Introduction:

Each football season, I compete in a fantasy football league with many of my friends and family. Though we do not play for a physical prize, the bragging rights secured by the winner last an entire year, a reward that transcends monetary value. Given the competitive nature of these leagues, I want another line of security on my fantasy football app to ensure that none of my leaguemates were to steal my phone and alter my team without my knowing. I will be using a facial verification model to ensure that I am the only one who has permission to edit my team.

My model will be trained using pictures of myself (positives) and pictures of my leaguemates (negatives). All pictures obtained of my friends and families were done so with their permission, so I would like to thank them here for their cooperation in this project and for the success they have allowed me to have in fantasy football over the years. I want to ensure that this model works with a high degree of success before implementing it, so I have set the following goals:

- False Positive Rate <= 2%
- Recall >= 90%
- Less than a second for verification

Data and Methodology:

The data used in this project consisted of 55 pictures of myself (positives used) and 45 pictures of my fantasy football leaguemates (negatives used). These pictures contained diverse settings, lighting, camera angles, facial expression, and style features such as glasses, hoods, and facial hair. These photos were selected to help my model generalize, meaning it can still identify my facial structure in many different real world scenarios. Given that I will be using my fantasy football app in diverse settings, this is a crucial step.

The photos were placed in training (31 positive; 24 negative), validation (11 positive; 10 negative), and test sets (12 positive; 11 negative). I also ensured that each of these sets had a fairly even distribution of images from different settings. In order to preprocess the images, I used MTCNN, or Multi-task Cascaded Convolutional Neural Network. MTCNN uses a cascading series of neural networks for face detection and facial landmark recognition. This will ensure that the images are in a consistent format and in the best form possible for our models.

Now that my images have been preprocessed, I am now prepared to begin building my models. I will be using the following approaches to predict whether an image is negative or positive:

- **Pretrained Embeddings + Baseline** uses the pretrained FaceNet model from pytorch, which extracts face embeddings , or numerical representations of faces. A reference embedding is calculated, which is the average embedding of all positive images. Cosine similarity is then used to determine how similar a new image is to the reference embedding, and classified according to a selected threshold.
- **CNN (Convolutional Neural Network)** builds and trains a custom neural network from my dataset rather than using a pretrained model. Each image is put through a CNN with convolution, pooling, and dense layers to identify discriminative patterns associated with each class. The final layer uses a sigmoid activation to output a probably between 0 and 1, which is then used for binary classification.

Results and Analysis:

Pretrained Embeddings + Baseline:

Class	Precision	Recall	F1 Score
Negative	0.636	0.700	0.667
Positive	0.700	0.636	0.667

CNN:

Class	Precision	Recall	F1 Score
Negative	0.830	0.500	0.620
Positive	0.670	0.910	0.770

The results from the two used methods show very interesting trends. The baseline method seemed to perform much better in identifying negative images, while the CNN performed much better in identifying positives but frequently missed on negatives. These recall results are crucial in model selection, as I would prefer my model to incorrectly lock me out rather than let a potential intruder have access to my lineup. While the baseline model is more likely to misidentify myself as an intruder, it also is more likely to correctly keep out an intruder. CNN has a very high likelihood of identifying me, but also much more likely to have security concerns.

Model Recommendation:

Given these results, I feel that moving forward with the pretrained embeddings + baseline model would be the best option, as unsuccessfully identifying intruders has much greater consequences than temporarily locking myself out, as I could probably just try scanning myself again. I used this model on my test set, and received the following results:

Class	Precision	Recall	F1 Score
Negative	0.600	0.818	0.692
Positive	0.750	0.500	0.600

Conclusion:

While I certainly overestimated my model's potential performances in selecting performance goals, I feel good about my model selection. My selected model had a false positive rate of 0.182, which is well short of my goal of 2% but still very solid for the results seen. I also set a goal of a recall score of greater than 90%, which was nearly accomplished for negative images and greatly missed on positive images. My model also performs as quickly as I would've hoped, which is also great.

In exploring future iterations of this model, I believe the key to unlocking better results is more data to train, validate, and test on. I was very limited in pictures I was able to gather of myself and leaguemates, so I had to make due with what I was given. If more data were used in future iterations, I believe the predictive power of the model will also increase. For now, I suppose I will need to keep beating my friends and family in fantasy football the old-fashioned way.