

# **TDL PROJECT REPORT**

## **VISUAL KINSHIP RECOGNITION**

B PRAVENA (PES2UG19CS076)  
DEEPA SHREE C V (PES2UG19CS105)  
KEERTHI JOSHI K(PES2UG19CS181)

### **Introduction**

Automated kinship recognition from face is a relatively recent problem that is mainly studied by the application of Deep Learning techniques. The problem at hand is mainly that, given a pair of images, a prediction of whether the people are related or not, needs to be made. The output is a label whether the people are related or not.

Visual kinship recognition has a lot of potential uses. A few of them are listed here:

- Family photo-album organisation
- Forensic investigation
- Missing person cases
- Social media analysis
- Creation of family trees and image annotation
- Kin-based face synthesis
- Connect families

To solve the problem using deep learning, we use a Siamese network. These networks work with image pairs and then find the similarity between the obtained feature vectors to predict the final output label.

## Dataset Description:

The dataset used is Families in the Wild(FIW). The folder ‘train’ consists of subfolders of families with names(F0123), then this family folder contains subfolders for individuals (MIDx). Images in the same MIDx folder belong to the same person. Images in the same folder F0123 belong to the same family.

## Literature Survey

- A. Shadrikov, "**Achieving Better Kinship Recognition Through Better Baseline**," 2020 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2020), 2020, pp. 872-876, doi: 10.1109/FG47880.2020.00137.

This paper uses transfer learning to solve the problem statement. The models used are RetinaFace for face registration and ArcFace for face verification model. Using these models the authors have built a pipeline that achieved state-of-the-art performance on two tracks in the recent Recognizing Families In the Wild Data Challenge.

- Robinson, Joseph & Shao, Ming & Wu, Yue & Liu, Hongfu & Gillis, Timothy & Fu, Yun. (2018). **Visual Kinship Recognition of Families in the Wild**. IEEE Transactions on Pattern Analysis and Machine Intelligence. PP. 1-1. 10.1109/TPAMI.2018.2826549.

The dataset used is Families in the Wild(FIW), whose description is already mentioned in the previous section. To extend the dataset, the authors have further extended the dataset by using a novel approach of labelling unsupervised data by using a proposed clustering algorithm. These methods save a lot of time as human effort is not required. The authors have performed two tasks, kinship verification and family classification. Pre-trained CNN models are fine tuned and applied on the dataset, as they outperform conventional CNN methods. To evaluate the models, the model performance is compared with human performance and they outperformed human predictions.

- Mengyin Wang, Xiangbo Shu, Jiashi Feng, Xun Wang, Jinhui Tang, **“Deep multi-person kinship matching and recognition for family photos”**, 2020, 107342, ISSN 0031-3203, <https://doi.org/10.1016/j.patcog.2020.107342>.

The proposed Deep Kinship Matching and Recognition (DKMR) framework contains three modules. First, a deep kinship matching model (termed DKM-TRL) to predict kin-or-not scores by integrating the triple ranking loss into a Siamese CNN model. Second, a deep kinship recognition model (named DKR-GA) to predict the exact kinship categories, in which gender and relative age attributes are utilised to learn more discriminative representations. Third, based on the outputs of DKM-TRL and DKR-GA, they have proposed a reasoning conditional random field (R-CRF) model to infer the corresponding optimal family tree by exploiting the common kinship knowledge of a nuclear family.

- Y. Guo, H. Dibeklioglu and L. Van Der Maaten, **"Graph-Based Kinship Recognition"**, *2014 22nd International Conference on Pattern Recognition*, 2014, pp. 4287-4292, doi: 10.1109/ICPR.2014.735.

Image-based kinship recognition is an important problem in the reconstruction and analysis of social networks. Prior studies on image-based kinship recognition have focused solely on pairwise kinship verification, i.e. on the question of whether or not two people are kin. Such approaches fail to exploit the fact that many real-world photographs contain several family members, for instance, the probability of two people being brothers increases when both people are recognized to have the same father. In this work, they have proposed a graph-based approach that incorporates facial similarities between all family members in a photograph in order to improve the performance of kinship recognition. In addition, they have introduced a database of group photographs with kinship annotations.

