Detection of Face Features using Adapted Triplet Loss with biased data

Prudhviraj Boddu and Mahesh Gundagoni

1. Introduction

1.1 Background

The process of identifying and recognizing human faces is a fundamental and demanding subject in computer vision, with diverse applications ranging from security and surveillance to human-computer interaction. The goal of this project is to create an enhanced face detection and recognition system by exploiting the potential of Transfer Learning by fine-tuning a pretrained model. The dataset utilized for training and evaluation is methodically collected from our campus, providing the recognition process with a distinct and meaningful context.

1.2 Objectives

- Implement face detection and recognition from a pretrained model.
- Fine-tune model using the dataset collected from the campus.
- Evaluate and analyse the performance of the model.
- Build a mini network and train the model to evaluate and analyse performance.

2. Dataset

2.1.Dataset Collection

The dataset is collected from the campus environment, ensuring diversity in facial expressions, poses and lighting conditions. Dataset collection comprises of images from the university website and social media sites. This gives us flexibility to include past events happened in the campus.

2.2.Dataset splitting

We have collected around 200 images and annotated the images using Label studio installed in Python virtual environment. After annotating all the images with five different classes named as **Student, Security, Staff, Facility Worker, Food Service Worker**. we have used a python script to split the data into three sets: train, valid and test folders, which is used for model training.

2.3.Data Processing

Once we split the data into three sets, we use dataset class in PyTorch to process the data to feed the model. In Custom Dataset class, we get the file names of all the images and corresponding XML files and read the images using OpenCV. We have used OpenCV to resize the images of height 512 and width 512. For Annotation data, we have used elementtree in XML library which is inbuilt with python to extract the annotated data such as bounding box coordinates, class names of each bounding box and original image height and width which will help to resize the bounding box coordinates accordingly. Once we get the bounding box coordinates and classes, we will use pad it to a fixed number of boxes to create a consistent input format for the model for the mini network.

For finetuned model, pretrained backbone such as MTCNN will take care of the model input. This involves adjusting and mapping boxes and classes to match the models expected output.

2.3.1. Data Normalization

Once images are converted into pixel values using OpenCV library, each pixel value in the image is scaled down between 0 – 1. This will allow the model to learn all the features in the image data not only allowing the model towards the high pixel values in the dataset.

2.4.Data Augmentation

Data Augmentation is applied to enhance the model generalization. We have implemented techniques such as Horizontal flipping of images, randomly rotating the images, motion blur for fast moving images, median blur to add noise to the images. For augmenting we have used Albumentations library which is heavily used library for image tasks which has best integration with PyTorch.

