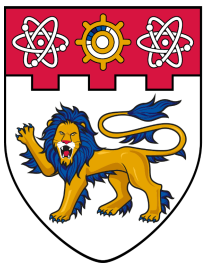


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC4000 Machine Learning Project

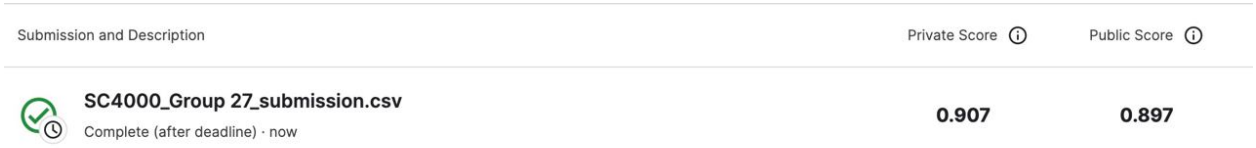
Group: 27

Contributor Name	Matriculation Number	Contribution
Mishra Apurva	U2120474C	Ensemble learning, Siamese research
Gupta Tushar Sandeep	U2123144E	Introduction & Problem Identification, Debugging, Video Editing, Table 1, Research ViT
Jadhav Chaitanya Dhananjay	U2121503D	Research and Implementation of Model 1, Model 2 - Siamese Networks
Sahoo Nirvik	U2123483H	Research and Implementation of Model 3 - Vision Transformer
Sharma Divyansh	U2023693D	Leader, Preparing Hard Negatives, Siamese Net with Triplet Loss

Youtube Presentation Video Link: https://youtu.be/ksbq0ld7UvU

Table of Contents

1. Kaggle Results



The csv file used to make this submission is attached along with this report as kaggle_submission.csv

1.1. Evaluation Score

Public Score	Private Score
0.897	0.907

1.2. Ranking Position

Public Leaderboard	Private Leaderboard
Top 97	Top 33

2. Problem Identification

2.1. Problem Statement

‘Can you determine if two individuals are related?’

A complex model is to be built to determine if two people are blood-related based solely on images of their faces. The purpose of the model is to analyse and extract various features from the images of 2 people’s faces in order to determine whether they are similar enough to be classified as ‘kin’.

2.2. Challenges

1. The images of people are not in a standardised format, thus the model may learn features which are irrelevant to the kinship recognition task. For example:
    - The images are shot at different locations, and shadows are present on the faces of the people due to poor and uneven lighting.
    - There are numerous images of people wearing caps, and other objects close to their face which can mislead the model into finding non-kins similar.

- People of the same age, especially children, or the same gender, due to features like facial hair, may look similar despite being from different families.
- There are images where there are more than one face present in them.
- There are several images which have text written across the faces of the people.

2. The images provided in the sample have varying clarity, thus our solution must be flexible enough to detect facial features across a wide variety of photo qualities.
3. Moreover, training_relationships.csv only contains positive examples of people who are kins. Thus appropriate size and sample of negative pairs must be created.

3. Siamese Neural Networks (SNN)

3.1. Introduction to Siamese Networks

Siamese networks are a special application of neural networks, where a single network learns the similarity (closeness) or distance (difference) between two input data points. In Siamese networks, the focus is on learning the semantic similarity between two inputs taken together, as opposed to learning features of two inputs taken independently. This is implemented in the form of two identical convolutional subnetworks, which have the same weights and parameter updates. In practice, a single subnetwork can be treated as two when passing a pair of inputs.

SNNs can be used for class prediction. The subnetworks produce embeddings for each input image on which operations are performed such as Euclidean distance, Manhattan distance, cosine similarity, etc. The operation in turn produces a scalar that then becomes a metric to make predictions on closeness between the two input images.

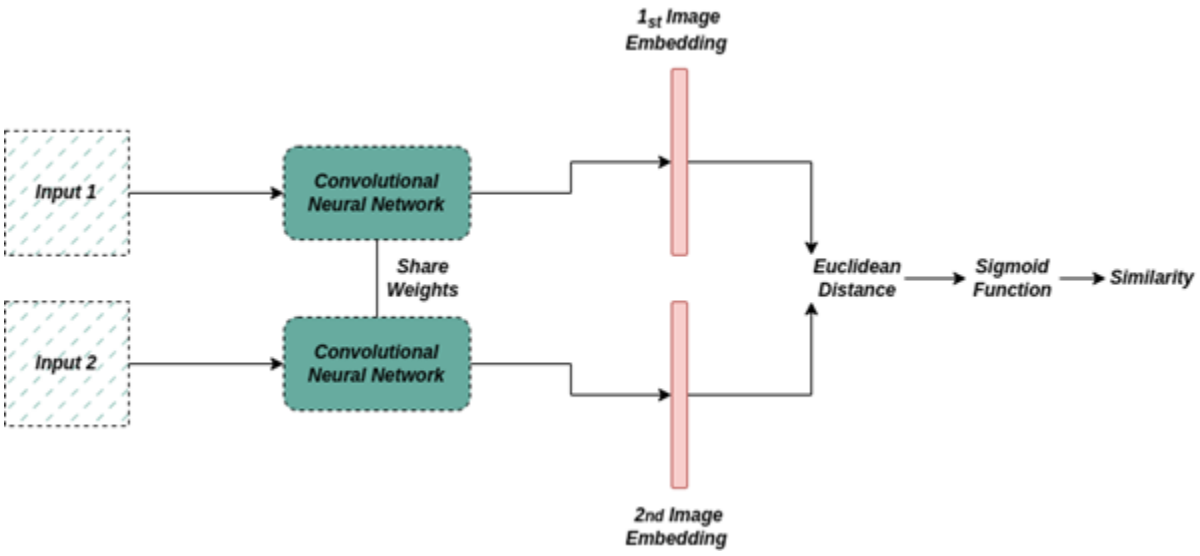


Figure 1. Vanilla Siamese network architecture.

3.1.1 Advantages of Siamese Networks

Since the objective is to learn a distance function, a Siamese network can generalise well over unseen data points. This is the main advantage that the Siamese network has over traditional classification models. Figure 2 shows a face recognition system example which is trained using a traditional classification model. In face recognition, a model is supposed to identify an image of a person and output the person's name or identity. However, as demonstrated in the figure, it does not work well when it encounters a data point that is outside its trained data. A face verification system on the other hand implements a Siamese network which is able to generalise better over unseen data points.