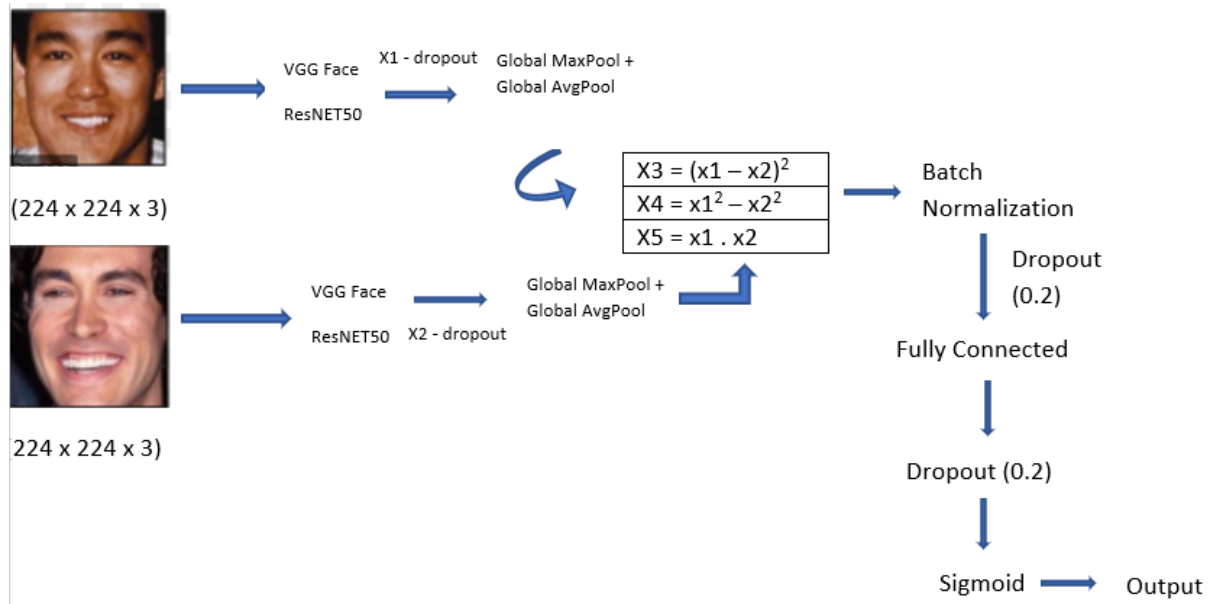
- X. Wu, E. Granger, T. H. Kinnunen, X. Feng and A. Hadid, **"Audio-Visual Kinship Verification in the Wild,"** 2019 International Conference on Biometrics (ICB), 2019, pp. 1-8, doi: 10.1109/ICB45273.2019.8987241.

This paper takes into account the voice modalities along with face modalities to improve kinship verification accuracy. A multi-modal kinship database called TALking KINship(TALKIN) is introduced which contains several pairs of video sequences with the subjects talking. A deep Siamese network for multi-modal fusion of kinship relations is proposed. Implementation is done by fine-tuning the VGG-Face CNN cascaded with an LSTM network for the face modality. For voice modality, they have fine-tuned a ResNet-50 pre-trained on VoxCeleb2. Finally a FC layer is added to fuse the audio and visual information. Experiments with the dataset indicate that the proposed network provides a significantly higher level of accuracy over baseline uni-modal and multi-modal fusion techniques. Results suggest that audio(vocal) information is complementary and useful for kinship verification problem.

- X. Zhang, M. XU, X. Zhou and G. Guo, **"Supervised Contrastive Learning for Facial Kinship Recognition,"** 2021 16th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2021), 2021, pp. 01-05, doi: 10.1109/FG52635.2021.9666944.

The authors of this paper have trained a deep CNN with supervised contrastive learning approach to address three different kinship recognition tracks(i.e.,kinship verification, tri-subject verification, and large-scale search-and-retrieval). Kinship verification is to determine whether a given pair of face images have a kinship. Tri-subject Verification is to determine whether the given parent images have a kinship relation to a child image, which is to a 2-to-1 verification problem. The search and retrieval task is to find images in the gallery that are most likely to have a kinship with the probe. ArcFace (pre-trained ResNet101) is used as a feature extraction network and SGD is used as the optimizer.

## Design -:



The two images are inputted to the model which is loaded on VGG16 with weights trained on ResNet50. To finetune our images with Keras we have to change dimensions to  $128 \times 128 \times 3$ .