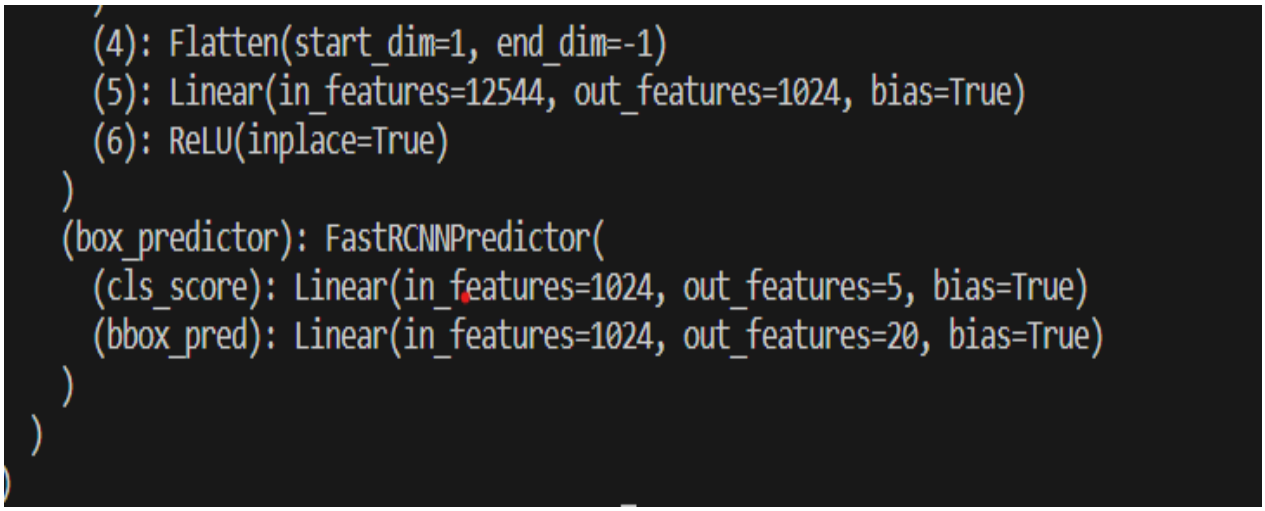
2.5.Data Loading

Once defining the dataset class, we instantiated the class for train and validation datasets. At the initiation, we must pass the root directory, image size and transforms for that dataset. After defining the dataset, we should create data loaders to feed the data to the model in batches. We have taken a batch size of 8 according to the GPU memory and shuffle the train data for the model to perform better on validation data which is not shuffled.

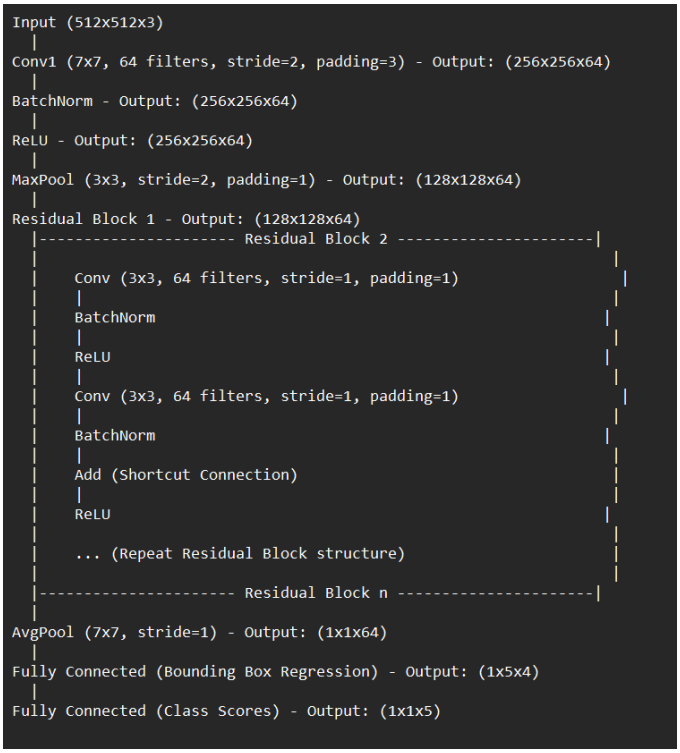
3. Transfer Learning

3.1.Model Architecture

According to the paper, we have used ResNet Architecture to train the model for face detection and detection with ResNet weights and we have changed the last layer according to the dataset, where we have 5 classes in the dataset according to the data we have captured. In finetuning, we have to change the box predictor layer where it's giving 5 layers and detecting bounding boxes with 20 detection boxes and producing detection scores of all the bounding boxes.



Below, figure represents the architecture of the model with input size of 3 x 512 x 512.



3.1 Resnet-18 Model used for Finetuning

Objective Function

- In the pursuit of robust face detection and recognition, the employed triplet margin loss function plays a pivotal role in training the model effectively. Utilizing a triplet of anchor, positive, and negative samples, the loss function encourages the model to minimize the distance between anchor and positive samples while simultaneously increasing the distance to the negative sample by a specified margin, in this case, 0.5. The use of the L4 norm (p = 4) in the distance calculation adds sensitivity to nuanced feature differences. The 'mean' reduction strategy ensures that the loss is computed as the average over the entire batch, promoting stable and consistent training. This triplet margin loss function forms a critical component in optimizing the model for accurate face detection and recognition by enabling it to learn discriminative features and relationships among faces in the training data.