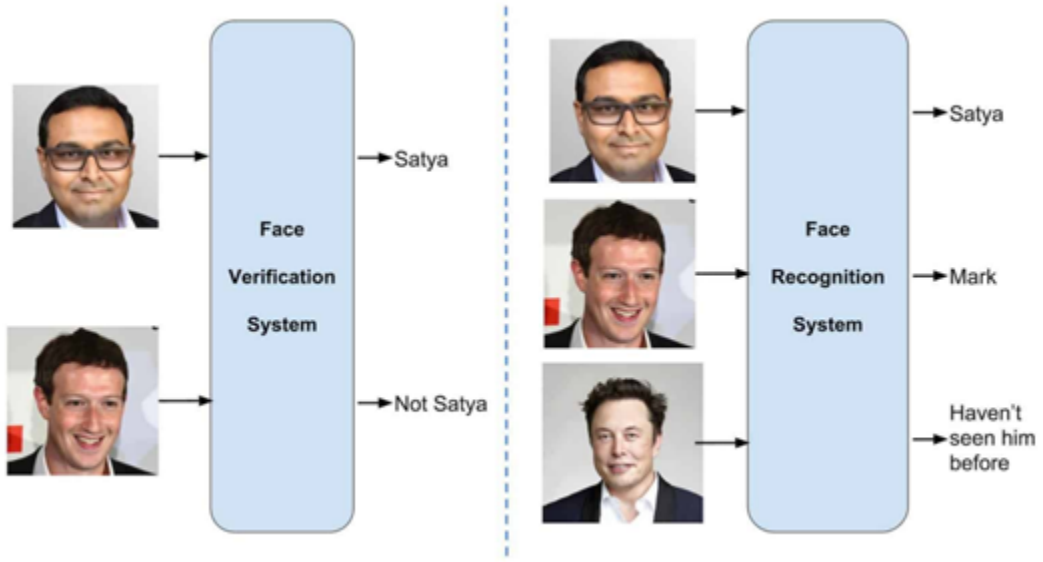


Figure 2. *Face Verification System* (Left) using Siamese Networks. *Face Recognition System*(right) using traditional classification model.

A Siamese network has many more advantages. A trained Siamese network that has learned to discriminate is very capable of handling noise and variations in the input data. For example, a person's pose, accessories, face tilt, expression, etc., are less likely to affect the distribution. Along with noise, the Siamese network is a suitable choice in problems with imbalance of data.

3.1.2. Applying Siamese Networks to Kinship Detection

Motivated by the success of Siamese networks in the face verification domain, we decided to employ Siamese models to solve kinship detection. We propose that face verification systems and kinship detection are intuitively similar problems. In this case of kinship recognition, pairs of images can be fed into the Siamese network to classify as kin or not kin. The semantic similarity in the representations (embeddings) can be learned by the SNN. SNN usually performs better than CNN when only small data is available, which may be true for the SMILE dataset.

A drawback is that training SNNs are more computationally intensive than normal classification. Moreover, the output of an SNN is not probabilities, but distance from each class, which may need to be converted into a form appropriate for the problem (such as picking a threshold.) Usually, contrastive loss or triplet loss functions are used for training a Siamese network.

In the following sections, we introduce two model architectures. The first section reviews the model architecture used in submissions on Kaggle. The next Siamese network model is a novel solution that our team proposed.

3.1.3 Basic Siamese Model: Current Popular Choice for Kinship Verification

Our team conducted a testing evaluation for the publically available submission codes and identified a recurring pattern in all submitted models. Mainly, all models propose the architecture described in Figure 3. Given two input images, a pre-trained convolutional neural network model produces embeddings. Operations are carried out on the embeddings to produce tensors that highlight relationships between the embeddings. The results of these operations are fed into a binary classification model that finally predicts a 0/1 classification. The loss function used for this task is a Binary Cross Entropy Loss.

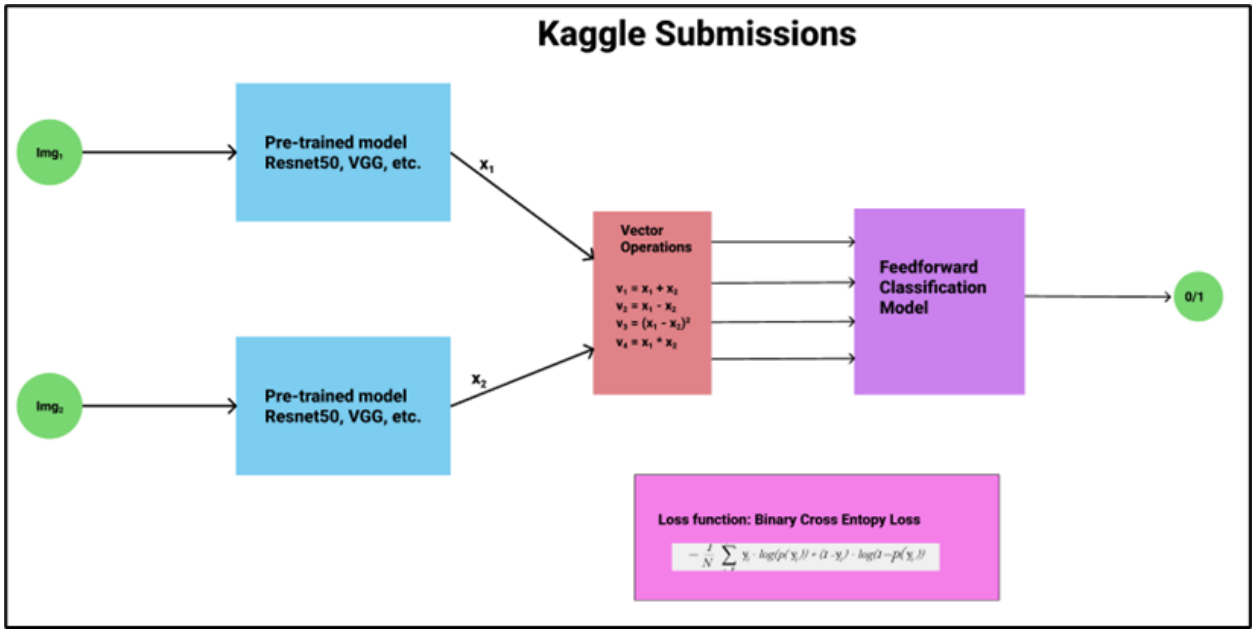


Figure 3. Showcases the basic Siamese Network model used on Kaggle

Nearly all submissions (including the winners) stem from this architecture and vary in generally 4 areas: pre-processing of the input, choice of embedding model, vector operations, classification model architecture, and the particular choices of each one used to lead to production of high performing models.