$X3$  - euclidean distance between points in 2 images

$X4$  - Manhattan distance between points of 2 images

$X5$  - Regularization

For our model to become more generalised we add some Dense layers

Compile method() → metric argument → Optimizer → Adam

Loss → Binary cross Entropy

Metrics → Accuracy

# Implementation:

## Installing the required libraries:

The current solution uses VGGFace with the weights of Resnet50 as Convolutional network part of the Siamese Network. Transfer learning performed better than handbuilt CNNs, hence VGGFace is used. To use this model, vggface has to be installed from keras. A few other installations required are keras\_applications, keras\_preprocessing to support vggface installation. A change has to be made in the models.py file of vgg\_face.

'Keras.engine.topology' has to be overwritten with 'tensorflow.keras.utils', as the former library is not supported by keras anymore. The following code does the job.

```
filename = "/usr/local/lib/python3.7/dist-packages/keras_vggface/models.py"
text = open(filename).read()
open(filename, "w+").write(text.replace('keras.engine.topology', 'tensorflow.keras.utils'))
import tensorflow as tf
```

## Importing the required libraries:

Libraries required for preprocessing the input and saving the model.

```
import h5py
from collections import defaultdict
from glob import glob
from random import choice, sample
import cv2
import numpy as np
import pandas as pd
from tqdm import tqdm
```

- h5py to save the trained model.
- defaultdict from collections to save the dictionary values as a map
- glob to match a regex pattern. Here, we use it to match the file paths of images.
- choice and sample from random, to select images at random to pass to the model while training
- cv2 to load the image given the path for preprocessing
- numpy to convert the image to an array

- pandas to load data from train\_relationships.csv
- tqdm to get the progress bar of an operation or a cell when it's running.

```
import tensorflow as tf
import keras
import tensorflow.python.keras.engine
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from keras.layers import Input, Dense, GlobalMaxPool2D, GlobalAvgPool2D, Concatenate, Multiply, Dropout, Subtract, Lambda
from keras.models import Model
from tensorflow.keras.optimizers import Adam
from keras_vggface.utils import preprocess_input
from keras_vggface.vggface import VGGFace
from keras import backend as K
from keras.models import load_model
```

- tensorflow and keras to import various requirements required to build the neural network
- keras.callbacks to implement ModelCheckpoint, ReduceLROnPlateau to include callback while training the model
- keras.layers to implement the different layers of the neural network
- Model from keras.models to build the model
- Import Adam as the optimizer
- preprocess\_input to preprocess the image in the required format to pass it to vggface
- backend to plug distinct backends of Tensorflow to keras
- load\_model to load a saved tensorflow model saved via save()

## Preprocessing the input

```
[ ] %%time
#content/drive/MyDrive/TDL_PROJECT/train/F0002/MID1/P00009_face3.jpg
all_images=glob(train_folders_path+'/**/*.jpg') #paths of all images
train_images=[x for x in all_images if val_families not in x] #path of images used for training
val_images=[x for x in all_images if val_families in x] #path of validation images (belonging to families starting with F09)

CPU times: user 242 ms, sys: 207 ms, total: 449 ms
Wall time: 1.53 s

[ ] ppl=[x.split('/')[-3]+'/' + x.split('/')[-2] for x in all_images] #obtaining the people in the format give in train_relationship

#Mapping people to their faces (list of faces)
train_person_to_images_map=defaultdict(list)
for x in train_images:
    train_person_to_images_map[x.split('/')[-3]+'/' + x.split('/')[-2]].append(x)

val_person_to_images_map=defaultdict(list)
for x in val_images:
    val_person_to_images_map[x.split('/')[-3]+'/' + x.split('/')[-2]].append(x)

[ ] #Obtaining relationship pairs and converting them to tuples
relationships = pd.read_csv(train_file_path)
relationships = list(zip(relationships.p1.values, relationships.p2.values))
relationships = [x for x in relationships if x[0] in ppl and x[1] in ppl]

[ ] #Dividing the tuples into train and validation
train=[x for x in relationships if val_families not in x[0]]
val=[x for x in relationships if val_families in x[0]]
```