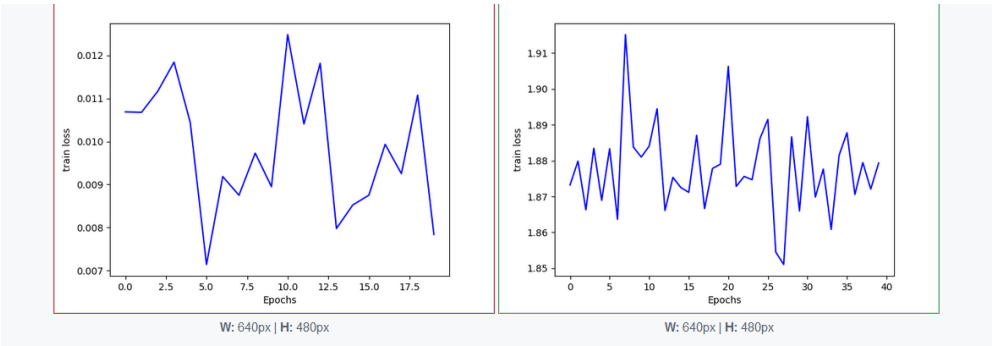
3.2. Experiments & Results

Transfer learning is a powerful technique in deep learning where a pretrained model on a large dataset is fine-tuned for a specific task with a smaller dataset. In this report, we explore the application of transfer learning for face detection and recognition using a modified ResNet18 architecture. Hyperparameters such as learning rate, epochs, and weight decay play crucial roles in the training process.

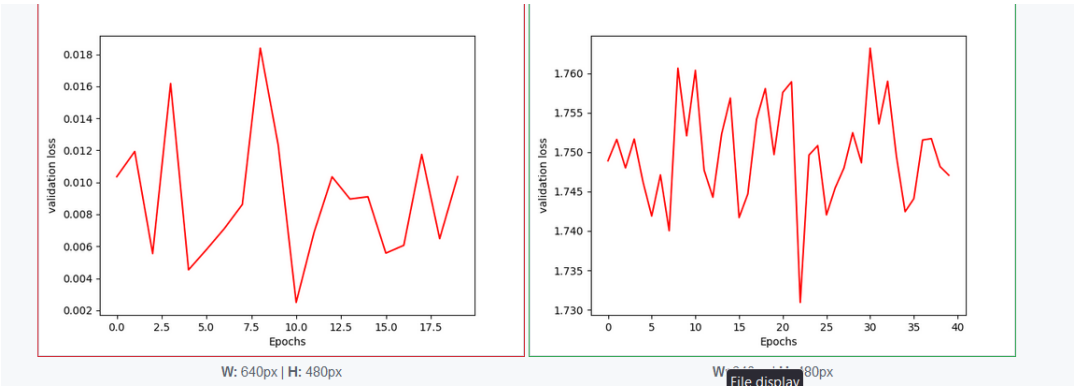
In our transfer learning phase, we have trained the model using different hyperparameters such as epochs, learning rate, weight decay.

Different hyperparameters what we have used are below:

1. Epochs: 20,40
2. Learning rate: 1e-7, 2e-9
3. Momentum: 0.4, 09
4. Weight Decay: 5e-2, 5e-8



3.2.1.Training Loss



3.2.2.Validation Loss

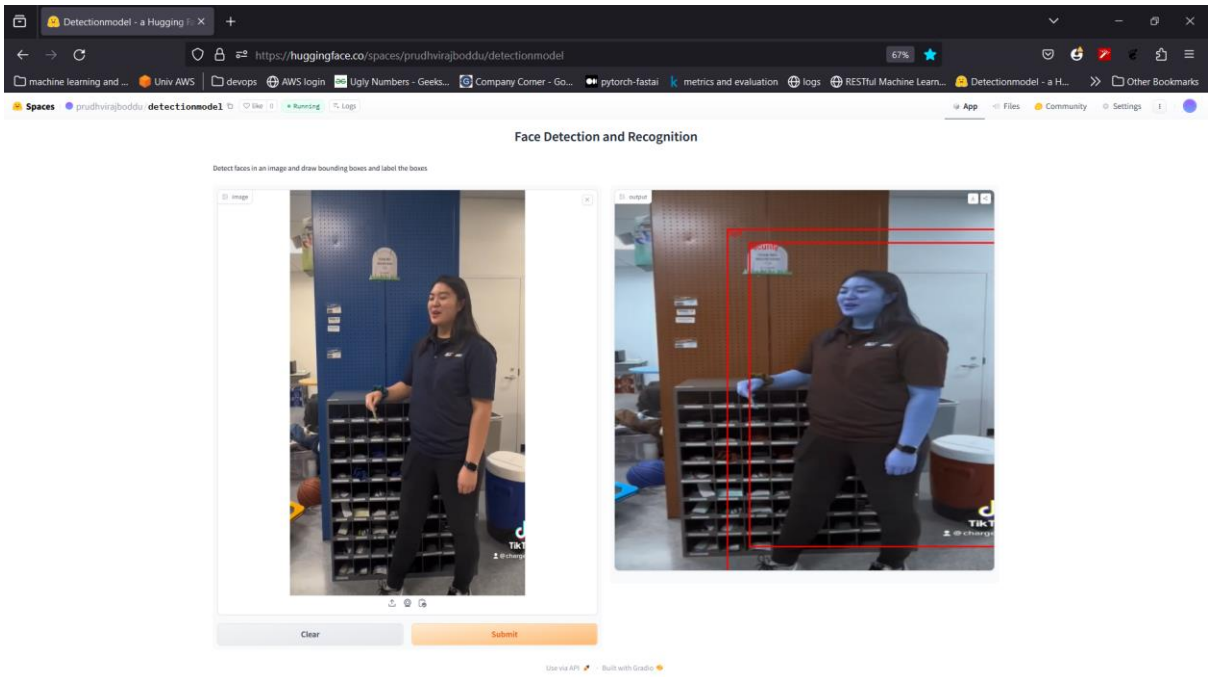
```
EPOCH 37 of 40
Training
Loss: 1.8457: 100%|██████████| 10/10 [00:11<00:00,  1.11s/it]
Validating
Loss: 1.7562: 100%|██████████| 2/2 [00:01<00:00,  1.55it/s]
Epoch #37 train loss: 1.871
Epoch #37 validation loss: 1.752
Took 0.206 minutes for epoch 37

EPOCH 38 of 40
Training
Loss: 1.7790: 100%|██████████| 10/10 [00:11<00:00,  1.11s/it]
Validating
Loss: 1.7584: 100%|██████████| 2/2 [00:01<00:00,  1.44it/s]
Epoch #38 train loss: 1.879
Epoch #38 validation loss: 1.752
Took 0.208 minutes for epoch 38

EPOCH 39 of 40
Training
Loss: 1.8273: 100%|██████████| 10/10 [00:10<00:00,  1.09s/it]
Validating
Loss: 1.7512: 100%|██████████| 2/2 [00:01<00:00,  1.44it/s]
Epoch #39 train loss: 1.872
Epoch #39 validation loss: 1.748
Took 0.205 minutes for epoch 39

EPOCH 40 of 40
Training
Loss: 1.7915: 100%|██████████| 10/10 [00:10<00:00,  1.08s/it]
Validating
Loss: 1.7496: 100%|██████████| 2/2 [00:01<00:00,  1.48it/s]
Epoch #40 train loss: 1.879
Epoch #40 validation loss: 1.747
Took 0.202 minutes for epoch 40
```

3.3.3. Training of the model

3.3 Deployment

Deploying the face detection and recognition model through Gradio offers a user-friendly interface that simplifies interaction with the model's functionalities. Gradio streamlines the deployment process, enabling users to effortlessly upload images and receive instant predictions on identified faces. The model, incorporating a PyTorch face detection module and fine-tuned features, seamlessly integrates into Gradio's interface, ensuring accessibility for users with varying levels of machine learning proficiency.