3.2 Proposed Model: Siamese Network with Triplet Loss

Our first proposed solution modifies existing Siamese architectures to learn and output a distance function that is then used to classify kinship in the images. The following are the modifications proposed.

1. Remove Vector Operations: We assume that our model can self-identify and optimise vector operations by itself. Therefore, we remove the vector operations and directly feed the embeddings to the feedforward neural network

2. Triplet Loss for Training: This architecture replaces the binary cross entropy loss.

3. Unfreeze last layers of the embedding model: This is done for fine-tuning the embedding model to produce better embeddings suited for our task.

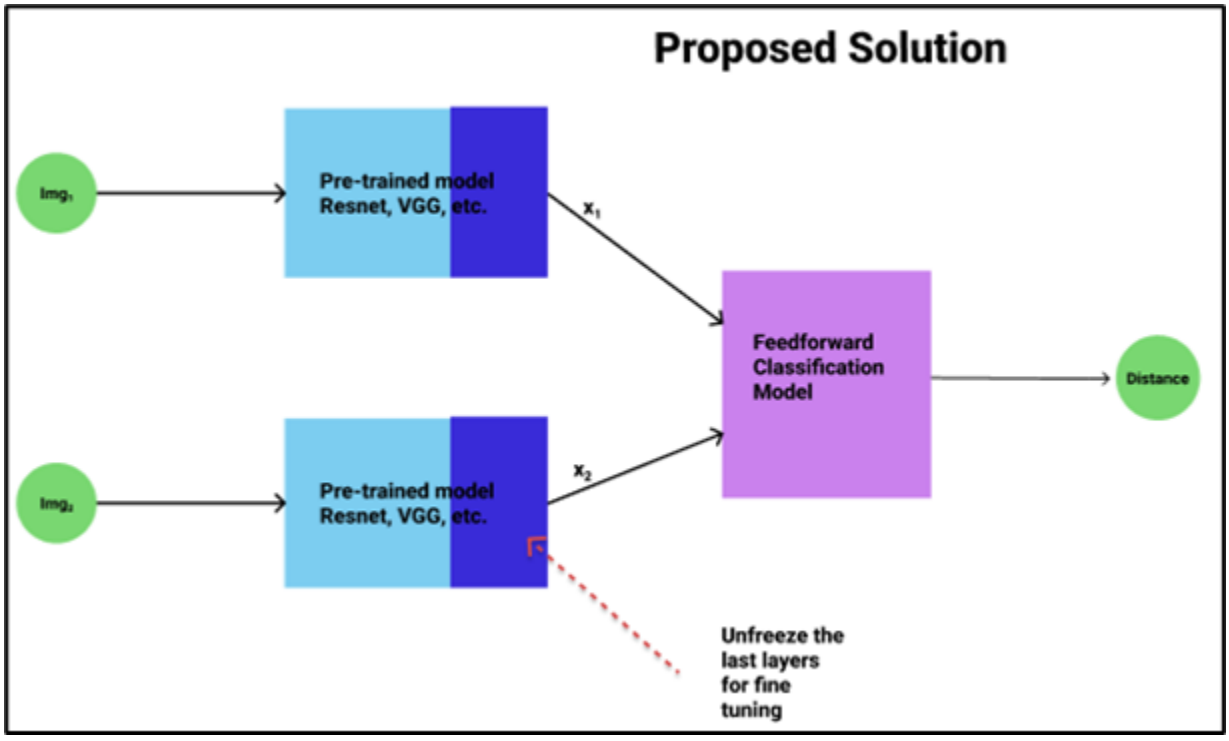


Figure 4. Showcases the modified Siamese Network model

3.2.1. Model Training

To train our Siamese model, we employ the triplet loss function, a metric specifically designed for learning similarity and dissimilarity between data points. This loss function requires three images as inputs: anchor image, positive image, negative image. This triplet is carefully chosen such that the anchor image and positive image are related while the anchor image and negative image are unrelated. Given distances produced from our model, the objective is to **minimise** the

distance between **images of related individuals** (anchor and positive) while **maximising** the distance between **images of unrelated individuals** (anchor and negative), effectively creating distinct clusters in the feature space. We introduce a margin parameter, set to 0.5 to enforce a minimum separation between positive and negative distances. The model's parameters are updated using the gradient of the triplet loss function, facilitating effective learning of kinship-related features.

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

Figure 5. Shows the Triplet Loss function. L(A,P,N), A, P, N, f and α are Loss function, anchor image, positive image, negative image, and distance function respectively

