

Building a Facial Recognition System

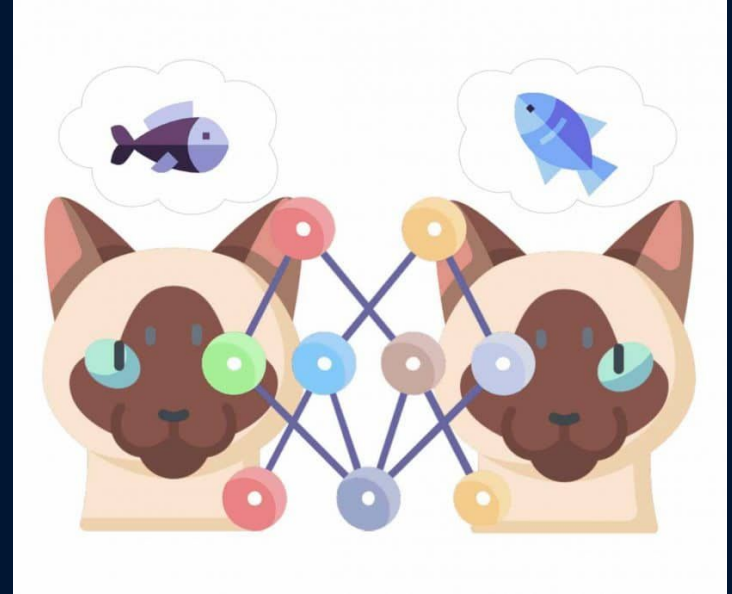
With a Siamese Network

Nathaniel Gilbert, Jeffrey Zheng, and Kristina Donders

Our Goal:

Create a Siamese network that can perform facial recognition.

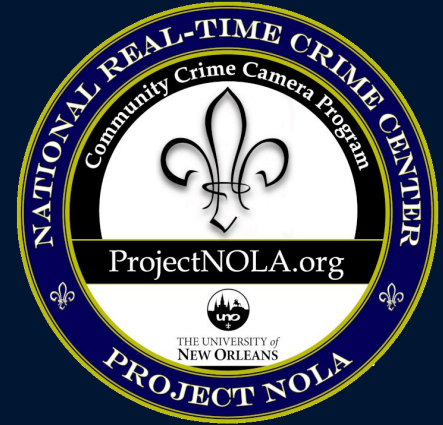
Our goal is to create a facial recognition system that can be used for surveillance -> to tell if a person of interest has been detected by the system.



AI for Police Surveillance:

Project NOLA:

- NOPD secretly installed AI-driven facial recognition systems
- Goal: monitor streets for suspects and wanted criminals
- In 2024, murder rates dropped 23% nationally and 39% in NOLA
- At times, police arrested suspects solely based on AI matches without additional evidence -> increased chance of false arrest
- Good in crowded places, but raises concerns about privacy
- Critics worry that this data will be misused for immigrant enforcement and put undocumented immigrants at risk



AI for Casino Surveillance:

OnWatch by Oosto:

- Face-based security software to help monitor gaming floors
- Goal: keep track of potential 'bad actors' in real time
- Allowed for better tracking of 'persons of interest' throughout venue without interfering with guests
- 3-5 potential detections have been made per day since system employed
- Makes players more comfortable



Ethics of AI Surveillance

- FRT enables real-time monitoring of people at scale, creating tension between security benefits and privacy rights
 - Surveillance is not inherently oppressive, but normalization without oversight risks abuse
- Many view FRT as invasive yet inevitable

Pros:

- Public safety
- Crime prevention
- Counterterrorism
- Humanitarian uses (locating missing kids, crisis response, in hospitals/schools for access control)

Cons:

- Privacy risks
- Lack of consent
- Unauthorized data collection
- Function creep
- Algorithmic bias- esp. misidentification of racial/ethnic minorities

Using a Siamese Network:

Problem: traditional CNN classifiers fail with unknown faces- always force a prediction, risking misidentification

Siamese approach: uses metric learning to compare faces by similarity instead of classifying into fixed categories

How it works: learns embeddings; low distance = same person & vice versa; uses a threshold to detect same and different people

Advantages:

- No retraining when people are added/ removed, just update the database
- Model stays constant (scalable)

Result: high performance model with **96.9% accuracy**

The Dataset:

We used the Labeled Faces in the Wild database on Kaggle, a database of face photographs designed to study the problem of unconstrained face recognition.