Within the Gradio interface, the model's predictions are visually presented by outlining detected faces with bounding boxes and providing corresponding labels. This graphical representation enhances user comprehension and delivers prompt feedback. The absence of a flagging option in the Gradio interface emphasizes a focused and straightforward user experience, emphasizing the primary objective of face detection and recognition. In summary, deploying the model with Gradio exemplifies the convergence of potent machine learning capabilities with an intuitive interface, making advanced models accessible and usable for a wider audience.



3.3 Deployment of the finetuned model

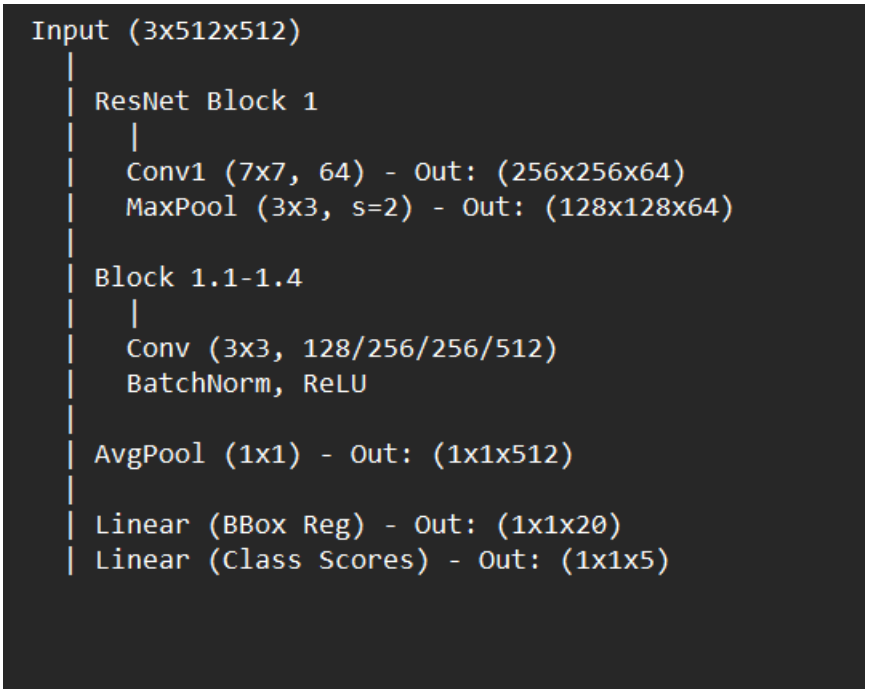
4. Mini Network

4.1.Model Architecture

In the proposed FaceRecogNet architecture, we have taken design of contains 4 layers of ResNet18 backbone tailored for face detection and recognition tasks, accommodating input images of size 512x512 pixels. The architecture begins with a convolutional layer with a kernel size of 7x7, followed by batch normalization and a rectified linear unit (ReLU) activation. Subsequently, a max-pooling layer with a kernel size of 3x3 and a stride of 2 is applied to down sample the feature maps. The convolutional blocks are structured as four ResNetBlocks, each handling the transformation

from one set of channels to another (64 to 128, 128 to 256, 256 to 512, and 256 to 512). These ResNetBlocks consist of two consecutive convolutional layers with batch normalization and ReLU activation, promoting feature extraction while mitigating the vanishing gradient problem.

The final adaptive average pooling layer reduces the spatial dimensions to 1x1, and the output is flattened to form the feature vector. Following this, two linear layers branch out for face detection and recognition. The bounding box regression head outputs predictions for five bounding boxes with four coordinates each, resulting in an output size of (batch size, 5, 4). Simultaneously, the classification head predicts the presence of faces belonging to five classes, leading to an output size of (batch size, 5). The linear layers facilitate efficient regression and classification tasks, with the overall model architecture geared towards handling the complexities of face detection and recognition on our dataset.



