Why I chose this/ how can this be useful in the real world:

- Attendance for school, work, meetings, etc
- Authentication & Access control for device unlocking or banking/payments (AntiFraud)
- Border/ Immigration control
- Health Monitoring (Hospitals)
- Identity verification for Automated services (Atms)

Goals:

- Create Anti Spoofing/ Liveness Detection Face Recognition System
- Works on different devices (phone, tablets, laptops, computers)
- Fast Refresh Rate & Frame Rate
- Deployable in the Real World
- How to collect data and train for own custom data

IDE: Pycharm (python) Community 😭 CUDA 11.7

Packages:

- Cvzone
- Ultralytics → to train using own data
- Mediapipe
- `pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118`
  `NOTE: Remove torch from project settings`
  `Install on command prompt(vm), not local`

Step 0: What is AI?

Artificial Intelligence: Methods which mimic human behavior

Machine Learning: Non explicit programming, Machine learns rules instead of pre defining them

Ex. Define features of cup and book in numbers(feature extraction), machine learns difference

Classification Problem: Limited # classes, sort new data into one of these classes

Regression Problem: Continuous numbers/outputs from a couple inputs

ML Process:

Training	Testing
Data	Predict
Feature	
Training	
Model	

3 Main Categories:

Supervised	Unsupervised	Reinforcement
Labeled Data	Unlabelled Data	Best Policy
Linear Regression	K- Means	Q Learning
Logistic Regression	C- Means	Sarsa
Support Vector MAchine	HCA	
KNN	Apriori	
Neural Networks		

Deep Learning: Multiple artificial neural networks(deep neural network)

- Inspired by our brains' neural network
- Finds relevant features itself → trains itself to differentiate between

$$DL \subset ML \subset AI$$

Why Python?

- Vast library, readability/simplicity, AI/ML tools, easy integration, visualization, jupyter notebook
- Although C++ or Java may be better performance wise and R may be better for statistics and data analysis, Python is still a leading choice for majority AI and ML related tasks

Step 1: Testing and Familiarizing with Libraries

FaceDetector Library Testing:

- Very simple and straightforward.
- Added box to detected object.

YOLO Library Testing:

- Reinstall Pytorch to use GPU
  - Originally using CPU [10.89 fps on average]
  - Higher fps [31.24 fps on average] 👏 3x
- Add object class name, confidence score, fps log

Test 1 & 2 Review:

To Improve:

- Area of box not large enough/ Does not cover entire face
- Shouldn't collect data when face is blurry
- Inaccurate object type detection

Run copy and pasted code even if it was running correctly before.

Step 2: Main Code to Collect Data

Improvements:

- Create offset for display box to ensure it covers entire face
    - Using a ratio to help with different distances or face sizes
    - Capturing a bit of background might actually be good for the algorithm
- Handle Blurriness
    - Check blurriness of just the face, if focused then collect data, else don't
    - Only collect data above a certain confidence threshold (80% and up)
        - Ensure we are getting good data
        - Output/format data into YOLO format
- Normalized values
    - Prevent corrupt data error by ensuring the values cannot go above 1.
    - This is very important as you will not notice this until after collecting all your data!

Checking if the script is working:

How to Use:

1. Set classID in script. 0 if collecting fake faces. 1 if collecting real faces.
2. Run script
3. Display 'real' or 'fake' human faces in front of the webcam accordingly.
4. Stop script once desired data has been collected
5. Cut and paste jpeg & txt files from CollectData folder into either Real or Fake folder accordingly.

Tips for Collecting Data:

- Keep ratio of fake and real 1:1
- When collecting data, make sure that it only collects/detects either fake or real.
- Include different lighting, background, people, clothes, expression, condition, number of people per frame, saturation, clarity/blur, etc..
- Examples for Fake: advertisements, social media, tv, netflix, paper prints, drawings, sculptures, etc…
- Check and remove unwanted data prior to Step 5.

Cv2.laplacian →….

Step 3: Split Data Script

Shuffle Data: Ensure data is randomly distributed across train, validation and test sets.

Split & Distribute Data: Split data up into ratio (train = 0.7, validation = 0.2, test = 0.1)

Handle odd total images issue: Put remaining images into training, ensuring that all images are used.

Confirmation: Confirm total # of images and how they are split between training, validation and tests.

Create data.yaml: data.yaml file specifies paths for training, validation, and test images with # of classes and names.

Step 4: Training Data (Offline & Online)

Collect lots of data, for simplicity I collected roughly 10000 total

For offline use, make sure to create a copy of data.yaml(dataOffline.yaml) and change the path to absolute path to the SplitData folder. Remove "../" from train,val, and test path
Then change input for the training script to direct to the new dataOffline.yaml file.
Make sure to uninstall previous PyTorch and reinstall to use GPU instead of CPU for speed.

```
path: C:\Users\hobos\PycharmProjects\LivelinessDetector\Dataset\SplitData
train: train/images
val: val/images
test: test/images

nc: 2
names: ['fake', 'real']
```

Using data found online and deploying it for custom applications is not the best practice for real world applications. Learn how to deploy in the real world. Create something we can actually use.
Create your own custom dataset → deploy → see performance. This requires collecting our own data. Main script is collecting data. Automation techniques (split data ourselves).