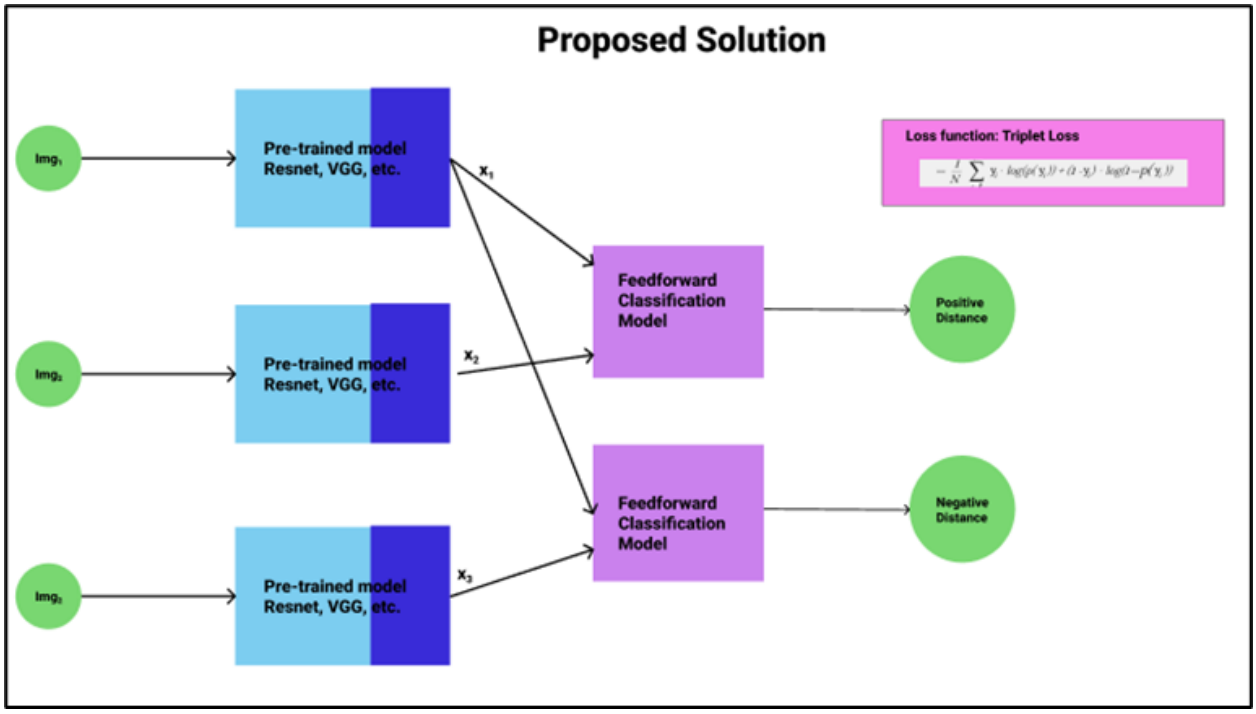


Figure 6. A Siamese network trained with triplet loss.

3.2.2. Preparation of Hard Negatives

Recognizing the importance of challenging the model with hard negative examples during training, we undertake the task of identifying such instances. Hard negatives are negative training examples that are challenging for the model to classify correctly. In our context, these

are images of unrelated individuals who may bear a visual resemblance. Visual similarity alone does not guarantee kinship, and we aim to equip our model with the ability to make nuanced distinctions.

To identify hard negatives, we leverage external tools, such as DeepFace, a machine-learning library capable of extracting features from faces and estimating factors such as including age, gender, and race attributes. By considering individuals of the same age, gender, and race as potential hard negatives, we enrich our training dataset with challenging examples that promote the model's ability to discern kinship beyond mere visual likeness. The figure below shows an example of a picture of a grandfather (left) and his daughter (middle). However, the grandfather resembles the picture of a man (right) who is as old as him. We want our model to look more closely into kinship features and not just return similarity of the faces.

Figure 7. Illustrated Example of Pairs used for Kinship Detection

3.3.3. Model Prediction of Kinship

A challenge still remaining to solve is how to convert the distance values to classification probabilities and what threshold to choose. Currently, our model only outputs ‘kinship distance’ between two images. We need to apply a threshold value for distance such that for any given images, if the calculated kinship distance is more than threshold, then they are unrelated. If the distance is less than the threshold, then they are classified as blood-relatives. To derive this threshold value, we employ a validation dataset. This validation dataset contains both an equal number of positive and negative examples that were unseen by the model during training. We calculate the mean of distances between all positive pairs, and similarly calculate the mean of distances between all negative pairs. Our threshold value lies in between the calculated positive mean and negative mean.

$$\mu_{positive} = \frac{1}{N}\sum_{i=1}^{N} D(Pval\,(img1_i, img2_i)) \qquad \mu_{negative} = \frac{1}{N}\sum_{i=1}^{N} D(Nval\,(img1_i, img2_i))$$

$$r_{threshold} = \frac{\mu_{positive} + \mu_{negative}}{2}$$

$$pred = 1(net(x) < r_{threshold})$$

Pval is a validation dataset containing all positive training examples. *Nval* is a subset of the validation dataset containing all negative examples. *D* is the learned distance function. The above equations calculate the mean of all positive distances and the mean of all negative distances. is the chosen value which is used to decide kinship from the output of Siamese net. Note that 1 means kinship and 0 means the opposite.

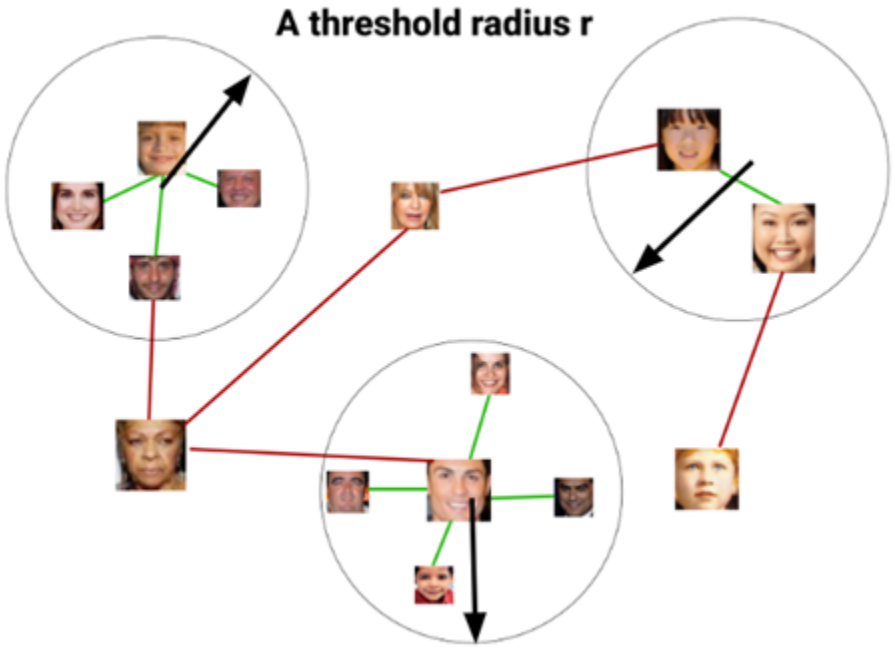


Figure 8. Visualisation of positive and negative distances. The circles represent circles with radius = $\mathbf{r}_{\mathbf{threshold}}$.

3.2.4. Results and Discussion for Kinship Verification

Despite training with different models and changing the learning rate, our model was not able to learn. There could be various reasons such as training examples were too hard, quality of the dataset, underestimating the hardness of the problem. Nevertheless, this unsuccessful attempt contributed by helping us understand that kinship detection is not as easy as face verification.

In kinship verification, the model must discover patterns in human faces that are carried in kins and then correctly identify them over similar looking individuals. Therefore, we require the model to 'attend' to certain features. This motivated half of our team to research and implement vision transformers which have a self-attention mechanism while others continued to enhance feature engineering.

Even though we failed at this Siamese network, we actually created a huge dataset of hard negative examples that was once not present. We believe that training on this dataset creates better models compared to training on conventional datasets where negative examples are chosen randomly. So the next models described here are all trained on the hard negative examples dataset. In addition, we also contributed to the hard negatives example dataset to the Kaggle Community.

