The Eyes of the Machine: A Journey into Face Detection

(Project ID: 41)

Submitted by:
Patel Subodh Shailendra
Roll No: 24B2509
Department of MEMS, IIT Bombay

Date: July 31, 2025

Introduction

This report marks the completion of my SOC 2025 project, titled **"The Eyes of the Machine: A Journey into Face Detection."** The project's aim was to build a functional face recognition system using a Siamese Neural Network, with all the implementation done in Python, leveraging libraries like TensorFlow and OpenCV.

The journey started from absolute basics—understanding Python, learning the essentials of machine learning, and gradually working towards deep learning concepts. Along the way, I got hands-on with core libraries (NumPy, Pandas, Matplotlib), explored the foundations of linear regression, and then dived deep into neural networks, particularly the Siamese Neural Network. Eventually, I managed to build a working face recognition model, learning a ton from all the successes, mistakes, and technical hiccups.

This report is a step-by-step story of everything I did, learned, and struggled with throughout this project. I'll talk about the tools and methods I used, the problems I ran into (and how I fixed them), and my honest reflections on the process. Hopefully, anyone reading this—especially those just starting out—will find it relatable and maybe even useful!

1. Getting Comfortable with Python & Machine Learning Foundations

When I first started this project, I was honestly not very confident in Python. So, I decided to go through the **"100 Days of Code: Python"** playlist by CodeWithHarry, which was highly recommended by mentors. I completed the first 70 lectures, and it was a huge help—not just for learning syntax, but for actually building the habit of coding every day.

Some of the key topics I picked up:

- Basics: variables, loops, if-else, functions, etc.
- File handling and string formatting
- Object-Oriented Programming (classes, objects, inheritance)
- Modules and virtual environments
- Exception handling

This wasn't just "theory." I wrote tons of small scripts and tried out new things, sometimes just for fun—like making a simple file writer, or a class to introduce myself (literally, I coded a "Student" class saying "I am Subodh from MEMS branch" just for practice).

Why does this matter? In machine learning, knowing how to write, read, and tweak code is a superpower. I kept all my scripts organized in folders (`python_basics/`), which made it easier to come back and check what I'd learned.

Learning Machine Learning (Theory)

After getting comfortable with Python, I started exploring what machine learning actually is. My first exposure was to the difference between **supervised** and **unsupervised**

learning. I learned that in supervised learning, you have "answers" (labels) for your data, while in unsupervised learning, you don't—so the model has to find patterns on its own.

I also watched videos about real-world applications and why machine learning is important—from recommending movies to classifying images, and so on. At this point, I felt like I was finally getting a sense of how programming and data can come together to solve cool problems.

2. Diving into Core ML Libraries: NumPy, Pandas, Matplotlib & OpenCV

NumPy – The Language of Arrays

Once I got the basics of Python down, the next big thing was learning **NumPy**. Honestly, at first, all those arrays and matrix operations looked confusing, but after trying out a bunch of examples, things started making sense. NumPy is pretty much the backbone for all numerical work in ML—whether it's handling datasets, images, or building custom functions.

Some cool stuff I learned to do:

- Create and reshape arrays,
- Do matrix multiplication and dot products,
- Use aggregation functions (mean, sum, std, max, etc.),
- Slice and broadcast arrays (like adding a row vector to a column vector).

Example I tried:

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([[10], [20], [30]])
print("Broadcasted Result:\n", a + b)
```

At first, I just ran this to see what happens, and then realized how powerful broadcasting can be for ML tasks.

Pandas – Making Data Actually Usable

After NumPy, **Pandas** felt like a blessing. All those CSVs and data tables that used to look scary were suddenly easy to load, filter, and analyze.

Some things I practiced:

- Reading CSV and Excel files into DataFrames,
- Filtering data based on conditions,
- Using `.groupby()` and `.value_counts()` for quick stats,
- Cleaning and preparing data for ML models.

Simple example:

```
import pandas as pd
df = pd.DataFrame({'Name': ['A', 'B'], 'Age': [21, 19]})
print(df[df['Age'] > 20])
```

Matplotlib – Visualizing to Understand

It's one thing to look at numbers, but visuals make everything clearer. With **Matplotlib**, I started plotting simple line charts, bar plots, and scatter plots. Seeing my data visually helped me spot weird values, trends, and outliers.

A basic plot I made:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4]
y = [5, 8, 6, 9]
plt.plot(x, y, marker='o', linestyle='-', color='green')
plt.title("Performance Chart")
plt.xlabel("Days")
plt.ylabel("Score")
plt.grid(True)
plt.show()
```

OpenCV – Bringing Images to Life

I also started playing with **OpenCV** for image processing. Just loading an image, converting it to grayscale, or displaying it using OpenCV made me realize how this tool forms the backbone of any computer vision project.

The syntax is a bit different, but after a few YouTube tutorials and a lot of Googling, I was able to process and display images in my own scripts.

Overall

Learning these libraries felt like assembling a toolkit. Once I got the hang of these, moving on to actual ML models (like Linear Regression and Neural Networks) became way less intimidating.

3. Linear Regression: My First Real ML Model

After getting comfy with Python and those ML libraries, I finally took the plunge into building an actual ML model—**Linear Regression**.

To be honest, at first, the name itself sounded like some big deal. But after watching a few videos and messing around with code, I realized it's basically about drawing the "best fit line" through some data points. Nothing too crazy!

What I Did

- Watched some quick tutorials on what linear regression actually is (lots of people using house prices as examples).
- Downloaded the assignment files our mentors gave us.
- Opened Jupyter Notebook (after finally getting Anaconda to work, which took a few tries ngl).
- Tried loading the training data and test data into Pandas.

The Assignment (a.k.a. the "Let's See if You Actually Understood Anything" Test)

At first, I was lost about how to connect all the code pieces. But step by step, I did things like:

- Loaded the data and checked how it looks (and yeah, some columns were confusing).
- Tried plotting the data to just "see" what's going on.
- Wrote the formula for the cost function (basically, how "wrong" your line is).
- Used gradient descent (which, in my head, is just "keep adjusting till you're as close to right as possible").

Did I make mistakes? **Yes, a lot.**

- Forgot to scale features at first, so my line was totally off.
- Missed a bracket in my formula once, and got the weirdest results.
- Didn't choose the right learning rate initially, so my training was either too slow or the cost was jumping around like crazy.

What I Learned

- Linear regression is actually pretty simple once you "see" what's happening.
- Plotting graphs is not just for marks—it actually saves time spotting errors.
- You don't need to memorize formulas, just understand what you're trying to do: fit a line so your predictions are close to actual values.
- And yes, sometimes it's totally fine to Google "why is my linear regression not working in Jupyter."

Doing this assignment made me less scared of machine learning. I felt like, "Okay, I can do this." It set the stage for moving to more complex stuff, like neural networks.

4. Siamese Neural Network: The Deep Dive (And the Struggles)

So, after surviving linear regression, the next level was **Siamese Neural Networks**. When I first heard the name, it sounded like some sci-fi robot—but it's actually just a type of neural network that compares two things and tells you how similar they are.

What's the Point?

The main goal here: If you give it two face images, the network figures out if they're of the same person or not. Cool, right? Basically, the computer learns to "spot the difference" (or, more accurately, the similarity).

How I Went About It

- Read the mentor PDFs (at least twice, because the first time, not much made sense).
- Watched a bunch of YouTube videos.
- Tried to understand the architecture. Turns out, it's just two identical neural networks (literally like twins/Siamese twins, hence the name) that process two images and spit out numbers to compare.

Coding the Base Model

This was a struggle, not gonna lie.

- First, I copied the basic code from resources (thanks mentors), then tried to understand each line.
- Had to learn what `Conv2D`, `MaxPooling`, `Flatten`, and `Dense` actually do.
- Most of my code was written in Jupyter Notebook, which crashed on me twice while running heavy stuff.

Mistakes & Funny Moments

- Forgot to match the input shapes—got a big fat error.
- Spent hours debugging a typo ("input_shap" instead of "input_shape").
- The model would sometimes run but just output garbage—turns out, I messed up the data pairs.
- Lost track of which folder my dataset was in more times than I can count.