Make a copy of the best.pt and store it in the models folder.

For the purposes of this project, I kept the trainer parameters fairly simple.
Remember to use "if __name__ == '__main__':" to prevent RuntimeError.

Make sure 0 images are corrupt in output(good annotation)

For Online Training, I used Google Collab.

Step 5: Testing/ Running Main Script

Steps I took:

1. Copy and pasted the code from yoloTest into main script
2. Load best training data instead of yolov8n to model variable
3. Update classNames to detect only “fake” and “real”
4. Run Test

Problem1: When running the main script using the training “best” data, nothing was being detected.
Solution1: After reviewing the code and training process, I found a couple potential issues.

1. Insufficient Training Data: The dataset is somewhat limited as for “real” faces, I’ve only included my own face. However, for “fake” faces I included multiple variety of faces.
   Fix: Enhance dataset with more varied images for “real” (diverse set of individuals)

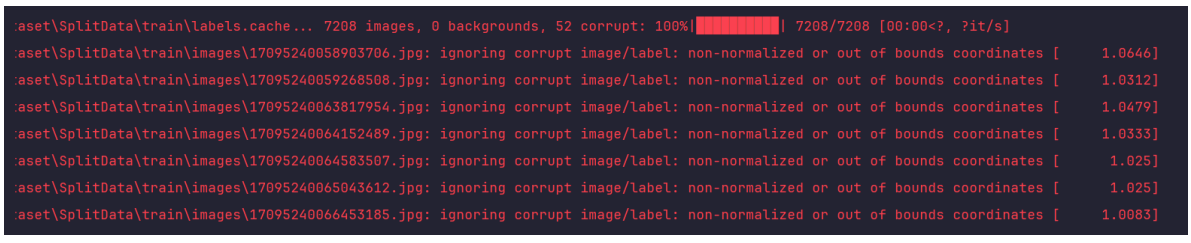
2. Overfitting: I trained the model with 3 epochs which may have lead to overfitting especially on a small dataset. This is concerning as the model complexity might be too high relative to the data variability.
   Fix: Increase # of epochs carefully. Monitor overfitting. Possibly implement dropout layers, regularization methods, and validate model performance via a separate validation set, however I want to keep this project simple.

3. Underfitting: Training with 3 epochs on a relatively small dataset might not have been enough for the model to learn sufficiently from the data. This is even more true for complex models and diverse datasets.
   Fix: Increase # of epochs. Monitor validation loss & training to ensure the model is learning effectively.

4. Incorrect Model Deployment: There is a chance the way the model is integrated into the main script has issues. This would lead to incorrect functionality.
   Fix: Verify model is loaded correctly and the data from the webcam is correctly fed to the model in the correct format.

5. Preprocess Inconsistency: Possible discrepancy in data preprocessing during training compared to live webcam feed, which can lead to face detection failures.
   Fix: Ensure preprocessing steps are consistent between the inference and training. This includes resizing, normalization, etc.

Problem 2: Corrupted data error during training process.



Solution 2: Revisited dataCollection.py, specifically check for correctness in normalization and handling edge cases regarding out of bounds. Pinpointed the issue is the handling when the box

is greater than 1. It incorrectly sets the values if there were to be values above 1 (Mismatched fields).

Step 6: User Friendly Update

Modifications to Main script:

- Correctly displays the confidence percentage instead of decimals
- Color code labels and box: Real(green) vs Fake(red)
- Slight adjustment to label scales and thickness

Limitations:

- Lack of diversity for 'real' data collection which possibly hinders accuracy
- Lack of variety in lighting especially in dimmer environments

Lessons:

1. **Balancing Data Diversity:** Having a diverse dataset is crucial. For my project, ensuring a balanced mix of real and fake faces under various conditions significantly improved my model's robustness and performance. This helped prevent bias and increase accuracy across different environments.
2. **Acknowledge Model Limitations:** The importance of understanding my model's limitations. Knowing these helped me set realistic expectations and plan for potential issues in high stakes applications and development.
3. **Strategizing for Deployment**: Transitioning from prototype to deployable system highlighted the importance of considering the target environment. I found that continuous monitoring post deployment is crucial in maintaining performance and adapting to change.
4. **Iterative Improvement:** AI Development is inherently iterative. Continuous feedback and model refinements were the key to improving accuracy and reliability, thus a need for adaptable development strategy.
5. **Ethical Considerations:** I became aware of ethical implications, particularly around privacy. In industry applications, it might be important to handle data respectfully and transparently.
6. **User-Centric Design:** Ensuring my system was user-friendly and met real world needs was a priority. Tailoring my project to enhance user experience, more intuitive and effective for end users.