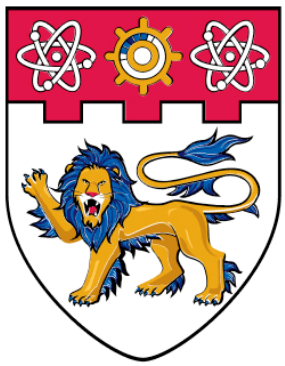

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4041 Machine Learning

AY 2022/2023 Semester 1

Project

Northeastern SMILE Lab - Recognizing Faces in the Wild

Charmaine Chng	N2202555E
Lam Hon Wei	U2040382A
Chan Yi Ru Micole	N2202554H
Ho Kae Boon	N2202559D

Member Contribution

Member	Contribution
Charmaine Chng	Problem Statement, Data Pre-Processing, Convolutional Neural Networks (ResNet50, InceptionV3, VGGFace)
Lam Hon Wei	Exploratory Data Analysis, Data Pre-Processing, Random Forests, Video Presentation
Chan Yi Ru Micole	Data Pre-Processing, K-Means Clustering, Challenges, Conclusion, Presentation slides and script, Video Presentation
Ho Kae Boon	Data Pre-Processing, Random Forests, Support Vector Machines, Video Presentation

1 Problem Statement

It is common and biological knowledge that blood relatives often share facial features. The background given for the chosen Kaggle competition is that researchers from Northeastern University are looking to improve their algorithm for facial image classification, with the hopes of bridging the gap between research and other familial markers like DNA results.

Our objective of this report is to introduce and evaluate the deep learning and machine learning methods that were used to determine if two people are blood related (based solely on images).

2 Exploratory Data Analysis (EDA)

EDA was carried out to better understand the images and the structure of the file directories. The main files that were provided for the competition, as well as their descriptions, are in Table 1.

File	Description
train-faces.zip	The training set is divided into Families (FXXX), and then into individuals (MIDX). Images in the same MIDX folder belong to the same person. Images in the same FXXX folder belong to the same family.

train_relationships.csv	This csv file shows the relationships between two individuals.
	A snippet of the csv file is as follows.
	`p1 p2`
	`0 F0002/MID1 F0002/MID3`
	`1 F0002/MID2 F0002/MID3`
	`2 F0005/MID1 F0005/MID2`
	`3 F0005/MID3 F0005/MID2`
	`4 F0009/MID1 F0009/MID4`

test-faces.zip	This test set contains the face images of unknown individuals and will be used to evaluate our final model.
sample_submission.csv	This csv file is a sample submission file in the correct format. The column img_pair describes the pair of images. The goal for the project is to predict if each pair of images in test-faces are related or not, where 1 means related and 0 means unrelated.

Table 1: Description of Files Provided

Upon submission to Kaggle, our results will be evaluated based on the area under the ROC curve between the predicted probability and the observed target. It is to be noted that not all pairs of images will be scored.

2.1 Overview

The training set consists of a total of **12379** images, with **3598** relationship combinations present. Moreover, we have also analyzed that there are a total of **470** families and **2318** unique individuals in the training set. We can observe from Figure 1 that the pictures of an individual in the training set are of various expressions and taken from different angles.

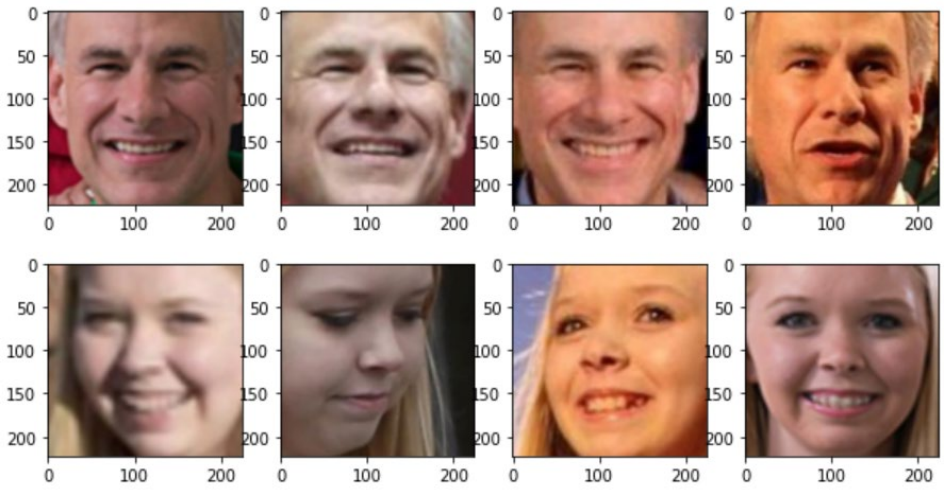

Figure 1: Samples from the Training Set

In addition, we observed that the images of individuals wearing accessories, such as sunglasses or hats, are also included in the training dataset. Photos taken at different life stages are also included. We hypothesize that these variables could potentially hinder the results of our model.

2.2 Data Distribution

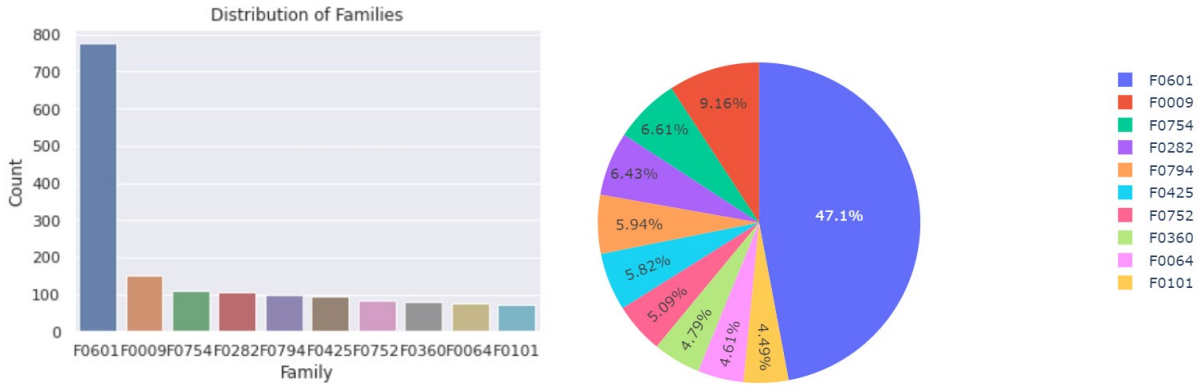

Figure 2: Distribution of Families

The bar graph and pie chart in Figure 2 demonstrates the distribution of families in the training set. For example, F0601 represents the folder name 601 for the same family photos. Due to the large number of families in our training sets, we choose to analyze the distribution of the largest 10 families. As shown above, it is obvious that F0601 Family Folder has the greatest number of images in the Kaggle dataset, followed by F0009 and F0754 so on and so forth as shown by the legend on the right where it is arranged according to descending order. This imbalance in families could potentially affect the accuracy of the model if the features it picks up and learns from are largely representative of FM601.

2.3 Distribution of Images of each Individual

We explore further into images of each family provided by the training set. We observe in Section 2.2 that FM601 makes up the largest proportion of images within the training set. Further investigation on FM601 shows that it is the British Royal Family.

With reference to Figures 3a and 3b, we can observe that individuals from FM601 have the highest number of photographs. Out of which, Prince William has the highest number of images at 95, followed by Queen Elizabeth II which tallies at 82, and so on and so forth.