- Firstly, all the image paths are loaded into a list `all_images`. A random folder is selected as the validation folder. Here, F09 is chosen. All the train images are loaded into the `train_image` list and the validation image paths are stored in the `val_images` list.
- `ppl` is a list of all the image paths in the same format as `train_relationships.csv`.
- All the people in both the train and validation sets are mapped to their faces and the respective results are stored in `train_person_to_images_map` and `val_person_to_images_map`
- The relationship pairs are obtained and converted into tuples. These tuples are stored in the list `relationships`.
- These tuples are then divided into train and validation sets.

```
[ ] #reads the image and converts into numpy aarray and finally returns the image processed as required by VGGFace
def img2arr(path):
    img=cv2.imread(path)
    img=np.array(img).astype(np.float)
    return preprocess_input(img)

[ ] #Generator to use with fit_generator to generate data in batches
def data_generator(list_tuples,person_to_images_map,batch_size=16):
    ppl=list(person_to_images_map.keys())
    while True:
        batch_tuples=sample(list_tuples,batch_size//2)
        labels=[1]*len(batch_tuples)
        while len(batch_tuples)<batch_size:
            p1=choice(ppl)
            p2=choice(ppl)

            if p1!=p2 and (p1,p2) not in list_tuples and (p2,p1) not in list_tuples:
                batch_tuples.append((p1,p2))
                labels.append(0)
        for x in batch_tuples:
            if not len(person_to_images_map[x[0]]):
                print(x[0])
            X1=[choice(person_to_images_map[x[0]]) for x in batch_tuples]
            X1=np.array([img2arr(x) for x in X1])
            X2=[choice(person_to_images_map[x[1]]) for x in batch_tuples]
            X2=np.array([img2arr(x) for x in X2])
            labels = np.asarray(labels).reshape((-1,1))
        yield [X1,X2],labels
```

- `Img2arr` function helps in the preprocessing of the image and converting it to an array(a vector of pixel values).
- `data_generator` takes in the list of tuples of image paths, `person_to_images_map` and the required `batch_size` as the arguments. The keys of the input map are stored in a list, to choose images at random. A sample of related tuples are chosen and appended to a list `batch_tuples`. As these tuples are related, the label 1 is appended to a list 'labels'. The next half of the `batch_tuples` needs to contain image pairs



that are not related. This is because a Siamese network needs both positive and negative pairs to train on. Hence, two random images are chosen from the ppl list and checked for their parent folder. If the parent folder of both these images is the same then they are discarded, because we already have our share of related image pairs. After the number of image pairs is equal to the batch size, the loop is terminated.

The image pairs are then broken down to X1 and X2 for preprocessing and then passed through `img2arr` function. The labels list is resized to a 1d array for passing it to the model for training. The lists X1,X2 and the array labels are returned back using `yield`. We use `yield` instead of `return`, because we use `fit_generator` instead of `fit` to train the model.

### **Building the model:**


Since we are passing two inputs, we build a siamese network. We define our base model as VGGface whose weights are pretrained on Resnet50 dataset, using transfer learning. We pass our two inputs x1 and x2 to the base model as sister networks, where each network is a vertical concatenation of global maxpool and global average pool to get all the face embeddings.



The siamese distance metric used here is euclidean distance, manhattan distance and regularization on inputs. Euclidean distance between 2 points of two images is done to get the similarity between two images. As distance between two points increases, similarity decreases and viceversa. Manhattan distance is done with square of each point between two points in an image. Regularization is also added to get good results. For our model to be more generalized, we add some combinations of dense and dropout layer. We use 'relu' activation function for inner dense layers and 'sigmoid' for the outer dense layer.

The metrics considered is accuracy which include train and validation accuracy. The loss is binary crossentropy loss since it is a binary classification problem. We also use Adam's optimizer for optimization.

We make use of checkpoint to save the best model on validation accuracy. If validation accuracy is not increasing after some epochs, we tune it using reduce learning rate on plateau. A callback list containing of two parameters

namely checkpoint and reduce\_lr\_on\_plateau is generated for each epoch.

