Team Nightwing:

Kunind Sahu, Karan Dahiya and Sharnam Faria

Problem Modelling

We modelled the problem using a Siamese Network paradigm. We took a pair of images, and passed them through the same network (that is two different networks with the same architecture with weight sharing), embedded both of them separately and tried to get a similarity probability as an output using the Softmax function. This modelling seemed very natural, given the format of the training data.

Additionally, there existed a **class imbalance** in the training data. The training data only had **36% positive examples** - examples of similar faces/people. Thus we used a **weighted version** of the **Cross Entropy Loss** as our objective function**,** to deal with this and improve the model's performance.

The optimizer used is AdamW - which was proposed in the paper "Decoupled Weight Decay Regularization". AdamW is a modification of Adam in PyTorch, where AdamW corrects the weight decay issue prevalent in many momentum based optimizers (Adam is a combination of RMSProp and Momentum based Gradient Descent). Hyperparameter Settings are given in the last section.

Layer Information

1. Multitask Cascaded Convolutional Neural Networks (MTCNN)
2. Inception ResNet V1
3. Neural Tensor Network
4. RBF Kernel Similarity
5. Feedforward Neural Net

Multi-task Cascaded Convolutional Neural Networks (MTCNN)

A pre-trained MTCNN layer has been used - implemented in **`facenet_pytorch`** to pre-process images by intelligently cropping only the facial region of the images. All those images, if any, in which the MTCNN model could not find any faces, have simply been resized to the size 160px * 160px. This model is the current **State of the Art** in face detection.

Inception ResNet V1

A pre-trained Inception ResNet V1 layer has been used - implemented in **`facenet_pytorch`** to generate vector embeddings of the input cropped images. The pre-trained model has been trained on the VGGFace2 Dataset which contains **3.31 million images of 9131 subjects**. We implemented the concept of **Transfer Learning** for this layer. We **froze all but the last layer** of this model, to allow finetuning of the ResNet model to better fit our Dataset. This is done under the assumption that all the layers but the last (they are convolution layers) contain general information about extracting facial landmarks from images such as the eyes, nose, mouth etc. The last layer is tuned to sit well with the dataset we have. This model is also a **State of the Art** in face recognition tasks.

Neural Tensor Network

The Neural Tensor Network along with the RBF Kernel similarity **begin the decoder** part of our model. The Neural Tensor Network has been used successfully by Bai et.al in SimGNN for the problem of **Graph Similarity Computation** for calculating **Graph Edit Distances**. Also, noting that since graphs can be understood as a generalization of images, or **images can be considered as a special form of graphs**, we thought that this technique could prove to be successful here as well. Additionally Neural Tensor Networks have proven their mettle in **computing semantic similarity between** a pair of **word** embeddings. This layer takes two image embeddings as an input and calculates 'K' similarity scores between the two embeddings (where K is a hyperparameter).

What this algorithm does is, instead of directly calculating the inner product/ cosine similarity between the pair of image embeddings; which often leads to a weak modelling of the interaction between the two vectors, we use the NTN formulation. It ensures that there is an **explicit multiplicative relation** between the two embeddings; such that each **tensor slice** in the tensor network learns a **different relation** between the image embeddings, instead of simply concatenating the two image embeddings and feeding it into a Feedforward Neural Network.

RBF Kernel

The RBF Kernel similarity, is just a sanity check to supplement the similarity scores of the Neural Tensor Network. Its output is just a single value. It takes in two image embeddings and calculates the value of the exponential raised to the power of the square of the L2-Norm of the difference between the two vector embeddings. Although, this part is just a helper function to the NTN Layer, and can be omitted

Feedforward Neural Network

The output from the Neural Tensor Network and the RBF Kernel are concatenated and jointly fed into the feedforward neural network layer which predicts if the two images are of the same person or not. Assuming a binary classification problem of same or not-same.

Hyperparameter Settings

1. MTCNN Layer

   Used the implementation of the facenet_pytorch framework.

2. Inception ResNet V1 Layer

   Used the implementation of the facenet_pytorch framework. With the exception that all the layers before and including the last Convolution Layer block were frozen to not have gradients pass through them. Only the layers after the last convolution block were unfrozen and were allowed to update their parameters to allow fine-tuning them to the dataset we have at hand.
   The dimension of the output image embeddings is 512. It takes in a (3,160,160) dimensional PyTorch Tensor

3. Neural Tensor Network Layer

   The input dimension = 512
   Output Dimension = No. of Similarity Scores = K = 128 (see the SimGNN paper)
   Non -Linearity : Leaky ReLU with negative slope = 0.1

4. RBF Kernel

   Gamma = 1

5. Feedforward Neural Network Layer

   (128+1, 64) -> (64, 32) -> (32, 16) -> (16, 8) -> (8, 2)
   All the layers use a ReLU activation function

6. Optimizer Settings

   Batch Size = 64
   Optimizer = AdamW
   Learning Rate = 0.005
   Weight Decay = 10^-5
   Epochs = 3

Model Evaluation

We achieved a train accuracy of about **93.55%**. We did not do any kind of Hyperparameter tuning, since the search space and the computational time was huge. Although, I believe that the hyperparameters we have used are good enough for the job, to get a competitive accuracy.

Areas of Improvement

1. Hyperparameter Tuning to improve the performance of the architecture. Instead of a Grid Search due to a large search space, one could use Bayesian Optimization instead to tune the hyperparameters.
2. Data Augmentation - This is a trick which we could not utilize owing to time constraints. This is essentially inducing random image transformations to increase the amount of data we have and thus improve our models performance by tuning the weights properly.
3. Robust image pre-processing - we could have pre-processed the images much better and could have tried to reduce the immense background noise present to make our training data much more uniform and cogent.
4. Literature Review for the current state of the art in similarity search/ siamese decoder networks to improve accuracy.