Kin No:	
F0601/MID6	95
F0601/MID2	82
F0601/MID5	65
F0601/MID19	63
F0601/MID7	48
F0601/MID20	43
F0009/MID1	41
F0601/MID12	40
F0794/MID1	37
F0601/MID11	33

Name: uniqueId, dtype: int64

Figure 3a: Number of Photographs

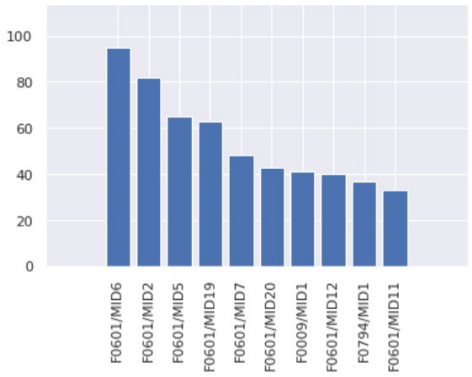

Figure 3b: Chart of Number of Photographs

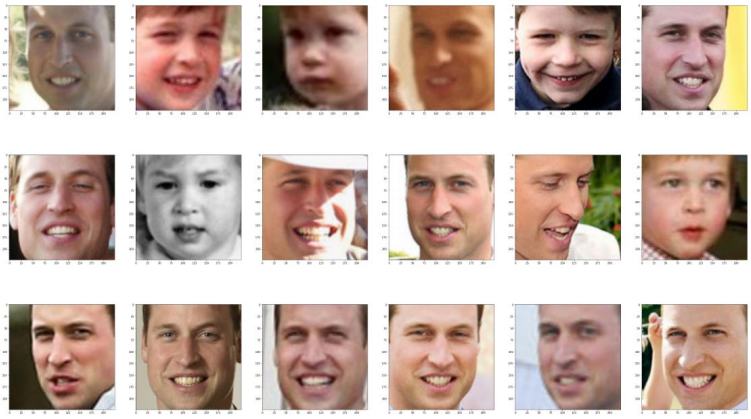

Figure 4: Images of Prince William

2.4 Dimensionality Reduction

Dimensionality reduction is used to transform data from high dimensional spaces into low dimensional spaces, with minimal loss of information, in order to provide similar information from the original data. It is used in reducing the number of input features due to the large volume of datasets. With such reduction in datasets, the space required to store the datasets are significantly reduced. Some examples of **Dimensionality Reduction** include **Principal Component Analysis (PCA)** where it maps m-dimensional input features to k-dimensional latent features, and **Singular Decomposition Values (SDV)** extracts data in the directions with the highest variances respectively.

2.4.1 Principal Component Analysis (PCA)

Given that FM601 makes up the largest proportion of images within the training set, it is necessary to use PCA to estimate the number of components to describe large datasets. This can be performed by finding out the relationship of the explained variance ratio against the number of components. It is vital to import ***PCA*** from ***sklearn.decomposition*** to set the number of components to explain the variation to the train dataset.

Figure 5: Eigenfaces of Prince William (3 x 8 array)

The results are fascinating as it tells us how the images vary. For instance, the first eigenface (from the top left) seems to be associated with a shadow, and images beside it along the first row show some angles in the lighting that seem to be picking out certain features, such as eyes, noses, and lips. On top of that, the last two rows of images show some contrast in shadow and lighting with similar facial characteristics.

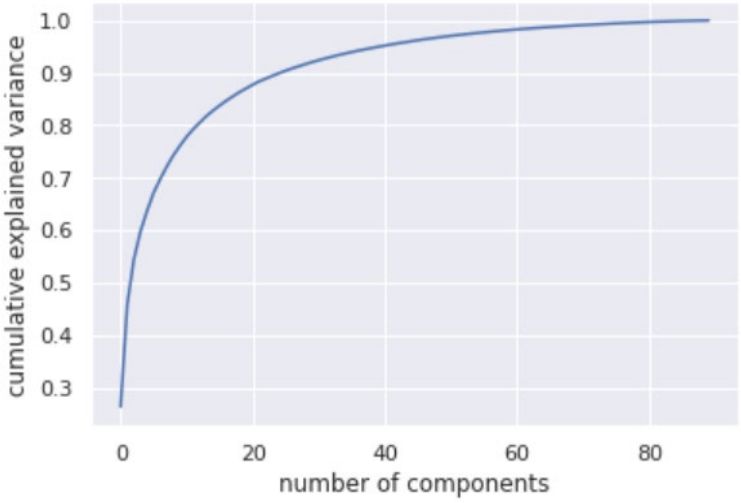

Figure 6: Graph showing the number of components along with explained variance

We see that close to 70 different images account for just over 100% of the variance. That would mean that 70/95 images can recover most of the essential characteristics of the data. With that, it is vital to use PCA to reduce the number of components from 70 to at least 22 to approximately 90% training set, which can be used in Random Forests and Support Vector Machine to make predictions in data.

2.4.2 Singular Value Decomposition (SVD)

Singular Value Decomposition is used to decompose the matrix into 3 sub-matrices namely **U, S, V** where U is the left eigenvector, S is a diagonal matrix of singular values and V is called the right eigenvector. It is used in unsupervised Machine Learning to capture the image and use part of them to reconstruct the image. In order to perform it, the one-dimensional numpy array needs to be converted into a 2-dimensional numpy matrix where the image size of the F0601/MID6 folder of the train data set is **50176.**

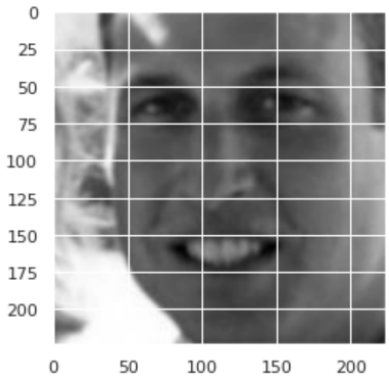

Figure 7a: Original Image size in F0601/MID6

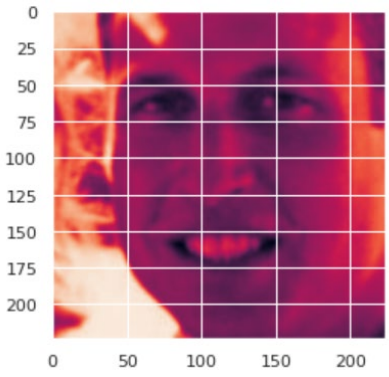

Figure 7b: R band image for SVD

```
matrix([[235, 236, 237, ..., 107, 111, 115],
        [234, 235, 237, ..., 108, 112, 115],
        [233, 234, 235, ..., 109, 113, 116],
        ...,
        [154, 164, 178, ...,  73,  77,  80],
        [163, 174, 189, ...,  72,  79,  84],
        [170, 180, 196, ...,  72,  80,  86]], dtype=uint8)
```

Table 2: 2d numpy matrix

After performing the necessary changes in the R band image for the SVD, it is crucial to find out the dimensions of U and V matrices. After executing the code for **U.shape** and **V.shape**, U and V turns out to be square matrices and their dimensions match the image size which is **224** by **224**. The corresponding eigenvalues of s are simply ordered in a descending form of a vector as shown in the table below.

```
array([1.72990312e+02, 1.06144674e+02, 5.52084668e+01, 3.76908163e+01,
       3.43302671e+01, 2.75164688e+01, 2.52623254e+01, 2.17131068e+01,
       1.76012557e+01, 1.54559519e+01, 1.27761831e+01, 1.13836387e+01,
       1.05409102e+01, 9.48437918e+00, 8.87089021e+00, 7.22593170e+00,
       6.85737239e+00, 6.02185089e+00, 5.39441997e+00, 5.32955998e+00,
       4.83710065e+00, 4.39677205e+00, 4.12917636e+00, 3.63447817e+00,
```

Table 3: s vector

The eigenvalues from SVD can compute the amount of variances explained by each singular vector. The first singular vector or principal component explains most of the variation in the image. For instance, it explains 60% of the total variation and the second one explains close to 22.5% of the variation.

```
array([0.596, 0.225, 0.061, 0.028, 0.023, 0.015, 0.013, 0.009, 0.006,
       0.005, 0.003, 0.003, 0.002, 0.002, 0.002, 0.001, 0.001, 0.001,
       0.001, 0.001])
```

Table 4: Variance explained by Top 20 Singular Values

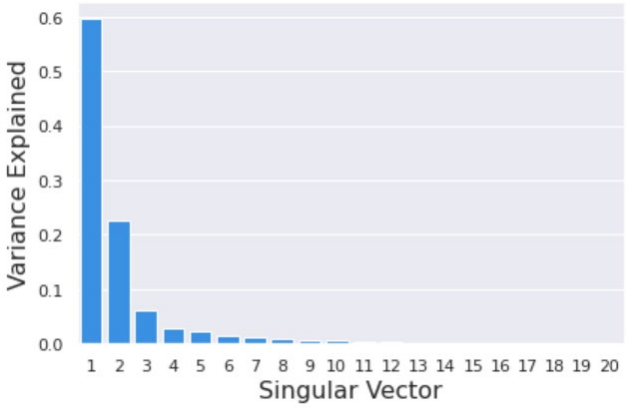

Figure 8: Variances explained from bar plot captures variation in the image of each vector

The Top k singular vectors capture most of the variations and to see how it compares it with the highest k component of the images. Figure 9 shows the reconstruction of the Top K singular vectors.