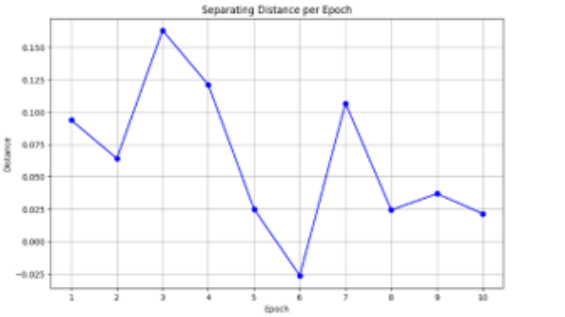


Fig. shows a plot of separating distance ($\mu_{negative} - \mu_{positive}$).

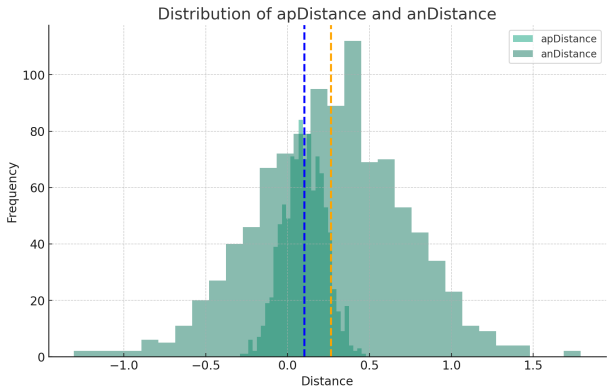


Figure 9a. **Figure 9b.**

Figure 9a. Representation of the Separating Distance between positive and negative samples, Figure 9b Distribution of positive and negative distances for samples.

Figure 9b shows the distribution of positive and negative distances. The graphs are merged into each other, whereas ideally they should be distant apart. This affected the ability of our model to train on this dataset.

3.2.5 Results

Despite training with different models and changing the learning rate, our model was not able to learn. There could be various reasons such as training examples were too hard, quality of the dataset, underestimating the hardness of the problem. Nevertheless, this unsuccessful attempt contributed by helping us understand that kinship detection is not as easy as face verification.

S/N	Images	Image Description
1.		Images of people wearing caps
2.		Black and white photos
3.		Faces with another face in the background
4.		Tilted partial face in black and white
5.		Image is not in clarity
6.		Image is in clarity
7.		Image with text written across them
8.		Images with shadows on face due to poor/uneven lighting
9.		Images with people wearing sunglasses / spectacles

Table 1. Representing various types of images present in the dataset provided and their respective descriptions.

4. Proposed Models

4.1 Model 1 - Targeted Siamese Model

Our first model uses a Siamese architecture for comparing pairs of images that are represented in RGB. It was able to achieve a score of 0.889 on the Kaggle leaderboard through the use of transfer learning, custom fully connected layers and feature combination.

4.1.1 Model Architecture

The first layer of the model uses InceptionResnetV1, pre-trained on the 'vggface2' dataset for obtaining the base encodings. Through domain research, we found that this model is known for its efficiency in extracting facial features and is trained on a large dataset. The last five layers of the encoder were made trainable to allow the InceptionResnetV1 model to adapt to the requirements of kinship detection. The encoder outputs two sets of embeddings, referred to as $emb1$ and $emb2$, for the two input images. Initially, these embeddings were passed directly to our connected layers, which led to sub-par results. We then began to explore feature engineering and began trial and error of various preprocessing operations that can be carried out on $emb1$ and $emb2$. Our best performing model calculates and concatenates the following features before passing it to our fully connected layers:

1. **Squared Difference of Individual Embeddings:**

$$x1 = emb1^2 - emb2^2$$

This operation highlights the individual differences in facial features. Squaring the embeddings and finding their difference emphasises the distinctive features of each face, which is very informative when training for kinship detection.

2. **Squared Element Wise Difference:**

$$x2 = (emb - emb2)^2$$

This operation is able to effectively bring out the differences in embeddings between two images. Squaring the differences ensures that even the most subtle differences in embeddings are made prominent. Again, this is useful in kinship detection where small variations can lead to significant results.

3. **Element-Wise Product:**

$$x3 = emb1 * emb2^2$$

This operation captures the similarity between the embeddings. If both embeddings have high values in the same dimensions, the product will be high, indicating similarity in those features.

4. **Element-Wise Difference:**

$$x4 = emb - emb2^2$$

This operation is a straightforward comparison that outputs a linear measure of the similarity between features.

5. **Sum of Squared Differences**

$$x5 = \sum_{i=1}^{n} (emb1_i - emb2_i)^2$$

This operation computes the sum of squared differences between corresponding elements of the two facial embeddings, emb1 and emb2. This metric quantifies the overall dissimilarity between the two embeddings, capturing the cumulative difference across all dimensions of the embedding space

6. **Cosine Similarity**

$$x6 = \frac{\sum_{i=1}^{n} emb1_i * emb2_i}{\sqrt{\sum_{i=1}^{n}}}$$

This operation measures the similarity between two facial feature vectors regardless of their magnitude. This is important for identifying kinship from facial feature embeddings.

7. **Euclidean Distance**

$$x7 = \sqrt{\sum_{i=1}^{n} (emb1_i - emb2_i)^2}$$

This operation quantifies the overall distance between two vectors in our embedding space and therefore reflects the similarity between two facial feature embeddings.

Finally, $x1, x2, x3, x4, x5, x6, x7$ are concatenated and passed to our fully connected layers. By performing these computations on top of our accurately extracted features, we are able to fine-tune ensure that our model is solely targeted towards learning how differences in embeddings can be used to predict kinship.

These layers are designed to process our processed features. The first linear layer reduces the dimensionality of the feature from 2051 to 200. This reduction in feature space is essential for focusing on the most relevant information for kinship detection. A batch normalisation layer was positioned after the first linear layer to normalise the output of each layer across each batch. It uses the mean and variance for each feature in a batch to normalise the output for the neurons in the first layer. This prevents activations from becoming too high or too low, avoiding the

problems of exploding gradients or vanishing gradients. Overall, it ensures a more stable and efficient training while helping our model achieve generalisation.

A ReLU activation function was placed after every linear layer, which outputs the input directly if it is positive, or outputs zero otherwise. This layer was chosen to introduce non-linearity into the model to allow our network to learn and represent more complex functions. This is essential for kinship detection, where the relations in the data are not linear or straightforward.

