyes, noses etc. Hence, we applied additional linear projection to each input to the encoder layer, and add it to the query of the encoder of the other image, incorporating information from the other image for it to attend to (Figure 13). Interestingly, despite the introduction of extra parameters, the model (m5) doesn't show overfitting, but the training and validation accuracy stuck at around 0.68 (Figure 14). It seems to suggest that the addition of linear projections helps increase the model's capacity, allowing it to capture more complex patterns in the training data and in turn reduced overfitting. However, it lead to increased difficulty in optimization and training, especially when the amount of data is limited. The final Kaggle private score is **0.74**

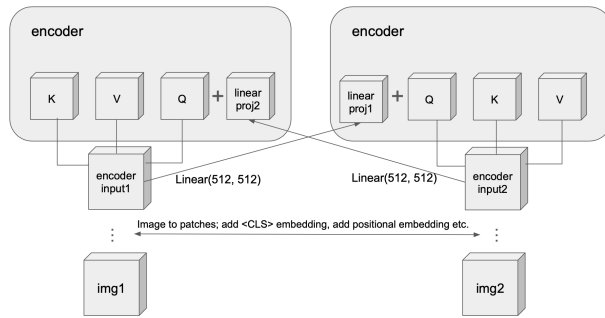


Figure 13: Design of Additional Linear Projection

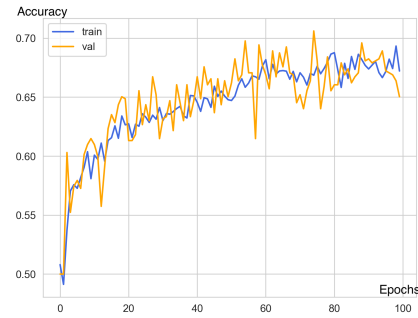


Figure 14: Training Logs

**m5\_loss\_0.739.csv**  
Complete (after deadline) · 13s ago

**0.739**

**0.723**



### 5.3.4 ViT WITH CONTRASTIVE LOSS

Similar to our exploration with CNN, we also experimented with contrastive loss in the context of ViT (m6). In this approach, we avoided training features like  $x_1 * x_2$  and instead directly fed pairs of embeddings of the  $<CLS>$  token for each image to the loss function. All layers were fine-tuned with a learning rate of  $1e-5$ . Notably, during training, both the training and validation accuracy stuck at approximately **0.65**. However, the final Kaggle private score, assessed using AUC, reached **0.81**. This discrepancy suggests that while the overall model accuracy may lag behind ViTs trained with BCELoss in previous sections, the contrastive loss strategy creates a more substantial margin for classification, resulting in a relatively satisfactory AUC score though the accuracy is not high enough.

**loss\_contrast64.csv**  
Complete (after deadline) · 17d ago

**0.813**

**0.805**



### 5.3.5 RESULTS FROM ViT EXPERIMENTS

The table summarizes the results from the above ViT experiments.

Model ID	Model name	private score
m1	Fixed Weights	0.772
m2	Finetuned Last Encoder	0.841
m3	Finetuned randomly selected encoder (change <code>requires_grad_</code> )	<b>0.895</b>
m4	Finetuned randomly selected encoder (create new optimizer)	<b>0.884</b>
m5	Additional linear projection	0.74
m6	Contrastive loss	0.81

Table 3: Results from ViT experiments

## 6 SELECTED RESULTS

Ultimately we aim to use an ensemble of multiple methods, and to that end, we individually test various versions of all of the above proposed ones. Based on these results, we shortlist the best performing ones to be part of the ensemble (see Table 4).

Model	Train Strategy	Join Function	Private AUC	Public AUC
Siamese ConvNet	Unfreeze	$x_1, x_2, (x_1 - x_2)^2, x_1^2 - x_2^2$	0.866	0.851
Siamese ViT	Rand. Select	$x_1 - x_2, x_1^2 + x_2^2, x_1 * x_2$	0.895	0.886
Siamese ViT	Rand. Select	$x_1 - x_2, x_2 - x_1, \text{signedsqrt}(x_1^2 + x_2^2), x_1 * x_2$	0.859	0.848
Siamese ViT	Rand. Select	$x_1 - x_2, \text{signedsqrt}(x_1^2 + x_2^2), x_1 * x_2$	0.884	0.884
Siamese ViT	Rand. Select	$x_1 - x_2, x_1, x_2, \text{signedsqrt}(x_1^2 + x_2^2), x_1 * x_2$	0.895	0.878

**Table 4:** Summary of Results from Siamese Networks

where

- $\text{signedsqrt}(x) = \text{sign}(x) * \sqrt{\text{abs}(x) + 1e - 10}$

### 6.1 FINAL ENSEMBLE














Our final ensemble is composed of the Siamese ConvNet, the three best-performing Siamese ViTs, and finally, three publicly submitted solutions (all based on VGG-Face nets).