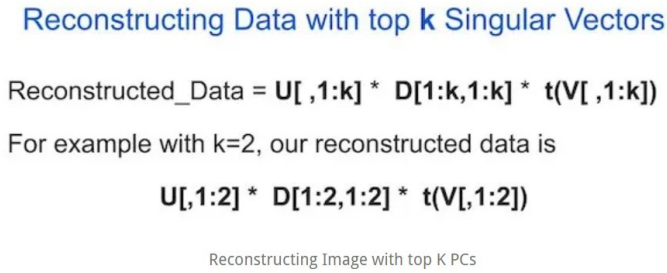

Figure 9: Data Reconstruction with top k Singular Vectors

In this example, we used 1, 3, 10 and 20 singular vectors to reconstruct the matrix using matrix multiplication as shown above. As shown in the images below, it turns out that the top 3 components are not enough to reconstruct the images as they are blurred. However, adding more singular vectors can help to improve the quality as shown at the bottom. This suggests that the images turn sharper as more singular vectors are added to reduce the variation explained.

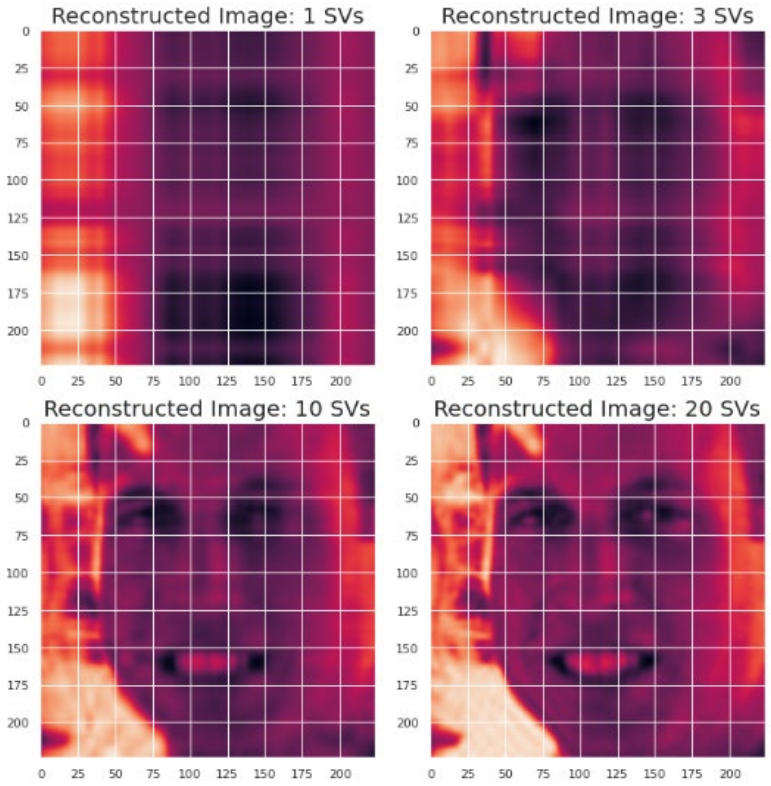

Figure 10: Reconstructed Images from SVD with different SVs

3 Data Pre-Processing

Before working on the models, it is required that we split the data provided in the training set into training and validation sets. The purpose of the training set would be for the model to train and "learn" from. Thereafter, the derived model will be applied to the validation set which is used for hyperparameter tuning and tweaking the model. This is to ensure that the model is generalisable to out-of-sample data so as to prevent instances of overfitting.

In this project, we split the given train dataset into approximately 90% training set, and 10% validation set based on train_relationships.csv. We are not required to account for the test set in this case as we will be utilizing the test set provided by the Kaggle competition.

Besides splitting the images into training and validation sets, further preprocessing was also performed to ensure that the images are suitable inputs. Certain preprocessing steps, such as image augmentation, also prevents overfitting by having the model learn from "varied" images.

Preprocessing Step	Model(s)
Resizing & Combining Images	Support Vector Machine, Random Forests, K-Means Clustering, Neural Networks
Flattening of Images	Support Vector Machine, Random Forests, K-Means Clustering
Image Normalization	Support Vector Machine, Random Forests, K-Means Clustering, Neural Networks

Table 5: Preprocessing Steps and the Respective Models they are Applied to

After splitting the data into the training and validation sets, we noticed that not all individuals found in train_relationships.csv are found in the provided train dataset. Hence, such instances are removed and we have a resultant of **3362** relationships in total - 3066 relationships to the training set and 296 to the validation.

3.1 Resizing & Combining Images

```
pd.Series(image_size).unique()

array([(224, 224)], dtype=object)
```

Figure 11: Dimensions of Images

Resizing of images is an essential part of preprocessing as machine learning models tend to train faster on images that have smaller pixel dimensions. Upon analyzing the full train dataset (the original dataset before splitting into training and validation sets) that was provided, we found that all images have the same dimensions of 224 x 224 pixels as shown in Figure 5. In theory, there is no need for resizing for the images.

For our implementation of Support Vector Machine and Random Forest, in order to save computation power and decrease the number of features, we resize the images to 100 x 100 pixels. To ensure consistency, we also convert all color images into grayscale images. These would change the dimensions of each picture from (224, 224, 3) to (100, 100). Since there might be multiple images for a single person, we also merge all images of the same individual by averaging each pixel's value.

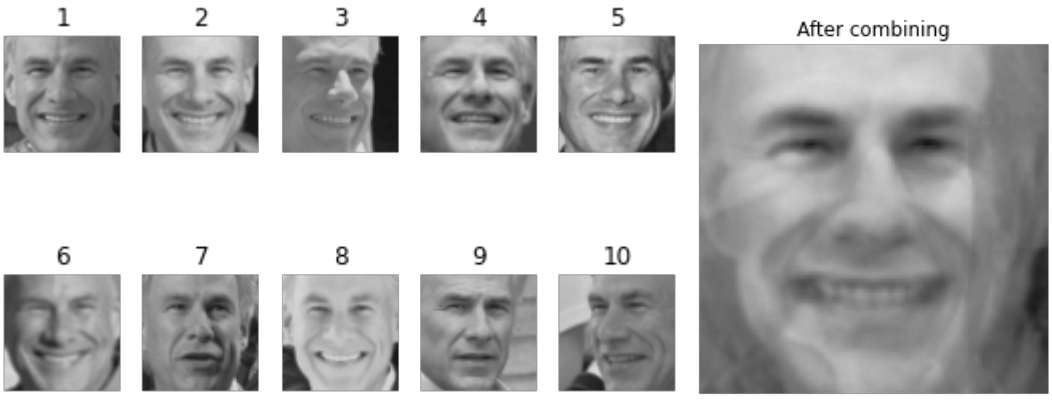

Figure 12: Combining F002/MID1

3.2 Flattening of Images

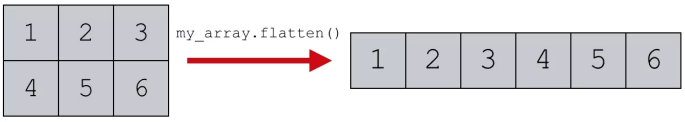

Figure 13: Image Flattening from 2D to 1D array

One aspect of Data Pre-Processing is flattening of images from a 2 Dimensional-Array into a 1 Dimensional Array. Since all our images have the same dimensions which are 224 x 224 pixels, we have to flatten it into (224)^2 = 50,176 pixels. The data cannot be read multi-dimensionally, so image flattening is performed to convert all images using *img.flatten()*.

Furthermore, Image flattening is used in our models such as Support Vector Machine and Random Forests. In these two models, our features value will be the difference between each pixel of two individuals. Hence by flattening our images, we can easily compute their value differences and treat each index, that is the pixel difference value, as the features directly.

3.3 Image Normalization

Image normalization can ensure that the models reach convergence in a shorter duration as the distribution of the data is similar.

For Support Vector Machine and Random Forests, we carry out normalisation by dividing each pixel value with 255 before computing the differences. This would help us minimise the variance of each feature hence allowing quicker computation.

As for convolutional neural networks, we can take a look at image normalization below.

	Input size	Data format	mode
Xception	299x299	channels_last	tf
VGG16	224x224	channels_first / channels_last	caffe
VGG19	224x224	channels_first / channels_last	caffe
ResNet50	224x224	channels_first / channels_last	caffe
InceptionV3	299x299	channels_first / channels_last	tf
InceptionResNetV2	299x299	channels_first / channels_last	tf
MobileNet	224x224	channels_last	tf
DenseNet	224x224	channels_first / channels_last	torch
NASNet	331x331 / 224x224	channels_first / channels_last	tf
MobileNetV2	224x224	channels_last	tf

Table 6: Keras Preprocess Implementations

With reference to Table 6, calling the "prepocess_input" function on the images will scale and normalize the pixels or color channels accordingly, so that it is compatible with the convolutional neural network models that are provided with keras. Further exploration of the source codes show that the function accepts an array and performs the transformation as shown in Table 7, and is dependent on the mode feature in Table 6.