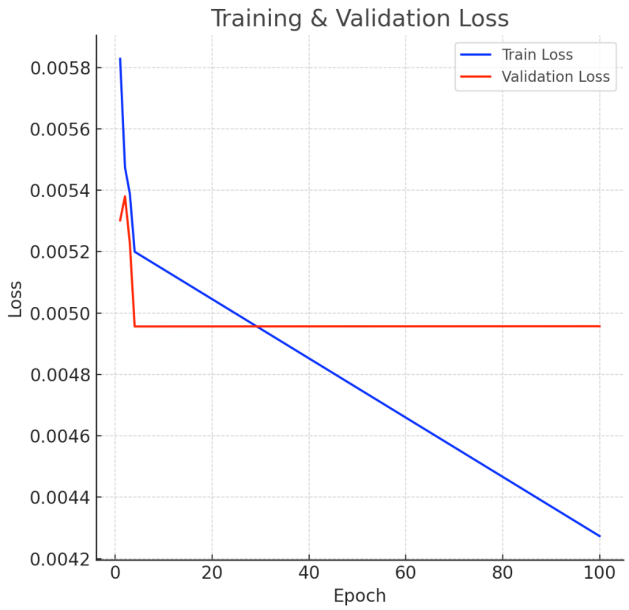
Lastly, the output layer of the network consolidates all information into a single value which represents the likelihood of kinship between two different images. A diagrammatic representation of this model, visualised in Python, is available in the file Model1.pdf attached along with this report.

4.1.2. Training

This model was trained for 100 epochs with early stopping and a patience of 10. The initial learning rate was set at 1×10^{-3}, which is a popular baseline. This was dynamically adjusted by the Adam optimizer and learning rate scheduler that adjusted the learning rate during training to achieve faster generalisation and prevent overfitting.

The loss function used was Binary Cross-Entropy with Logits Loss. This loss function was chosen due to its suitability for handling binary classification tasks (in this case, kinship or no kinship). It combines a sigmoid activation function with Binary Cross-Entropy loss into a single function. This approach is more computationally efficient and is shown to have sensitivity to class imbalances.

The best performing epoch from this training set was used to perform the predictions on the test set. The plot of the training and validation losses and accuracy is shown in the figure below, which indicate steady increases in train and validation accuracy. Hence, we can conclude that our model is not overfitting.



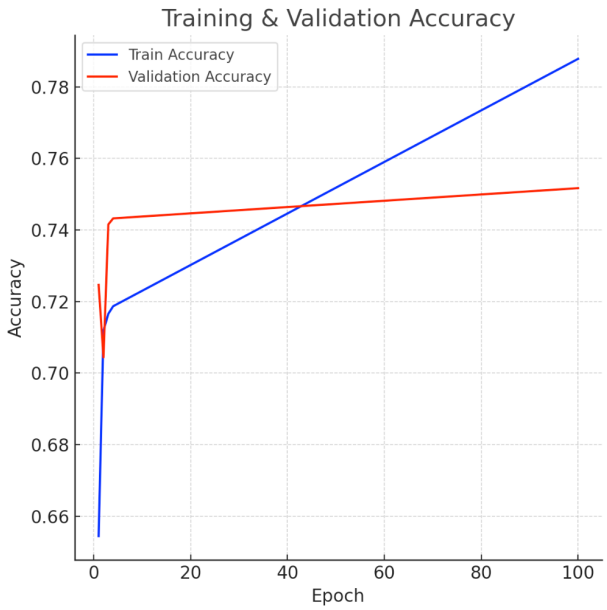


Figure 10a. **Figure 10b.**

Figure 10a. Representation of the Training and Validation Loss, Figure 10b Representation of Training and Validation Accuracy

4.2 Model 2 - Non-Correlated Siamese Model

The results from our first model revealed that we were on the right track. Our findings showed that the use of transfer learning on top of a custom model architecture is able to produce effective results. Given the limited scope of the training data, we recognized that it would be very difficult to create a single model that ranks well on the Kaggle leaderboard. We changed our strategy to create multiple non-correlated models and adopt Ensemble Learning to boost our final submission score.

The small training set provided to us in this competition meant that we could not use methods such as bagging and boosting. The complex nature of kinship detection required our model to have access to all the training data available for training. Therefore, we needed to explore other ways of introducing non-correlation in our data. A diagrammatic representation of this model, visualised in Python, is available in the file Model2.pdf attached along with this report.

4.2.1 Introducing Non-Correlation

We identified three methods of introducing non-correlation to the solution described in Model 1 to create Model 2:

1. **Changing the encoding layer:** we use the same InceptionResnet model for generating our encodings. However, we have made the last 7 layers trainable, compared to the last 5 in our first model.

2. **Changing the architecture of connected layers:** we use linear layers for this model. However, we only use two layers in this model compared to four in our first model.
3. **Changing the features calculated and concatenated:** our second model concatenates only four of the seven features discussed in the previous section. Namely, the squared difference of individual embeddings, the squared element wise difference, the element wise product and the sum of embeddings.

These tweaks in our model architecture enabled us to train a model that retains its effectiveness through transfer learning and feature engineering, but provides a non-correlated output compared to our first model that makes it a viable candidate for transfer learning.

4.2.2 Model Training

This model was trained with the same training loop described in the previous section. The figure below shows the training and validation loss and accuracy of the model over its training cycle of 100 epochs. Again, we can see that there is a steady increase in accuracy and decrease in loss, indicating that there was no overfitting over the duration of our training.