Ensemble Type	Private AUC	Public AUC
Averaging	0.922	0.919
Weighted Averaging (private-score)	0.922	0.919
Weighted Averaging (private-score-pow(2))	0.922	0.919
Weighted Averaging (private-score-pow(6))	0.922	0.920
Weighted Averaging (private-score-pow(12))	0.923	0.920
Weighted Averaging (private-score-ranked)	0.916	0.914
Weighted Averaging (private-score-corr)	0.922	0.919
Cluster-Based	0.5	0.5
Binning	0.905	0.89
Max-conf	0.9	0.902

**Table 5:** Results of the different Ensembling Methods

As can be seen in Table 5 the best performing method is *Weighted Averaging (private-score-pow(12))*, a slightly aggressively weighted average. Surprisingly, this simple technique in combination with simple but strong base models was enough to exceed the current top-1 performance (See Figure 15 and Figure 16).

Public Private

This leaderboard is calculated with approximately 50% of the test data. The final results will be based on the other 50%, so the final standings may be different.						
#	Team	Members	Score	Entries	Last	Solution
1	partham		0.919	157	4y	
2	lynesyChen		0.918	173	4y	
3	mattemillo		0.917	169	4y	
4	AlexeyK		0.916	64	4y	
5	maki		0.916	40	4y	
6	SCUT111		0.914	115	4y	
7	DAMA		0.914	114	4y	
8	Chaohui Song		0.913	32	4y	
9	dasdasd		0.913	23	4y	
10	dsad		0.913	19	4y	

**Figure 15:** The current state of the public leaderboard.

 <b>submission_all_models.csv</b> Complete (after deadline) · now	<b>0.923</b>	<b>0.92</b>	<input type="checkbox"/>
 <b>submission_just_our_models.csv</b> Complete (after deadline) · 2m ago	<b>0.912</b>	<b>0.912</b>	<input type="checkbox"/>

**Figure 16:** Our final score.

As mentioned, we augmented our ensemble with 3 previous submissions. To show that our models contributed significantly, we compared our final ensemble with one only using our own models. The ensemble composed of solely our models (using the same ensemble strategy as for our top-1 submission), achieved a public score of 0.912. Thus, adding the three external models increases our public score by 0.008.

Thus, solely using our models, we still achieve a top-14 (top 3%) ranking on the public leaderboard (so if we are not allowed to utilize previous models, please use that submission as the final one). It is worth pointing out that the three public models we used were VGG-face-2 conv-nets fine-tuned on the dataset, and with enough compute available, it should be no problem to re-create them ourselves (especially since the training strategy was published).

## 7 CONCLUSION

In this project, we leveraged the Siamese network architecture with two primary frameworks for computer vision tasks: Convolutional Neural Networks (CNN) and Vision Transformer (ViT). Our exploration involved experimenting with diverse loss functions for CNN and introducing innovative training strategies and architectures for ViT, both of which had not been previously explored on Kaggle. Additionally, we exploited various ensemble techniques, extending knowledge learned from lectures to real-world scenarios. While the experimentation with different losses did not yield significant improvements for this specific task, it helped us gain a deeper understanding of different losses and model behaviors. Despite the outcome, the insights gained contribute valuable lessons for future endeavors in similar tasks. ViTs generally showed better performance in our experiments, and we also noticed how ViT is usually prone to overfitting especially with limited data. Effort was made to deal with this classical challenge of overfitting, by trying out both conventional approaches and innovative approaches which surprisingly yielded the best results. It showed that with enough persistence and creativity, it is always possible to further improve scores. Ultimately, we truly saw the power of ensembles. When combining a number of decently performing models in an ensemble, we are able to achieve a great performance.

## 8 CONTRIBUTIONS OF EACH GROUP MEMBER

All group members contributed equally to all parts of the group project.

## REFERENCES

- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL <http://arxiv.org/abs/1512.03385>.
- Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *CoRR*, abs/2201.03545, 2022. URL <https://arxiv.org/abs/2201.03545>.
- Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.
- Yaoyao Zhong and Weihong Deng. Face transformer for recognition, 2021.
- Github of loss exploration: <https://github.com/charlottacao41/face>