What I Actually Built

I eventually got the basic Siamese Network working:

- Took two images,
- Passed them through the network,
- Compared their output "vectors" (using something called Euclidean distance).

If the distance is low, the faces are probably of the same person. If it's high, they're different.

What I Learned

- Reading error messages is a skill.
- Even if you don't understand every line, keep moving and figure things out as you go.
- Sometimes, just restarting the notebook or your laptop solves weird bugs.
- Neural networks are not some "magic"—they're just maths + patience + lots of trial and error.

Siamese Networks are actually super useful for stuff like face recognition, signature verification, etc.

5. Face Recognition Model: Bringing It All Together

After all the practice with basic ML, coding, and building the Siamese Network, it was finally time for the main event: **face recognition**.

Honestly, I was both excited and nervous—because bringing everything together is where things usually break (at least for me).

How I Started

- Watched the recommended YouTube playlist on face recognition.
- Set up all the folders for my code, models, and images.
- Made sure TensorFlow, OpenCV, and all other libraries were installed (which took longer than coding itself because of compatibility issues).

Step-by-Step What I Did

1. Collected some sample images for training and testing (my own face, random faces from the internet, and a few dummy photos).
2. Used OpenCV to preprocess the images (resize, grayscale, etc.).
3. Created pairs for the Siamese Network—some pairs of the same person, some of different people.
4. Trained the model (basically running my images through the Siamese Network and letting it learn which faces are similar).
5. Tested the model by giving it two faces and checking if it guessed correctly.

Where Things Got Messy

- TensorFlow errors: Sometimes the model just wouldn't run because my versions of TensorFlow and Python weren't getting along.
- GPU not being detected: I was convinced my laptop was haunted. Turns out, I didn't have the right drivers.

- OpenCV image reading issues: Some images just refused to load. Discovered I was using the wrong file paths.
- Weird accuracy: At first, the model was getting everything wrong. Fixed it after realizing my training pairs were not shuffled properly.
- Random crashes: Jupyter Notebook would just freeze mid-training. Solved by splitting training into smaller chunks and saving progress regularly.

What Worked

- Once the setup was finally right, the face recognition model actually started predicting correctly for most cases.
- Tweaked some thresholds for "distance" between face encodings to get fewer false positives.
- The model wasn't perfect (definitely not ready for a security system), but it worked decently for a student project.

What I Learned

- Patience. Most time goes into fixing "why isn't this running?" and not actual coding.
- Always check your data and labels, or you'll waste hours with useless results.
- A working model—even if not perfect—feels super satisfying after so many bugs and errors.

Building a face recognition model from scratch was easily the most challenging and rewarding part of the project. I learned way more from what went wrong than what went right.

6. Problems, Mistakes & How I Solved Them

Honestly, this project wasn't just about learning fancy ML models. It was more about learning how to deal with stuff going wrong—which happened a lot. Here are some of the main struggles I faced (and how I somehow survived):

1. TensorFlow Installation Nightmares

If I had a rupee for every time TensorFlow refused to install or work properly, I'd be rich. Sometimes it was a Python version mismatch. Sometimes the pip install just didn't work, or I'd get random errors halfway through.

How I solved it: Lots of Googling, asking ChatGPT (not kidding), trying different versions, uninstalling, reinstalling... The key lesson: Never blindly follow old Stack Overflow threads. Eventually, I figured out what works for my setup, but it took many failed attempts.

2. GPU Issues

There's nothing more annoying than knowing your laptop has a GPU, but TensorFlow refuses to use it. Tried all sorts of commands to check if my GPU was available. Installed and reinstalled NVIDIA drivers multiple times. At one point, I even thought my GPU was broken.

How I solved it: Turns out, I needed the exact right combination of drivers, CUDA, and TensorFlow versions. I followed official guides (which felt more complicated than the ML project itself), and finally got it working. Pro tip: If CPU is working, don't stress too much unless your dataset is massive.

3. Jupyter Notebook Drama

Jupyter Notebook is amazing, until it randomly freezes or crashes when you run heavy models. Lost work a couple of times because I didn't save checkpoints. Accidentally ran an infinite loop once—laptop sounded like a helicopter taking off.