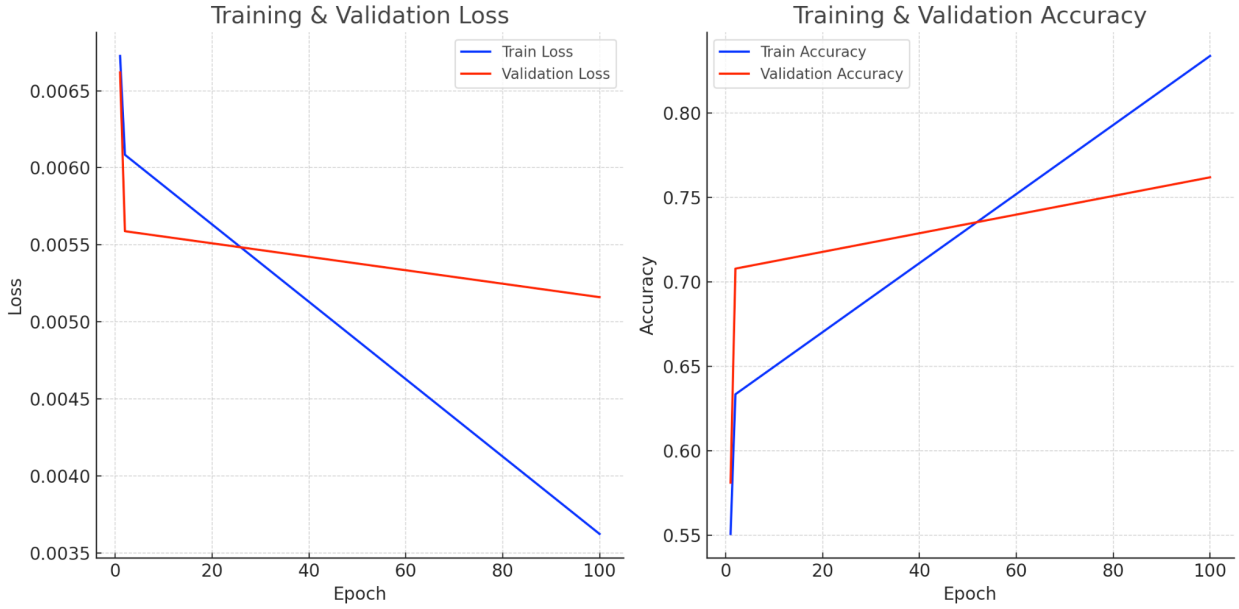


Figure 11a. **Figure 11b.**

Figure 11a. Representation of the Training and Validation Loss, Figure 11b Representation of Training and Validation Accuracy

4.3 Model 3 - Vision Transformer

In order to introduce further non-correlation into our data, we created a third model that leverages the capabilities of Vision Transformers (ViT), a transformer-based paradigm that was initially intended for sequential data processing. Transformers, which were created by Vaswani et al. in 2017, have shown impressive results in a number of applications, most notably natural

language processing. An important advancement in the field of facial feature analysis is the use of transformers, like ViT, for computer vision tasks like kinship recognition.

Vision transformers divide face photos into fixed-size patches for kinship detection, which enables the model to identify complex correlations and patterns between facial features. This flexibility is essential for tasks such as kinship recognition, where a model that can comprehend both local and global links is needed to identify faint familial ties based on visual features. ViT is particularly good at identifying long-range dependencies in the input data because of its self-attention processes. High-level features can be extracted by utilising pre-trained ViT models on large datasets, which offers a solid base for kinship detection similar to the encoding layer in the first and second model.

4.3.1 Model Architecture

Two similar neural networks share weights and parameters in a Siamese structure, which is used in the SiameseNet design. Here, the image encoder used by SiameseNet is a pre-trained ViT. Only the final layers of ViT are fine-tuned for kinship identification, with the majority of ViT parameters locked throughout training to preserve generic features and avoid overfitting. By utilising the knowledge encoded in pre-trained models, this strategic design enables efficient transfer learning and improves performance in kinship detection tasks.

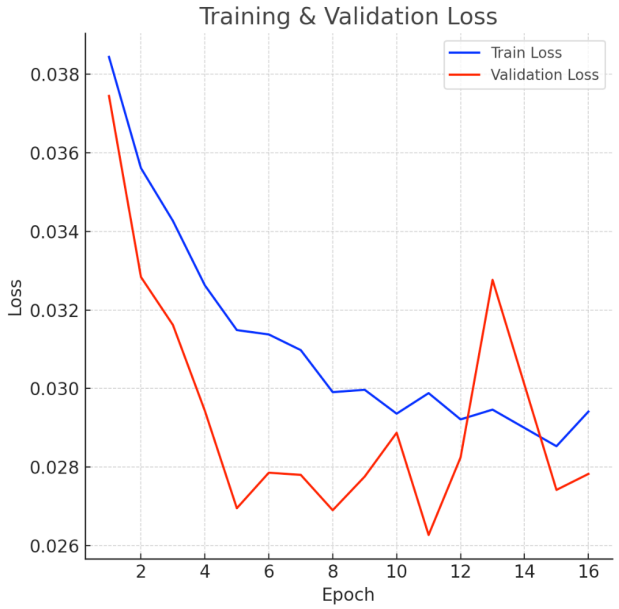
4.3.2 Logic and Methodology

Using the trained ViT model, features are extracted from input images in the SiameseNet procedure. Features that combine elements, such as addition and subtraction, can represent different facets of the interaction between faces. After that, a sequence of fully linked layers are applied to these combined features, resulting in a single output unit for kinship classification. By using a sophisticated representation of facial traits, this method enables the SiameseNet to identify delicate familial ties.

There are several layers in the SiameseNet design, and each has a specific function. Two face photos are fed through the ViT encoder to produce feature embeddings, which are then used as input embeddings. Feature combination layers use concatenation and element-wise operations to produce various relationship representations. The kinship categorization is subsequently determined by combining these combined features into a single output unit through the use of final fully connected layers. A diagrammatic representation of this model, visualised in Python, is available in the file Model3.pdf attached along with this report.

4.3.3. Training

SiameseNet is effectively trained by transferring knowledge from previously trained ViT models and making use of their capacity to capture general visual features. By keeping most ViT parameters fixed, parameter freezing helps to maintain general features while fine-tuning. With this approach, SiameseNet can take advantage of the wider knowledge encoded in pre-trained models while adapting to the particulars of kinship detection. This model was trained for 16 epochs, terminating due to early stopping. The figure below shows the accuracy and loss progress per epoch.