Mode	Normalization
tf	Scales pixels between -1 and 1 sample-wise.
caffe	First, it converts the images from RGB to BGR. Then it zero-centers each color channel with respect to the ImageNet dataset without scaling.
torch	Scales pixels between 0 and 1 and then normalizes each channel with respect to the ImageNet dataset.

Table 7: Normalization based on Mode

4 Introduction and Evaluation of Models

This section aims to review the 6 models that were trained and evaluated - K-Means clustering, Support Vector Machine, Random Forest, and 3 pre-trained convolutional neural network models. An overview of these methods is also included under their respective subsections.

4.1 K-Means Clustering

K-means Clustering is an unsupervised learning algorithm that is used to group an unlabeled dataset into k different non-overlapping clusters based on each point's proximity to the center of the k groups.

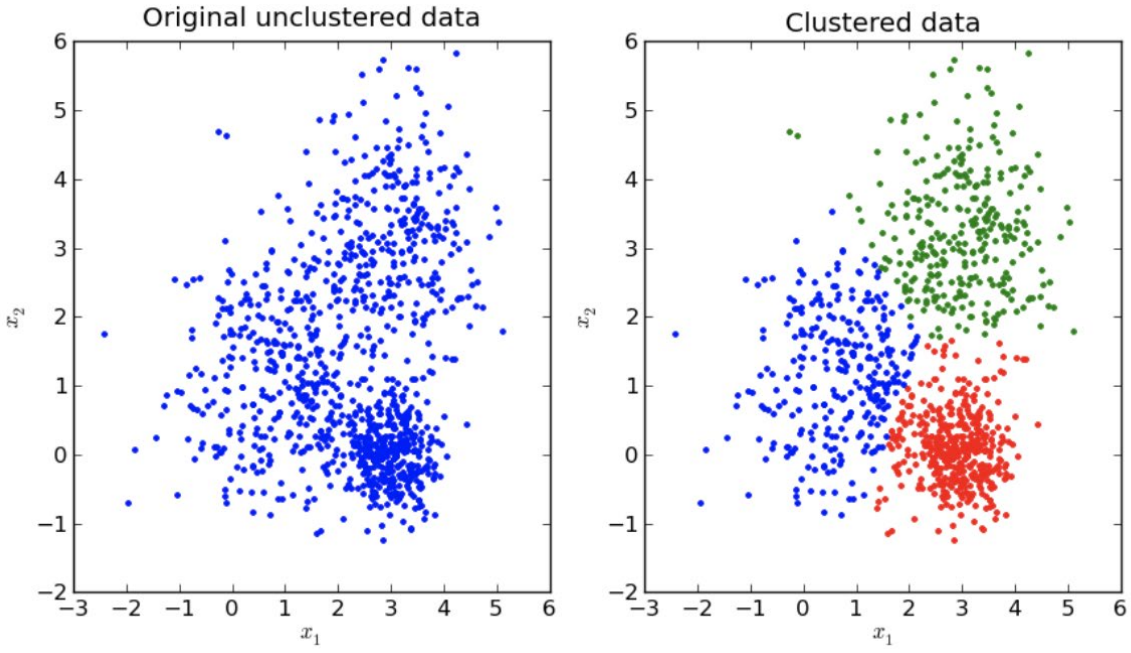

Figure 14: K-Means Clustering

It is a partitional clustering approach that associates each cluster with a centroid. The algorithm works by iterating the process of assigning each point to a cluster with the closest centroid then recomputing the centroid, until the centroids are no longer changed.

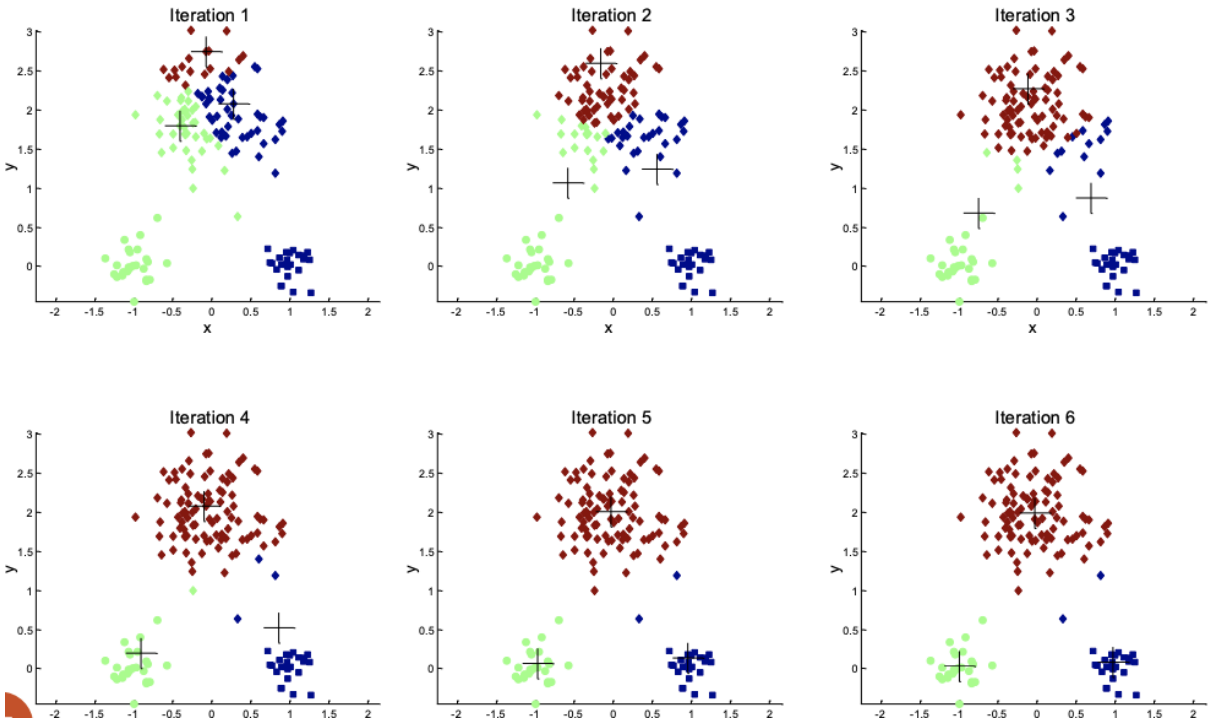

Figure 15: Process of K-Means Clustering (taken from Lecture 10)

Step	Description of K-Means Clustering Algorithm
1	Choose the number of clusters, k
2	Select k random points to be centroids of each cluster (as denoted by the symbo Figure 13)
3	Compute distance of each data point (x_i, y_i) to each centroid
4	Assign each data point to the centroid it is closest to. Data set is now split into k clusters.
5	Find the centroid of each of these k clusters. These will be the new centroids.
6	Repeat steps 3-5 until the centroids do not change.

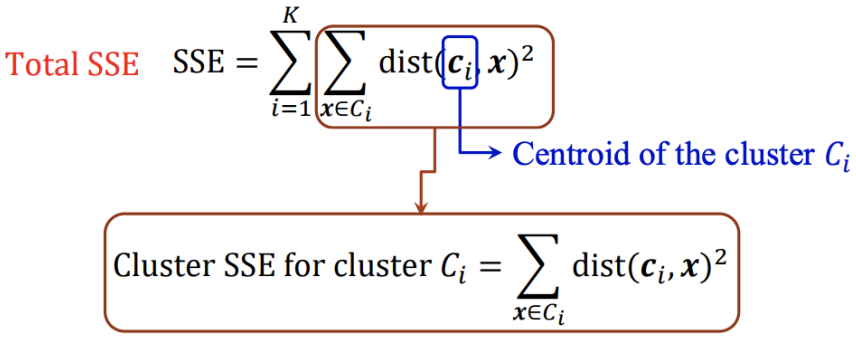

Figure 16: Sum of Squared Error (SSE) Formula

We aim to minimize the total Sum of Squared Error. For each data instance, the "error" is the distance to the centroid.

For our dataset, we used the elbow method to identify an optimal k (in our case k = 20).