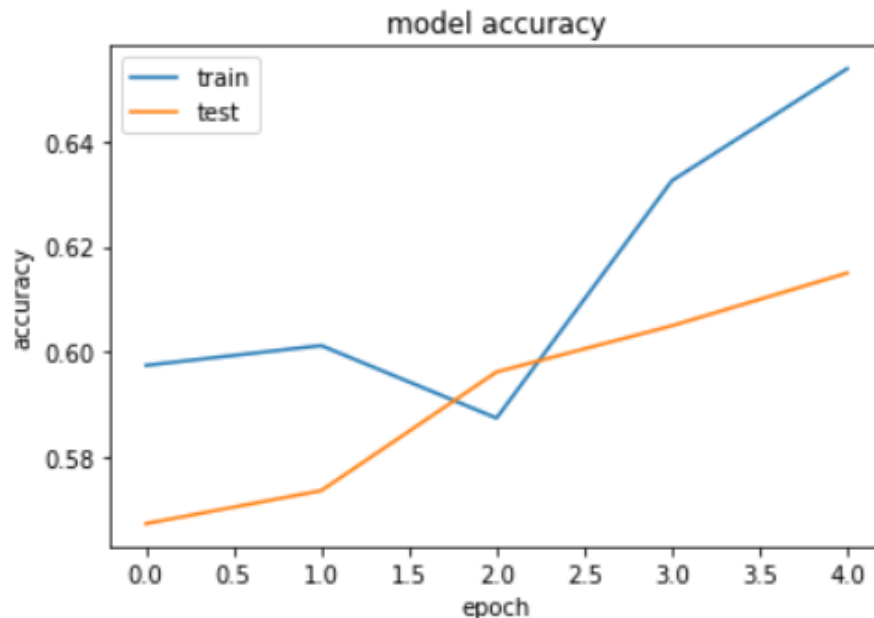
 Model: "model\_1"

Layer (type)	Output Shape	Param #	Connected to
input_7 (InputLayer)	[(None, 224, 224, 3 )]	0	[]
input_8 (InputLayer)	[(None, 224, 224, 3 )]	0	[]
vggface_resnet50 (Functional)	(None, None, None, 2048)	23561152	['input_7[0][0]', 'input_8[0][0]']
global_max_pooling2d_4 (Global MaxPooling2D)	(None, 2048)	0	['vggface_resnet50[0][0]']
global_average_pooling2d_4 (Gl obalAveragePooling2D)	(None, 2048)	0	['vggface_resnet50[0][0]']
global_max_pooling2d_5 (Global MaxPooling2D)	(None, 2048)	0	['vggface_resnet50[1][0]']
global_average_pooling2d_5 (Gl obalAveragePooling2D)	(None, 2048)	0	['vggface_resnet50[1][0]']
 concatenate_4 (Concatenate)	(None, 4096)	0	['global_max_pooling2d_4[0][0]', 'global_average_pooling2d_4[0][0]']
 concatenate_5 (Concatenate)	(None, 4096)	0	['global_max_pooling2d_5[0][0]', 'global_average_pooling2d_5[0][0]']
subtract (Subtract)	(None, 4096)	0	['concatenate_4[0][0]', 'concatenate_5[0][0]']
multiply_1 (Multiply)	(None, 4096)	0	['concatenate_4[0][0]', 'concatenate_4[0][0]']
multiply_2 (Multiply)	(None, 4096)	0	['concatenate_5[0][0]', 'concatenate_5[0][0]']
multiply (Multiply)	(None, 4096)	0	['subtract[0][0]', 'subtract[0][0]']
subtract_1 (Subtract)	(None, 4096)	0	['multiply_1[0][0]', 'multiply_2[0][0]']
multiply_3 (Multiply)	(None, 4096)	0	['concatenate_4[0][0]', 'concatenate_5[0][0]']
concatenate_6 (Concatenate)	(None, 12288)	0	['multiply[0][0]', 'subtract_1[0][0]', 'multiply_3[0][0]']
dense_2 (Dense)	(None, 100)	1228900	['concatenate_6[0][0]']
dropout (Dropout)	(None, 100)	0	['dense_2[0][0]']
dense_3 (Dense)	(None, 1)	101	['dropout[0][0]']
=====			
Total params: 24,790,153			
Trainable params: 24,737,033			
Non-trainable params: 53,120			

The metric values are displayed when we call the generator\_fit() function which is logged to History object and returns History object.

## Results:

The model is made to run for five epochs and the train accuracy is 65.38% and validation accuracy is 61.5%.



## Future Scope

This model determines whether the given pictures are related by only considering the facial features.

The model's accuracy can be improved by also considering other attributes as vocals, genetics, etc. The vocals can be converted to images of frequency and using different models it can be found if the voices or audio signals are similar. Similarly, the genetic structures can also be compared to check for kinship. This method might provide the most accuracy.

Models can also be built to predict by considering the DNA of the people we wish to run the kinship recognition on.

Further, the project can be extended to predict the type of relationship between the given people, based on features like age, gender etc.

## References

- [1]<https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942>
- [2]<https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939>
- [3]<https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd09190245ce>
- [4]<https://medium.com/@kenneth.ca95/a-guide-to-transfer-learning-with-keras-using-resnet50-a81a4a28084b>
- [5]<https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-with-convolutional-neural-networks-3-datasets/>