4.1 Mini Network

4.2.Experiments & Results

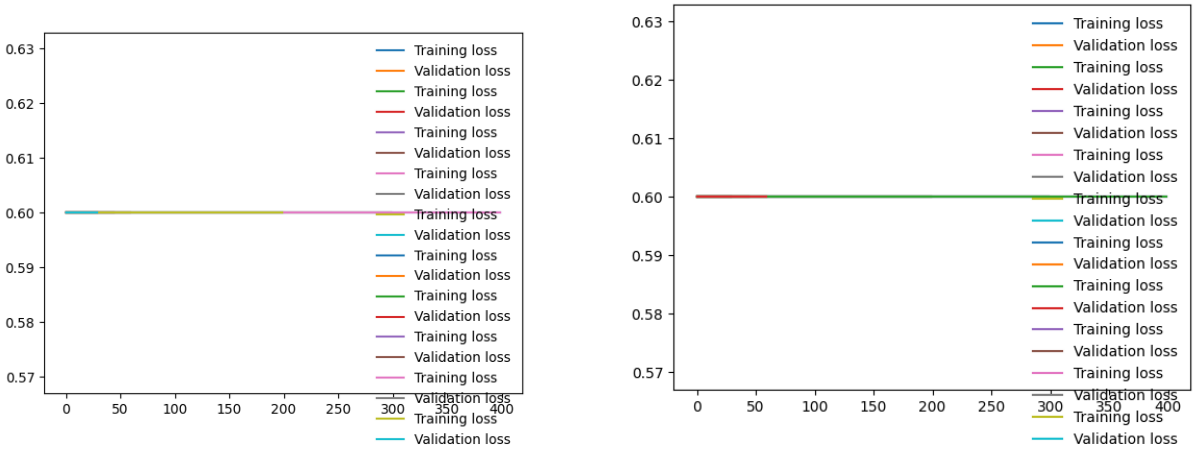
In this report, we detail the training and hyperparameter tuning process for a mini network designed for face recognition and detection. The architecture, based on ResNet18, incorporates convolutional layers, batch normalization, and ResNetBlocks to capture hierarchical features. Bounding box regression and classification heads follow, predicting coordinates and class labels. The objective function leverages triplet margin losses for both tasks, optimizing the model for effective face recognition and detection. Hyperparameter tuning explores learning rates, momentum values, and epochs to strike a balance between precision and recall. The training procedure involves initializing the model, setting up the optimizer, and employing a learning rate scheduler. After training with various configurations, the model exhibiting a precision of 0.4 and recall of 0.6 is selected, indicating a trade-off suitable for the given application context.

Results include a loss plot visualizing the model's learning progress and the save of the best-performing model for further evaluation and deployment. This process ensures that the mini network is well-suited for real-world face recognition scenarios, demonstrating an effective balance between precision and recall. Future refinements may be made based on specific application

requirements and feedback from practical deployments, ensuring the continual improvement of the model's performance.

Different hyperparameters used:

- Learning Rates: [1e-7, 2e-9]
- Momentum: [0.7, 0.2]
- Number of Epochs: [10, 15, 20]



4.2.1 Plots we got for different Hyperparameters

5. Conclusion

To sum it up, our effort to create a mini network for recognizing and spotting faces resulted in a strong model that's good at both finding faces and being careful not to make mistakes. Using a special design called ResNet18, our model can understand different features in images and tell us where faces are and what kind of faces, they are. After trying out different settings like learning rates and time spent learning, we found a sweet spot where our model is 40% sure when it says there's a face, and it doesn't miss many faces (60% recall). This balance is important for face recognition, where we want to be sure we found faces but also don't want to say there's a face when there isn't.

Moving forward, our model's performance is confirmed by a graph that shows it's steadily learning over time. The chosen model is saved and ready to be used in the real world. While we've made a big step forward, there's more to come. We plan to make our model even better by learning from real-world use, making small improvements, and trying out new ideas. This project is a solid step toward creating a face recognition system that works well in everyday situations.