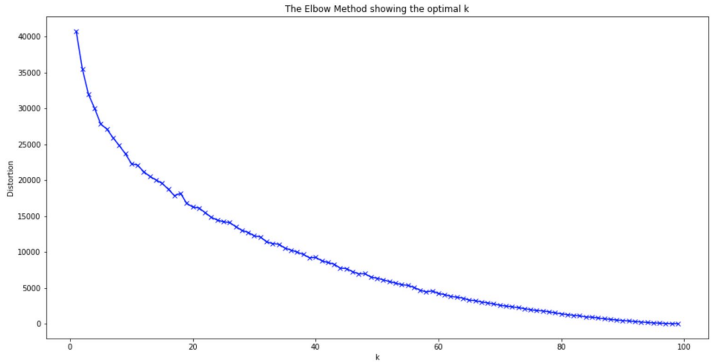

Figure 17: Elbow Method to determine optimal "k"

Motivation

K-Means Clustering as it is a well known distance-based algorithm that guarantees convergence, is scalable to large data sets and widely used in image segmentation. Its simplicity and high speed makes it a good model for our group to start off with, to get an idea of the structure of this large data set.

Results:

```
Accuracy: 0.324
```

4.2 Support Vector Machine (SVM)

Overview

Support Vector Machine (SVM), a supervised machine learning is one of the most useful techniques in Face Recognition problems.

The goal of SVMs is to learn an optimal hyperplane in an N-dimensional where N is the number of features that distinctly classify the data points. After we plot the data points accordingly, we classify the data points by finding a hyperplane that separates the different classes.

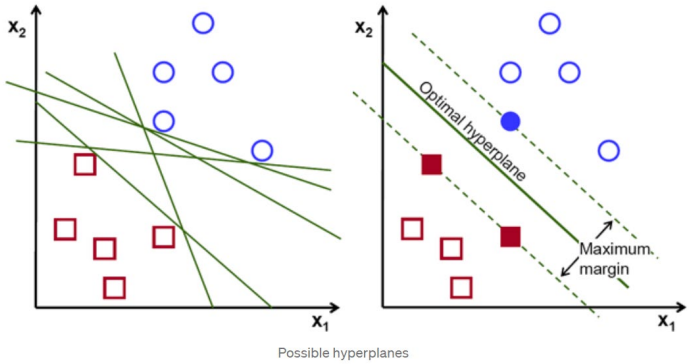

Figure 18: Finding the optimal hyperplane using Support Vector Machine

As seen in the left half of Figure 18, there can be multiple hyperplanes that can separate the training examples perfectly. The algorithm chooses the hyperplane with the maximum margin, as shown in the right half of Figure 18. This is based on the assumption that larger margins are more likely to reduce generalization errors. Thus, the hyperplane with the maximum margin will reduce overfitting and ensure that the model is generalised and more tolerant to errors.

Motivation

SVM is a powerful model for complex data sets with high dimensional spaces. It is also memory efficient as compared to the other models.

The following scores were retrieved after uploading the CSV generated for the test set onto Kaggle.

Private Kaggle Score: 0.460

Public Kaggle Score: 0.470

4.3 Random Forests

Overview

Random Forests is a supervised machine learning that is widely used in Classification and Regression Model. It is a class of ensemble methods designed specifically for decision tree classifiers. The Random Forests algorithm grows "*t*" many trees and uses bagging, known as Bootstrapping, to fit multiple models on different subsets of training datasets which is required for the predictions from all models. Moreover, Bagging helps to improve the accuracy and efficiency of Machine Learning Algorithms.

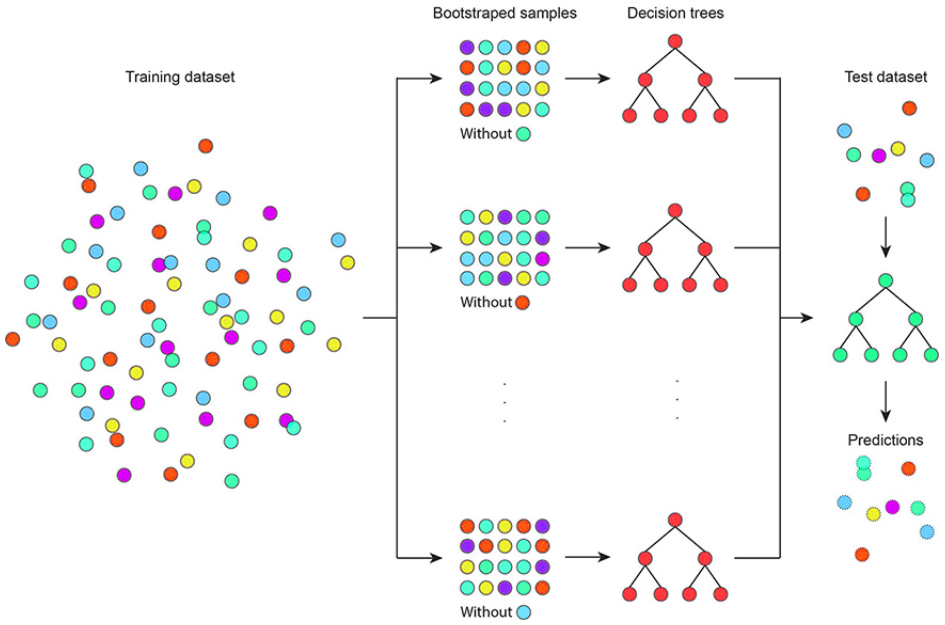

Figure 19: Random Forests

The Random Forest Algorithm generates the outcomes based on the predictions of the decision trees. The final classification is based on the majority votes among the various trees in the forests to choose the c classification result with the most votes. Growing (increasing the number of trees) increases the accuracy of the results which also ensures a better voting system.

Motivation

This algorithm is far more forgiving than other algorithms for overfitting and handling outliers. Furthermore. Ensemble Methods combine many classifiers to provide solutions to complex problems. This improves model performance in terms of accuracy by aggregating the predictions of multiple models.

In our case, Random Forests seems to perform better than Support Vector Machine.

The following scores were retrieved after uploading the CSV generated for the test set onto Kaggle.

Private Kaggle Score: 0.519

Public Kaggle Score: 0.471

4.4 Convolutional Neural Network

Convolutional neural networks (CNN) are commonly used to process and classify images due to their ability to extract features from high dimensional image data. With the goal of the SMILE prediction competition being to determine the similarity of 2 images and determining if they share kinship, CNN is hypothesized to be useful and possibly provide the most accurate results.

To ensure robustness of our results, we have selected the following 3 available models from the Keras library for our research. ResNet50 and InceptionV3 were chosen due to their contrast in architecture designs, where the former is a deep model and the latter is a wide model. VGGFace is chosen due to it being pre-trained on faces, which should ideally be useful for this research.

Model	Brief Description
ResNet50	Deep CNN model that has been pre-trained with ImageNet. ImageNet is an image database that consists of more than 14 million images, consisting of 1000 categories. However, "humans" or "faces" are not part of the 1000 categories that the model is trained on.
InceptionV3	Wide CNN model that has been pre-trained with ImageNet.
VGGFace	Deep CNN model that has been developed for face recognition and is pre-trained using the VGGFace database that consists of over 3 million images of 9131 individuals.

Table 8: Models Used and their Description

However, the biggest limitation faced would be that training CNNs are computationally expensive and the team lacks the necessary computational resources required to execute the training procedures and find a best fitted model. Specifically, 25GB of RAM will be exhausted when more than 20 epochs are used to train a single model. Taking into consideration the limited computational resources available, Table 9 outlines the parameters used to fit the respective CNN models for this research.

Parameter	Value
Batch Size	16
Epochs	20
Steps Per Epoch	50
Validation Steps	50

Table 9: Parameters for Model Fitting

In an ideal scenario where computational resources are more easily accessible, parameter values set in Table 9 would be increased to find a convergence point via grid search cross validation.

Overfitting of the model will be analyzed through plotting accuracy and loss charts that compares between the training and validation sets. A divergence of the plots would suggest that there is overfitting and hence, a need to retune the parameters to better fit out-of-sample data.

In addition to purely implementing the CNN models, additional steps are to be taken to ensure that we are able to combine and contrast the 2 input images and determine their similarity. Additional layers as outlined in Figure 20 were added to the CNN architecture.

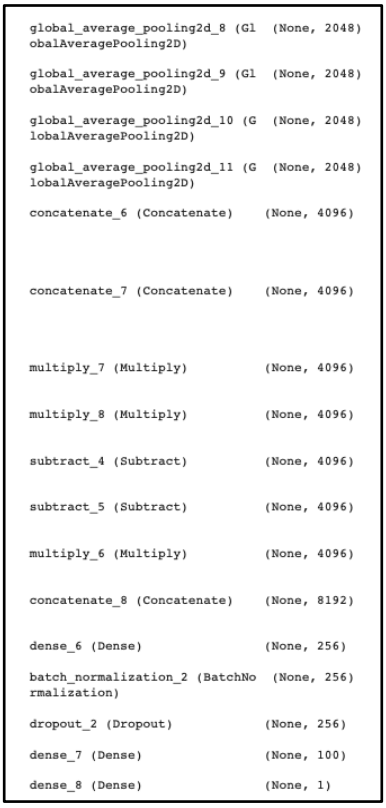

Figure 20: Layers Proceeding the Keras Model

A description of the layers and blocks in Figure 20 are outlined in Table 10 (Peltarion, n.d.).