How I solved it:

- Started saving my work constantly (`Ctrl+S` is my new best friend).
- Broke big training tasks into smaller chunks.
- Restarted the kernel whenever things felt laggy.

4. Dataset Confusion

I downloaded the same dataset twice, in different folders, and kept wondering why my model was giving weird results. Sometimes, my image paths were just wrong, and OpenCV would silently fail.

How I solved it:

- Made a proper folder structure and stuck to it.
- Double-checked paths before running code.
- Added print statements everywhere to see what files were actually being loaded.

5. Code Typos & Debugging

Some mistakes were just classic typos—like missing an underscore or adding an extra bracket. Other times, I'd just copy code from somewhere and forget to change a variable name.

How I solved it:

- Slowed down and read error messages carefully.
- Started writing smaller functions instead of giant blocks of code.
- Took breaks when stuck (some bugs magically disappear when you return after a walk).

6. Model Not Learning / Bad Accuracy

Sometimes my model's predictions were just... terrible. Either it guessed everything as "same person" or "different person." Turns out, my training data wasn't balanced, or I hadn't shuffled it well.

How I solved it:

- Shuffled training pairs before each epoch.
- Balanced positive and negative pairs as much as possible.
- Didn't panic—just kept tweaking things and re-running.

General Lessons Learned

- Google is your best friend.
- There's no shame in asking for help, whether it's on forums or from friends.
- Most problems are normal and happen to everyone, not just me.
- Don't quit after one error—it usually just means you're 10 minutes away from a breakthrough.

7. Reflections & Takeaways — How This Project Changed My Approach

Looking back, I can genuinely say this project was way more than just coding or building a machine learning model. It kind of forced me to face my own limits, learn new stuff on the fly, and get comfortable with being stuck (a lot).

Some Realizations

- **Being Confused is Normal:** I used to think everyone else just "got it" and I was the only one stuck. Turns out, everyone is lost at some point. The real skill is not freaking out, but finding a way forward.
- **Small Wins Matter:** Getting the first code cell to run without error, loading a dataset properly, or getting a half-decent prediction—each of these little "wins" kept me motivated.
- **Progress Over Perfection:** There's no perfect code or perfect solution. What matters is that I kept trying, made progress, and actually finished something.
- **Community & Resources Help:** Using YouTube playlists, mentor PDFs, forums, and even ChatGPT honestly made things a lot easier. You don't have to do everything alone.
- **Learning How to Learn:** I feel like now I'm better at picking up new topics, just because I got used to breaking down big scary things into smaller, doable steps.

If I Could Start Over...

- I'd spend more time planning my folder structure and keeping things organized from the start.
- Wouldn't be afraid to ask "dumb" questions earlier.
- Would remember to save and backup my work all the time (learned this the hard way!).

What I'm Proud Of

- I didn't quit, even when things were super confusing or when my model kept failing.
- Managed to get a face recognition system working, even if it's not perfect.
- Actually started enjoying debugging (never thought I'd say that).

This project changed the way I approach not just coding, but learning in general. I'm less afraid to make mistakes now, and more comfortable figuring things out as I go.

8. Conclusion

To sum it all up, working on this SOC project has been a proper roller-coaster. I started off not knowing much about Python, machine learning, or neural networks, but step by step, I figured things out.

There were days when things just worked, and days when nothing made sense. But honestly, that's what made it interesting. From learning the basics of linear regression to struggling (and finally succeeding) with a Siamese Neural Network for face recognition, I picked up a lot more than just coding skills.

Most importantly, I learned how to be patient with myself, to keep moving forward even when the solution isn't clear, and to appreciate the process (not just the end result). Now, looking back, I feel a lot more confident about taking on bigger, more complex projects in the future.

Thanks to everyone who helped out—mentors, friends, online forums, and of course, the occasional YouTube tutorial binge. If anyone's just starting out on their ML journey, I'd say: it's not about being a genius, it's just about sticking with it and enjoying the learning.

Acknowledgments

I would like to thank all the mentors and coordinators for their constant support and for sharing useful resources, as well as my friends and online communities who always came through whenever I felt stuck.