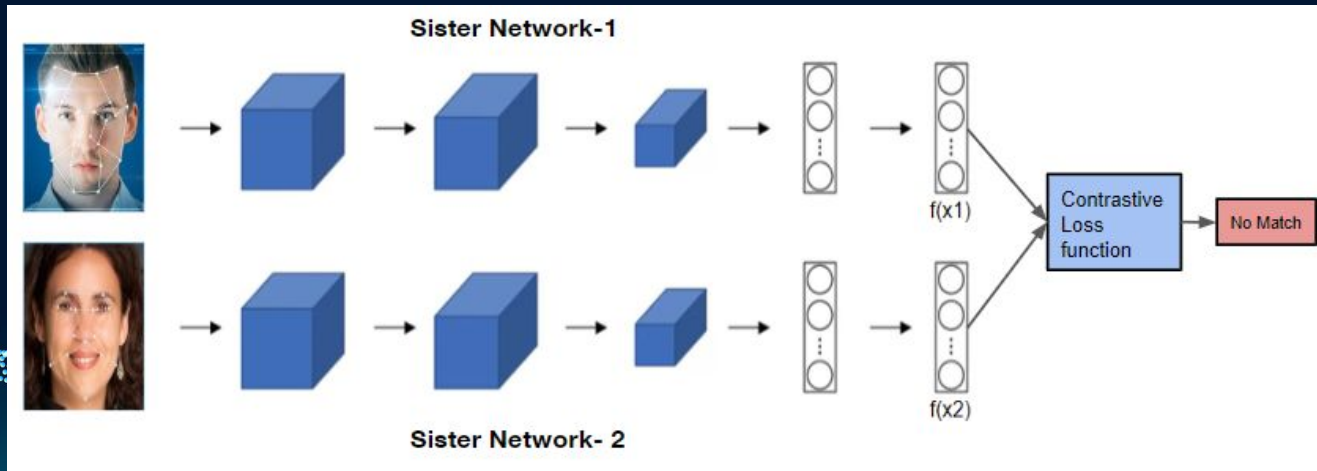
Contains over 13,000 images of faces collected from the internet.

1,680 of the people pictures have 2+ distinct photos in the dataset.



Model Architecture:

1. Two input images fed into two identical “sister” networks
2. Each image passes through an identical CNN to extract key features
3. Features are flattened and passed through fully connected layers
4. Each network outputs an embedding vector
5. Vectors are compared using contrastive loss function
6. Then the pair of images is labeled as same/different person



Pre-Processing & Image Augmentation:

Pre-processing:

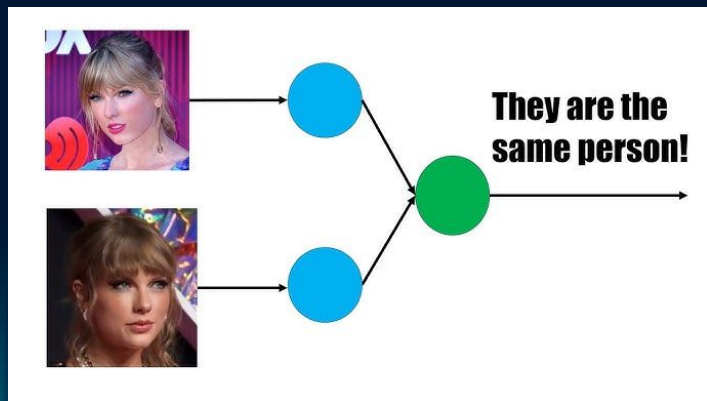
- Split into test and train datasets with a 20/80 split

Image Augmentation:

- Defined function that randomly changes image in a few ways so model can learn to handle variation
- Randomly flipped, blurred, and rotated images



Preparing Data for a Siamese Network:



Made a function that, when a dataset is inputted:

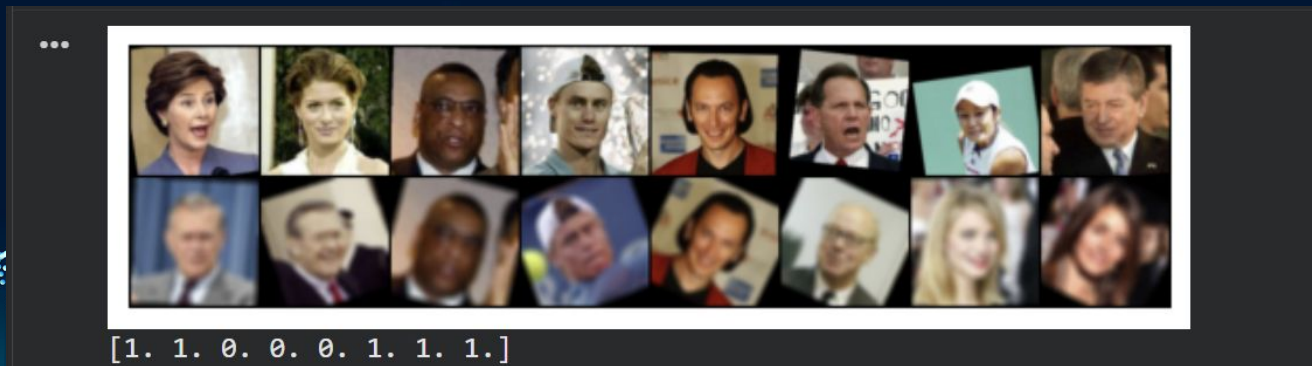
- Picks random image
- Randomly chooses a partner image
- Loads both images
- Applies augmentation
- Converts to tensors
- Returns (image1, image2, label)
 - 0: same person
 - 1: different person

Image Examples:

A DataLoader feeds the model mini-batches of paired images during training

- Takes siamese dataset as input
- Batch size of 8 image pairs
- Randomizes the order each epoch
- Uses 2 CPU threads for faster loading
- "1" indicates different people, 0 indicates the same person

```
vis_dataloader = DataLoader(siamese_dataset,  
                             shuffle=True,  
                             num_workers=2,  
                             batch_size=8)
```



Building the Siamese Network

Siamese NN has three parts:

- A CNN that turns each image into a feature vector
- A fully connected layer to compare the features
- A distance function that measures how different the two vectors are

Feature extractor:

- Extract important features like edges, textures, shapes, etc. to make compact feature map

Making embedding:

- Turns feature map into a final 2-D embedding
 - Measures how far apart those two embeddings are
 - Distance tells us whether the images are the same person



Training the Full Model

- We augment the training data to improve robustness of the model
- For the testing data we simply normalize it.
- We then create a training and testing siamese dataset and load them into a data loader.

```
transformation_train = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=10),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
    transforms.RandomGrayscale(p=0.1),
    transforms.GaussianBlur(kernel_size=3, sigma=(0.1, 2.0)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225]),
])

transformation_test=transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225]),
])
```

```
siamese_dataset = SiameseNetworkDataset(images=X_train,
                                         labels=y_train,
                                         transform=transformation_train)

siamese_dataset_test = SiameseNetworkDataset(images=X_test,
                                              labels=y_test,
                                              transform=transformation_test)

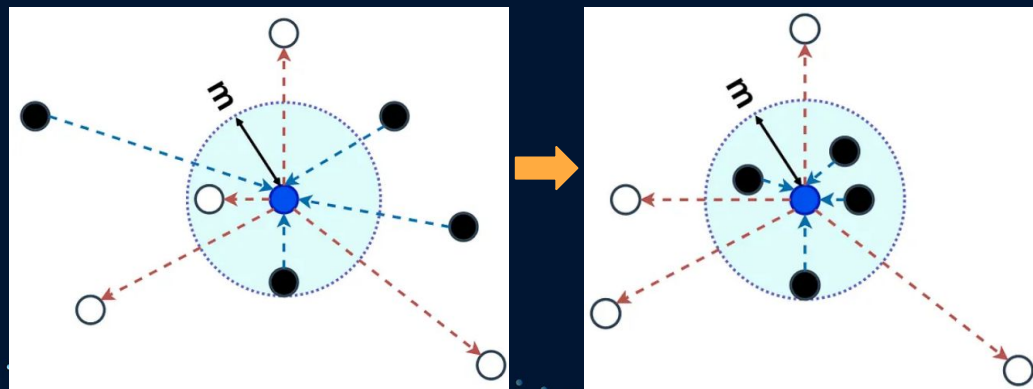
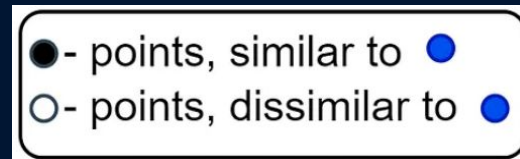
train_ds=siamese_dataset
val_ds=siamese_dataset_test

batch_size = 64

dls = DataLoaders.from_dsets([
    train_ds,
    val_ds,
    bs=batch_size,
    shuffle=True
])
```