Layer	Description
global_average_pooling_2d	The 2D Global average pooling block takes a tensor of size 224 x 224 x 3 and computes the average value of all values across the entire 224 x 224 matrix for each of the 3 input channels. It is used for all image instances.
concatenate	Concatenation is used to gather outputs from the first and second image in the image pair input. This is to retrieve a single output that will be fed into a series of dense layers and eventually the output layer that will provide the probability of kinship between the 2 images.

multiply	The multiply block takes in 2 input tensors, and returns a single tensor containing the element-wise multiplication over the inputs. We multiply 2 tensors together to find similar features between the tensors.
subtract	Subtract is used after the tensors are squared together with the multiply block. Hence, $(x-y)^2$ with x and y representing the 2 tensors, is used for equality relationships, and intuitively shows how close the 2 tensors are.
dense	A dense layer is a fully connected layer of artificial units and can represent the input, output, or hidden layers of a CNN model.
batch_normalization	Standardizes the inputs to a layer for each mini-batch and stabilizes the learning process, hence dramatically reducing the number of training epochs required to train the CNN.
dropout	Regularizes the CNN model by ignoring the specified proportion of neurons (0.01 in the research conducted) during the training phase at random.

Table 10: Description of Layers/Blocks Used

4.4.1 ResNet50

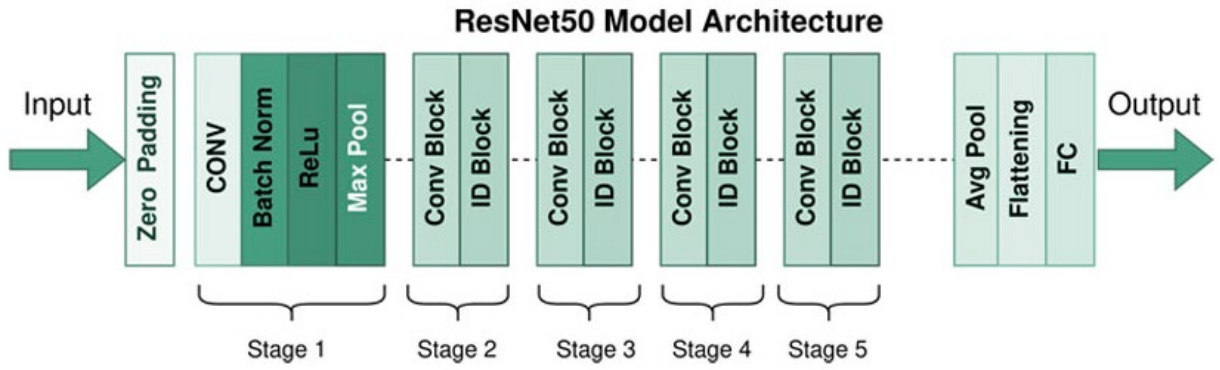

Figure 21: ResNet50 Model Architecture (eduCBA, n.d.)

ResNet50 falls under the ResNet family of models and consists of 48 convolution layers, 1 MaxPool layer that identifies the most prominent features of the layer, as well as an Average Pool layer that creates a downsampled feature map. It is also considered to be a deep model, as opposed to a wide one, as the number of layers it has is an important property.

The 50-layer ResNet uses a bottleneck design for the building block. A bottleneck residual block uses 1×1 convolutions, known as a "bottleneck", which reduces the number of parameters and matrix multiplications. This enables much faster training of each layer (DataGen, n.d.).

Parameter	Value

Activation Function (Hidden Layers)	ReLu
Activation Function (Output Layer)	Sigmoid
Loss Function	Binary Crossentropy
Learning Rate	0.00001
Optimizer	Adam

Table 11: ResNet50 Parameters

Table 11 outlines the parameters used for building the model. ReLu was used as the activation function for the hidden layers because it is able to overcome the vanishing gradient problem as well as being more computationally efficient. Sigmoid was chosen as the activation function in the output layer, and binary cross entropy as the loss function because we are predicting whether a pair shares kinship, and it is hence a binary problem. Lastly, Adam is utilized due to its computational efficiency which is appreciated when working with large datasets as it requires less memory to find an optimal gradient.

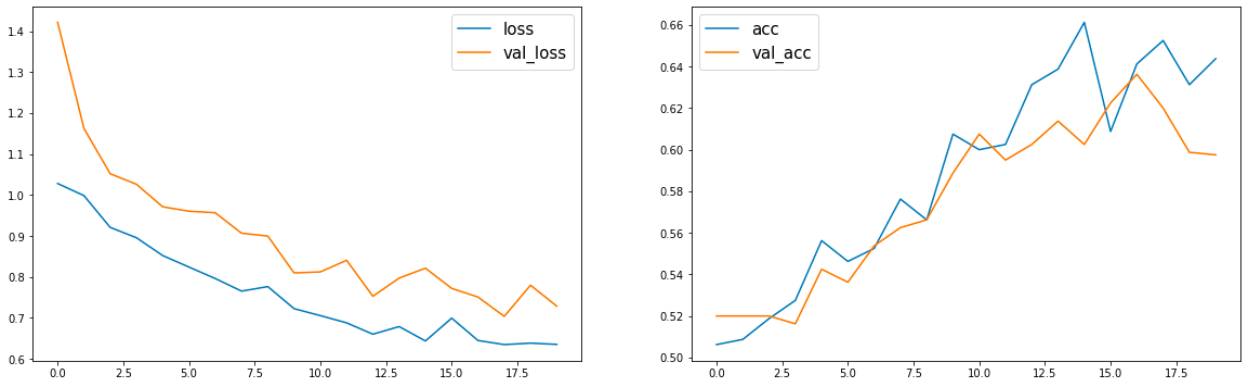

Figure 22: Loss and Accuracy Plots for ResNet50

After fitting the ResNet50 model to the training data, the plots shown in Figure 22 were generated. We can identify that there is no risk of overfitting at this stage as there is no deviance in the training and validation plots for both loss and accuracy. In fact, as mentioned in Section 4.4, limited computational resources have resulted in early stoppage in training, and there is still potential for the model to be further trained.

The following scores were retrieved after uploading the CSV generated for the test set onto Kaggle.

Private Kaggle Score: 0.656

Public Kaggle Score: 0.655

4.4.2 InceptionV3

Unlike regular CNNs, Inception models utilize filters of 3 different sizes (1x1, 3x3 and 5x5) within the same system, which makes the model wider and thus more computationally efficient. It is a wide model as compared to ResNet50, which is a deep model. Tapping on the idea behind Naive Inception, the InceptionV1 is made more computationally efficient by adding a 1x1 convolution before the 3x3 and 5x5 convolutions to limit the number of inputs into these convolutions. This concept is known as the InceptionV1 model.

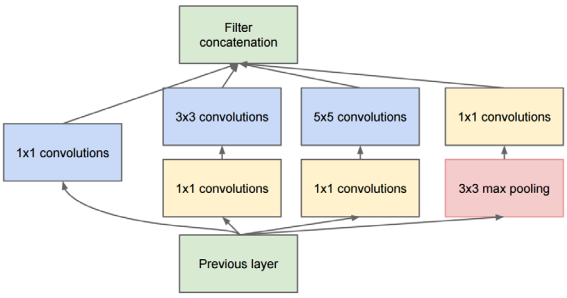

Figure 23 - InceptionV1 Architecture

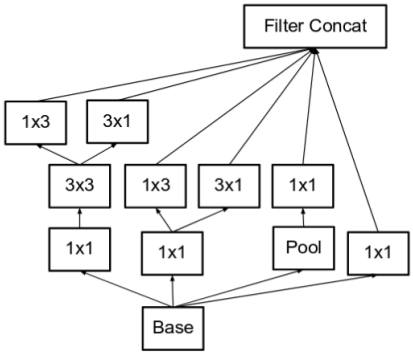

Figure 24 - InceptionV2 and InceptionV3 Architecture

We chose to use InceptionV3 as it has been described to be a more computationally efficient and commonly used module. As shown in Figure 24, the larger convolutions are broken down into smaller ones which help to further improve computational efficiency and tackles the above-mentioned issues of regular CNNs to a greater extent. InceptionV3 includes label smoothing (which helps to prevent overfitting) and additional optimizers.