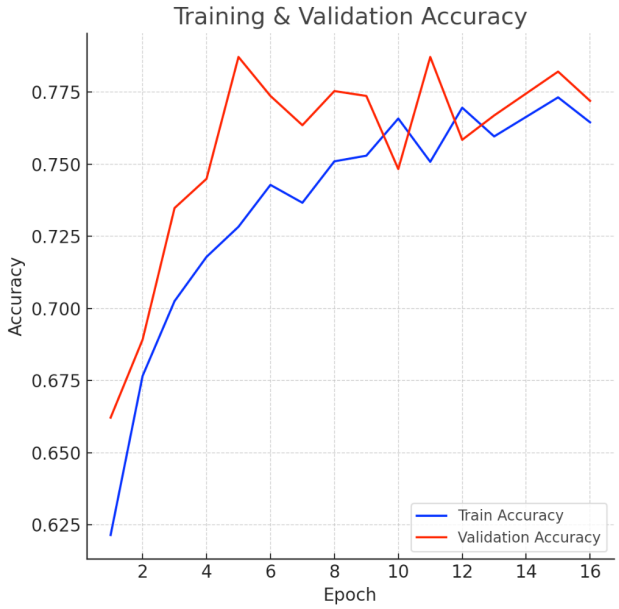


Figure 12a. **Figure 12b.**

Figure 12a. Representation of the Training and Validation Loss, Figure 12b Representation of Training and Validation Accuracy

4.3.4 Benefits of SiameseNet

For kinship detection, SiameseNet has a number of benefits. The model is able to extract complex facial features that are important in identifying family relationships. Transfer learning from pre-trained models improves performance, particularly in situations with a shortage of labelled kinship data. Furthermore, the SiameseNet architecture is applicable to real-world scenarios due to its resilience to changes in lighting conditions, poses, and facial expressions.

In conclusion, SiameseNet offers a promising kinship detection solution with the help of Vision Transformers.

5. Ensemble Learning

The intuition of ensemble learning is: for non-correlated classification models, different weights are learned, thus giving inconsistent results for a test record. If more than 2 models are available, ideally an odd number), this effect can be utilised to get better results. For example, given 3 models, the classification determined by the majority (at least ⅔) is taken as the final result. It is also possible to weight the models, or have a second layer of machine learning to obtain ideal results.

A simple voting may not give ideal results for the following reasons:

- Some models perform better than others, hence it is intuitive to give their results higher weightage
- To fully utilise the ensemble method, the architectures of the different models should be sufficiently different, so different weights are learned during training. In practice, however, architectures used may have similarities. Say there are 3 varied implementations of VGG and one siamese network. In such a scenario we do not want the results of the outvoted by the 3 models with lesser variation.

To address the above concerns, our solution uses weighted ensemble learning. A list of weights is passed to the code, and experiments with different weight combinations are performed to achieve optimal results. The weights provided for each model became hyperparameters that we looked to optimise.

Ideally, if Kaggle's leaderboard evaluation method was transparent, grid search could be used to find the best weight combinations for different model architectures. However, since submission files need to be manually uploaded to view the score, experiments were performed in a way that arrived at the best combination without having to examine the whole search space. The following table shows our experiments with weighted average and the resulting Kaggle score

#	**Siamese 1 (A)**	**Siamese 2 (B)**	**Vision Transformer (C)**	**Private Score**	**Public Score**	**Inference**
1	0.5	0.25	0.25	0.899	0.896	Higher weight to C is better than A or B
2	0.25	0.5	0.25	0.894	0.888	
3	0.25	0.25	0.5	0.908	0.896	
4	0.33	0.33	0.34	0.904	0.896	Lower score than #3
5	0.2	0.2	0.6	0.907	0.893	Further increasing C's weight does not improve score

6	0.3	0.3	0.4	0.906	0.897	Better than #3
7	**0.275**	**0.275**	**0.45**	**0.907**	**0.897**	**Best weights observed**

Table 2. Weighted Average Experiments

6. Conclusion

In conclusion, this report presents a novel approach to kinship detection by using negative sampling to improve the quality of the dataset and through Vision Transformer (ViT) models that improve the capabilities of traditional Siamese network architectures. The incorporation of negative sampling introduces a new dimension to the learning process, allowing the model to better differentiate between kinship and non-kinship relations. The use of Vision Transformers, a cutting-edge technology in image processing, further refines the model's ability to capture and analyse facial features crucial for kinship detection. Finally, the exploration of model architectures and feature engineering techniques offers new insights to the kinship detection challenge. With ensemble learning, benefits of different architectures are combined to boost scores. We put efforts to experiment with novel ideas in addition to learning foundational computer vision concepts.The submission is convincing due to the relatively high performance achieved, and use of methods like Siamese architecture which can be intuitively understood. There is still a room left for new techniques and explorations in the domain of kinship recognition.

7. References

Harwood, B., G, V. K. B., Carneiro, G., Reid, I., & Drummond, T. (2017). *Smart Mining for Deep Metric Learning* (arXiv:1704.01285). arXiv. http://arxiv.org/abs/1704.01285

Sahito, A., Frank, E., & Pfahringer, B. (2019). *Semi-Supervised Learning using Siamese Networks* (Vol. 11919, pp. 586–597). https://doi.org/10.1007/978-3-030-35288-2_47

Dey, S., Dutta, A., Toledo, J. I., Ghosh, S. K., Lladós, J., & Pal, U. (2017). Signet: Convolutional siamese network for writer independent offline signature verification. arXiv preprint arXiv:1707.02131.

Tian, M., Teng, G., & Bao, Y. (2022). Kaggle Kinship Recognition Challenge: Introduction of Convolution-Free Model to boost conventional. arXiv preprint arXiv:2206.05488.

Zhu, X., Li, Y., Li, D., Dong, L., & Chen, X. (2022, December). Lightweight Transformer Network and Self-supervised Task for Kinship Verification. In 2022 IEEE 8th International Conference on Computer and Communications (ICCC) (pp. 1506-1511). IEEE.

Tang, H. (2023). KINSHIP VERIFICATION USING VISION TRANSFORMERS (Master's thesis).

Serengil, S.I. and Ozpinar, A. (2021) 'Hyperextended Lightface: A facial attribute analysis framework', 2021 International Conference on Engineering and Emerging Technologies (ICEET) [Preprint]. doi:10.1109/iceet53442.2021.9659697.

Benhur, S. (2022) A friendly introduction to Siamese networks, Built In. Available at: https://builtin.com/machine-learning/siamese-network (Accessed: 24 November 2023).

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering. arXiv:1503.03832. https://arxiv.org/abs/1503.03832

Cao, Q., Shen, L., Xie, W., Parkhi, O. M., & Zisserman, A. (2018). VGGFace2: A dataset for recognising faces across pose and age. In Proceedings of the International Conference on Automatic Face and Gesture Recognition.

Yi, D., Lei, Z., Liao, S., & Li, S. Z. (2014). CASIA-WebFace: Learning Face Representation from Scratch. arXiv:1411.7923. https://arxiv.org/abs/1411.7923

Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. IEEE Signal Processing Letters, 23(10), 1499-1503. https://doi.org/10.1109/LSP.2016.2603342