Contrastive Loss Function

- Used Contrastive Loss Function:
 - Special loss function used in Siamese Network
 - Teaches SN's to tell whether two images are similar or different
 - Makes the network cluster same-people faces together and separate different-person faces



Accuracy Function

- Our model returns the distance between pairs, not an actual prediction. So we need to define an accuracy measure that creates a prediction.
- To do this we set a threshold, where if the distance is greater than the threshold for a given set of images it returns 1 (different people)
- To find the optimal threshold we test through a range of thresholds in a range and store the one with the best accuracy.
- In our case the best threshold was 3.6935

```
thresholds = np.linspace(0, 5, 200)
best_acc = 0
best_thresh = 0

for thresh in thresholds:
    preds = (all_outputs > thresh).astype(int)
    acc = (preds == all_labels).mean()
    if acc > best_acc:
        best_acc = acc
        best_thresh = thresh

print(f"Best threshold: {best_thresh:.4f}, Accuracy: {best_acc:.4f}")

Best threshold: 3.6935, Accuracy: 0.7707
```

Training the Model for 22 Epochs

- After 22 Epochs we achieved a 64.6% accuracy
- This can potentially be improved upon by running the model for more epochs

epoch	train_loss	valid_loss	siamese_acc	time
0	18.636961	15.788445	0.612391	02:32
1	18.713333	16.033604	0.647525	02:34
2	19.442451	16.460484	0.621836	02:31
3	20.069881	18.989304	0.599547	02:31
4	21.251053	18.658121	0.619569	02:30
5	21.526743	17.041140	0.601813	02:31
6	21.593229	19.121977	0.619192	02:30
7	20.870207	17.938406	0.578768	02:29
8	21.361769	17.013111	0.577635	02:29
9	20.543232	18.378038	0.622969	02:29
10	20.371809	16.072990	0.601436	02:28
11	20.971788	16.398510	0.588213	02:30
12	20.309021	16.956270	0.616169	02:28
13	20.464880	16.468258	0.576502	02:28
14	19.604908	15.835436	0.613525	02:29
15	18.596052	15.526560	0.613903	02:29
16	19.312525	15.425986	0.608236	02:27
17	18.341978	14.865148	0.596902	02:28
18	18.815538	14.794511	0.611636	02:29
19	17.868174	14.632793	0.639970	02:29
20	17.376966	14.026508	0.608614	02:29
21	17.400818	14.994331	0.646392	02:27

Comparison to a Baseline

In 'A Siamese-network-based Facial Recognition System' (2024), Chen et al. achieved a **96%** accuracy in their model.

Random people on Kaggle:

- 66.35%
- 75.78%
- 54.7%
- 70.31%

Our Model:
64.64%



Final Takeaway: Our model is solid.

... but one more thing

While building a Siamese Network facial recognition model was an interesting challenge, using systems like these in the real world demands caution.

Surveillance isn't inherently abusive, but history shows its closely tied to misuse, making ethical safeguards and mandated regulation essential.





**Any
Questions
?**