Parameter	Value
Activation Function (Hidden Layers)	ReLu
Activation Function (Output Layer)	Sigmoid
Loss Function	Binary Crossentropy
Learning Rate	0.00001
Optimizer	Adam

Table 12: InceptionV3 Parameters

With reference to Table 12, we can observe that the parameter values are kept similar to that of the ResNet50 model. This is because we are making a comparison between a deep and wide model and would like to hold all other variables constant.

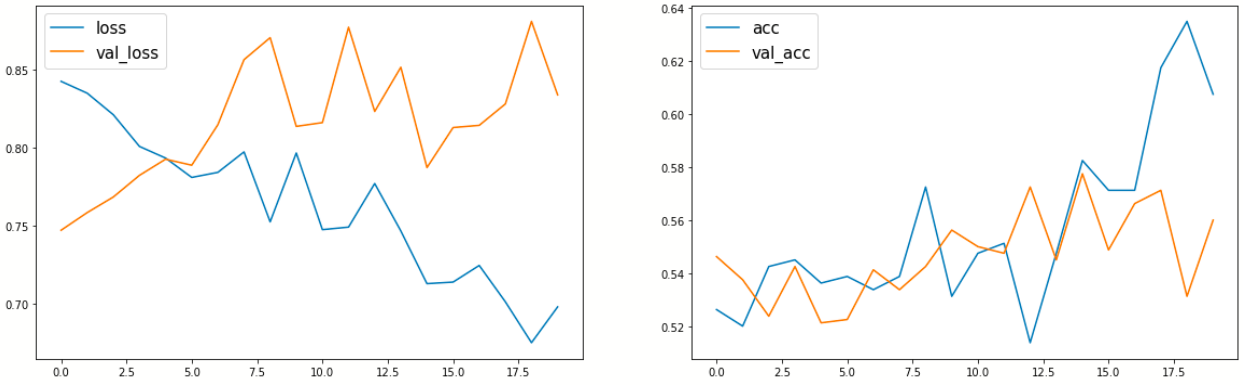

Figure 25: Loss and Accuracy Plots for InceptionV3

The InceptionV3 model performs much poorer than the ResNet50 model in terms of accuracy and loss values for both the training and validation sets. Figure 25 shows that there are signs of the model being overfitted due to deviations between the training and validation accuracy and loss plots. Hence, this could signify that a deep model performs better than a wide model for this particular dataset. In addition, we can also observe that the scores generated from Kaggle are also much poorer for the InceptionV3 model as opposed to ResNet50.

The following scores were retrieved after uploading the CSV generated for the test set onto Kaggle.

Private Kaggle Score: 0.580

Public Kaggle Score: 0.566

4.4.3 VGGFace

VGGFace is a series of pre-trained models that are developed for face recognition and the dataset used contains 3.31 million images of 9131 subjects, with an average of 362.6 images for each subject. Images are downloaded from Google Image Search and have large variations in pose, age, illumination, ethnicity and profession (e.g. actors, athletes, politicians) (Brownlee, 2019). The trained models are then evaluated on benchmark face recognition datasets, demonstrating that the model is effective at generating generalized features from faces, making it suitable for this research due to the importance of recognizing facial features.

VGGFace has been trained on ResNet50 and SENet50 - both of which are deep models. Hence, this further affirms the fact that a deep model is more suitable in this prediction. SENet50 is built on the ResNet50 model, with an added Squeeze-and-Excitation (SE) block. The purpose of the added SE block is to take into account how relevant each of the 3 channels are when computing outputs so that weights can be further optimized (Samavati, 2021).

The following subsections aim to discuss an overview of both models, as well as their results. With reference to Table 13, parameters used across ResNet50 and SENet50 are that of Section 4.4.1 as produced the best results given the computational power that is available.

Parameter	Value
Activation Function (Hidden Layers)	ReLu
Activation Function (Output Layer)	Sigmoid
Loss Function	Binary Crossentropy
Learning Rate	0.00001
Optimizer	Adam

Table 13: VGGFace Parameters

Using ResNet50

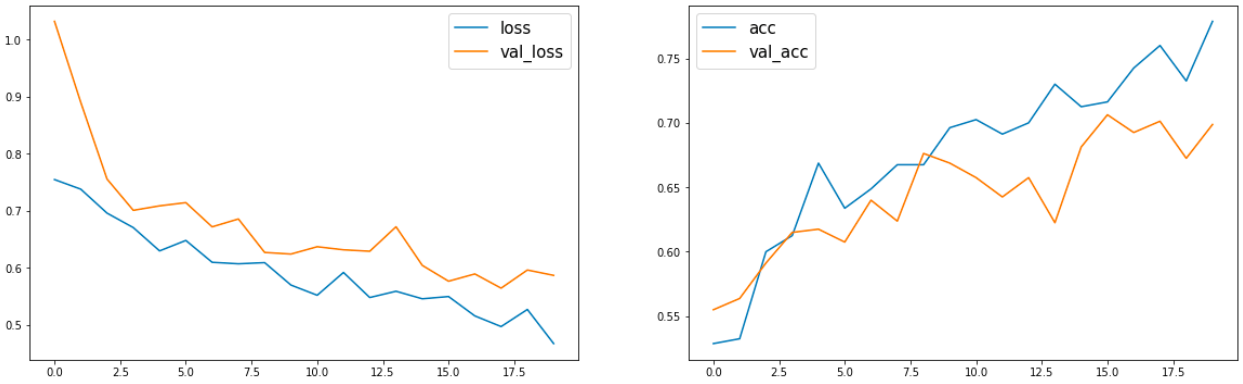

Figure 26: Loss and Accuracy Plots for VGGFace ResNet50

The following scores were retrieved after uploading the CSV generated for the test set onto Kaggle.

Private Kaggle Score: 0.771

Public Kaggle Score: 0.787

Using SENet50

	224 × 224		320 × 320 / 299 × 299	
	top-1 err.	top-5 err.	top-1 err.	top-5 err.
ResNet-152 [10]	23.0	6.7	21.3	5.5
ResNet-200 [11]	21.7	5.8	20.1	4.8
Inception-v3 [44]	-	-	21.2	5.6
Inception-v4 [42]	-	-	20.0	5.0
Inception-ResNet-v2 [42]	-	-	19.9	4.9
ResNeXt-101 (64 × 4d) [47]	20.4	5.3	19.1	4.4
DenseNet-264 [14]	22.15	6.12	-	-
Attention-92 [46]	-	-	19.5	4.8
Very Deep PolyNet [51] †	-	-	18.71	4.25
PyramidNet-200 [8]	20.1	5.4	19.2	4.7
DPN-131 [5]	19.93	5.12	18.55	4.16
SENet-154	**18.68**	**4.47**	**17.28**	**3.79**
NASNet-A (6@4032) [55] †	-	-	17.3‡	3.8‡
SENet-154 (post-challenge)	-	-	**16.88‡**	**3.58‡**

Figure 27: Comparison of Errors (Based on Other Research) (Tsang, 2019)

We can observe the effectiveness of adding the SE block to a ResNet model from Figure 27 as the errors have reduced significantly. Similar to the ResNet50 model that had been pre-trained with the VGGFace dataset, we can observe that despite having higher accuracies and lower loss values across the board, the model is not overfitted as the training and validation plots are not observed to be deviating from one another. In fact, the SENet50 model performs better than the ResNet50 model in terms of Kaggle score.

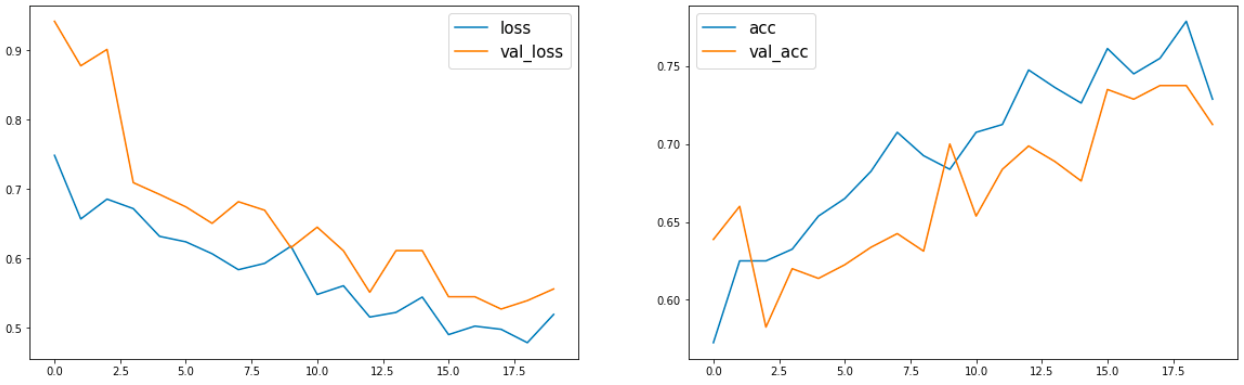

Figure 28: Loss and Accuracy Plots for VGGFace SENet50

The following scores were retrieved after uploading the CSV generated for the test set onto Kaggle.

Private Kaggle Score: 0.797

Public Kaggle Score: 0.795

5 Challenges

Our first model, K-Means Clustering, gave us the lowest accuracy levels as compared to the other models mainly due to the fact that the clusters are of differing sizes. In order to solve this, we need to do post-processing of the data by eliminating small clusters that may represent outliers, splitting "loose" clusters with relatively high Sum of Squared Errors (SSE), and merging clusters that are "close" with relatively low SSE. However, we decided that the other models were more efficient and it would render more accurate results without much post-processing needed.

Another major challenge for us was to define what kind of features to use. As we know, the input image data for one individual is very diverse, it is captured from different angles, age, background and so on. Hence to identify similarities between two people, we normally rely on observing their facial features, such as facial shapes and orientation of eyes and nose. Due to our limited knowledge in image feature engineering, we are restricted to only using pixel colour as our features. This would greatly reduce the accuracy of our supervised models especially for Support Vector Machines and Random Forest.

For SVM in particular, one limitation to note is that they work best on smaller, complex datasets. The complexity of the algorithm is heavily dependent on the dataset's size. The bigger the dataset is, the more costly the algorithm.

For the CNNs, the biggest limitation faced would be that training of CNNs is computationally expensive and the team lacks the necessary computational resources required to execute the training procedures and find a best fitted model. Specifically, 25GB of RAM will be exhausted when more than 20 epochs are used to train a single model. Taking into consideration the limited computational resources available, Table 14 outlines the parameters used to fit the respective CNN models for this research.

Parameter	Value
Batch Size	16
Epochs	20
Steps Per Epoch	50
Validation Steps	50

Table 14: Parameters for Model Fitting

In an ideal scenario where computational resources are more easily accessible, parameter values set in Table 14 would be increased to find a convergence point via grid search cross validation. This would confer greater accuracy levels.

6 Conclusion

After evaluating all models, our group came to a conclusion that neural networks are more suitable for this particular dataset. In particular, **SENet50 pre-trained with facial images from the VGGFace dataset** performed the best and gave the highest Kaggle score.

Throughout this project we also picked up on a few key learning points to take away.

1. **Don't Underestimate the Power of Exploratory Data Analysis**

In hindsight, we realised that Exploratory Data Analysis helped our group make sense of the complicated dataset given to us, to better understand the images and the structure of the file directories. It also helped lead us to consider dimensionality reduction later on in our pre-processing of data.

2. **Data Pre-Processing is Key**

Data preprocessing was one of the most essential steps in our process. The techniques we utilized in our data preprocessing ensured that our models were able to reach convergence in a shorter duration. In particular, we felt that minimizing the variance of each feature using image normalization helped us save a substantial amount of computational power.

3. **CNNs work better than other Machine Learning Techniques for this dataset**

For this project in particular, our team realised that convolutional neural networks performed far better than the other algorithms. Perhaps this is due to their ability to automatically detect essential features without the need of human supervision. This, coupled with their inbuilt convultional layer which effectively reduces the high dimensionality of images, makes CNNs more accurate in making predictions for image classification.

4. **Pre-training using images consisting of faces improves accuracy**

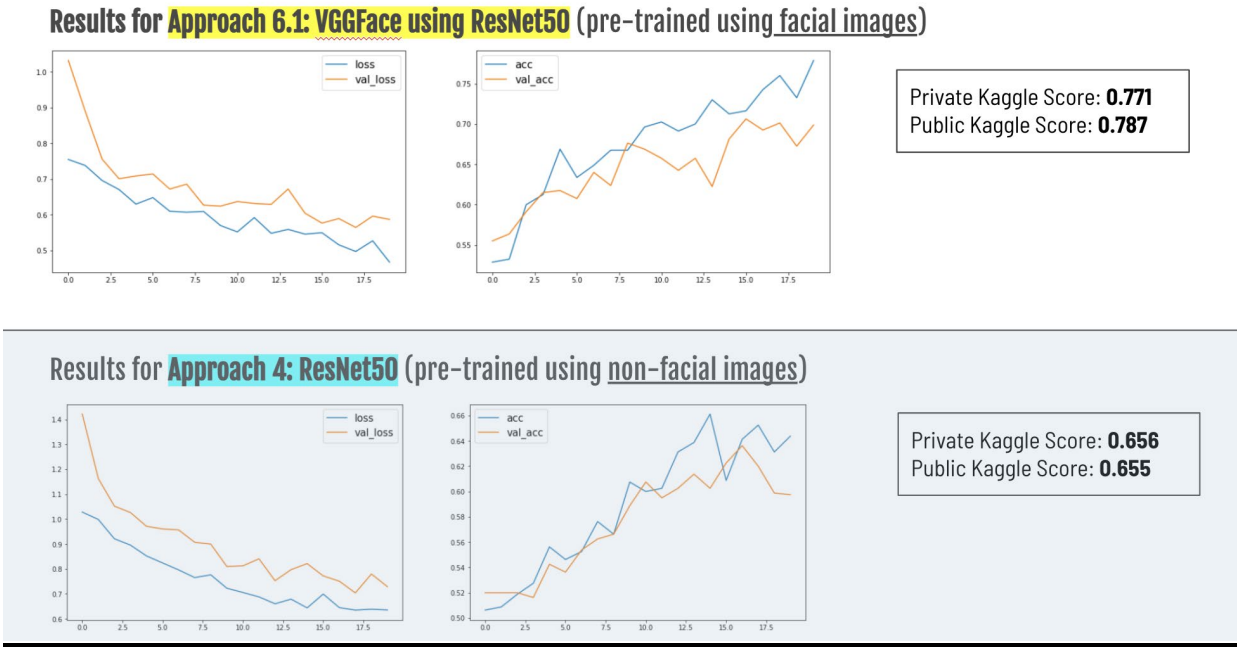

Figure 29: Comparison between 2 models

Comparing the 3 different CNNs we used in this project, we realised that out of the 3, VGGFace was pre-trained using images with human faces, while ResNet50 and InceptionV3 were pre-trained

using the ImageNet Database which did not contain any images with humans as the subject of the image. Even though the two later algorithms were pre-trained on more than 14 million images, more than 4 times that of VGGFace, VGGFace still gave us the highest accuracy. This shows how pre-training of the models using more relevant images is a much more important factor than the quantity of images used in pre-training.

9 References

Brownlee, J. (2019, June 5). *How to Perform Face Recognition With VGGFace2 in Keras*. Machine Learning Mastery. Retrieved October 16, 2022, from https://machinelearningmastery.com/how-to-perform-face-recognition-with-vggface2-convolutional-neural-network-in-keras/

DataGen. (n.d.). *ResNet-50: The Basics and a Quick Tutorial*. Datagen. Retrieved October 16, 2022, from https://datagen.tech/guides/computer-vision/resnet-50/

eduCBA. (n.d.). *Keras ResNet50 | Image File Handling and Transfer Learning*. eduCBA. Retrieved October 16, 2022, from https://www.educba.com/keras-resnet50/

Peltarion. (n.d.). *Dense block*. Peltarion. Retrieved October 16, 2022, from https://peltarion.com/knowledge-center/modeling-view/build-an-ai-model/blocks/dense

Samavati, T. (2021, September 27). *Squeeze-and-Excitation block explained | by Taha Samavati*. Medium. Retrieved October 16, 2022, from https://medium.com/@tahasamavati/squeeze-and-excitation-explained-387b5981f249

Tsang, S.-H. (2019, May 8). *Review: SENet — Squeeze-and-Excitation Network, Winner of ILSVRC 2017 (Image Classification)*. Towards Data Science. Retrieved October 16, 2022, from https://towardsdatascience.com/review-senet-squeeze-and-excitation-network-winner-of-ilsvrc-2017-image-classification-a887b98b2883

K-Means: https://i.stack.imgur.com/clDB3.png

RandomForest: https://www.analyticsvidhya.com/blog/2022/01/image-classification-using-machine-learning/

https://leslietj.github.io/2020/06/28/How-to-Average-Images-Using-OpenCV/

SVM: https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/