Face Classification

Jinyoon Kim, Aditya, Kenya, Peter, Sami

Background

Face classification is a subfield of computer vision that focuses on the identification and recognition of human faces. It has become an increasingly important area of research due to its wide range of practical applications, such as surveillance, security, and facial recognition technology.

Face classification research has been ongoing for several decades, with early approaches relying on manually designed feature extraction techniques, such as the extraction of edges, corners, and texture patterns. However, with the advent of deep learning techniques and the availability of large annotated datasets, deep neural networks have become the dominant approach for face classification.

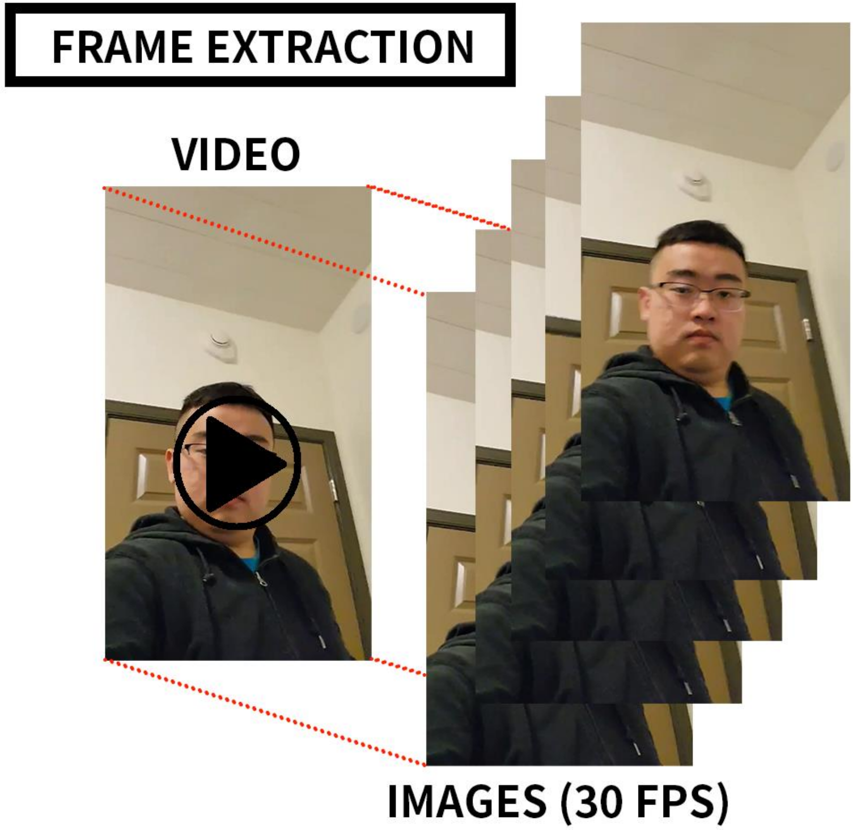
Methodology

1. **Acquire data**
    a. Two videos for each person, one indoors and one outdoors
2. **Preprocess the data for use by the model**
    a. Extract images from frames in the videos
    b. Augment those images using the techniques mentioned earlier to create variety for the model
    c. Transform the images into tensor objects, which is needed for the model to perform backpropagation
    d. Create a stratified split of the data
        i. Image set to get raw features
        ii. Training set
        iii. Testing set
3. **Initialize the resnet model**
    a. Model comes pre trained on a wide variety of images
    b. Use the images set aside to get 512 raw features
4. **Preprocess the features acquired from the resnet model using a correlation matrix**
    a. Reduce the number of features to the top 64 features with the highest correlation with class variable
5. **Perform training using CNN with backpropagation**
    a. Run through several epochs (10) until loss is minimized
        i. Diminishing returns after successive epochs
6. **Test the model**
7. **After step 4, initialize and train decision tree**
    a. Using gini gain, which works better than entropy when working with continuous values
8. **Test the decision tree model**

Dataset

To collect face images of our group member, we took videos of 30 seconds for each member in various backgrounds.

Using OpenCV library, we had extracted images from videos 30 frames per second and we could create about 1000+ raw images of each group member.

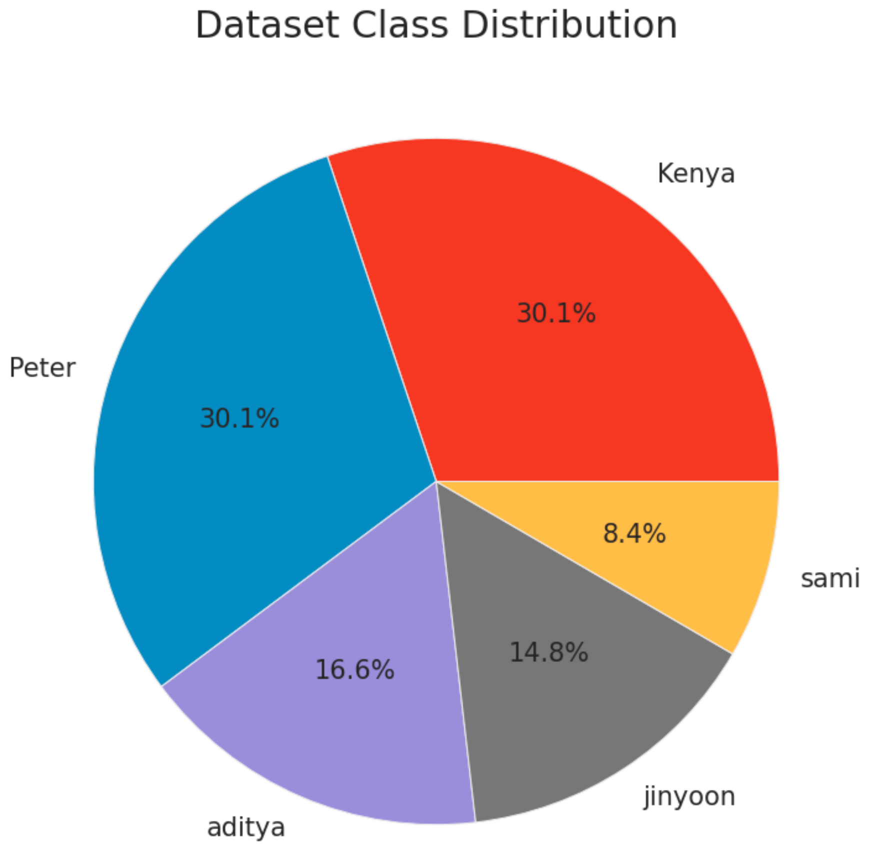


Dataset Distributions

The dataset distribution refers to the way data is distributed across different categories or classes in a dataset. The distribution of data can have a significant impact on the performance of deep learning models, as models may have a bias towards certain classes if they are over-represented in the dataset.

In this project, the dataset consists of images of individuals' faces, where each image is labeled with the name of the person in the image. This distribution shows that the dataset is relatively well balanced for some classes, with the exception of sami. This under-representation can cause a bias in model if not combated. This is important for ensuring that the model is not biased towards any specific individual and can generalize well to the faces in different circumstances.

To nullify the effect of the slight imbalance in the distributions, we used data augmentation.

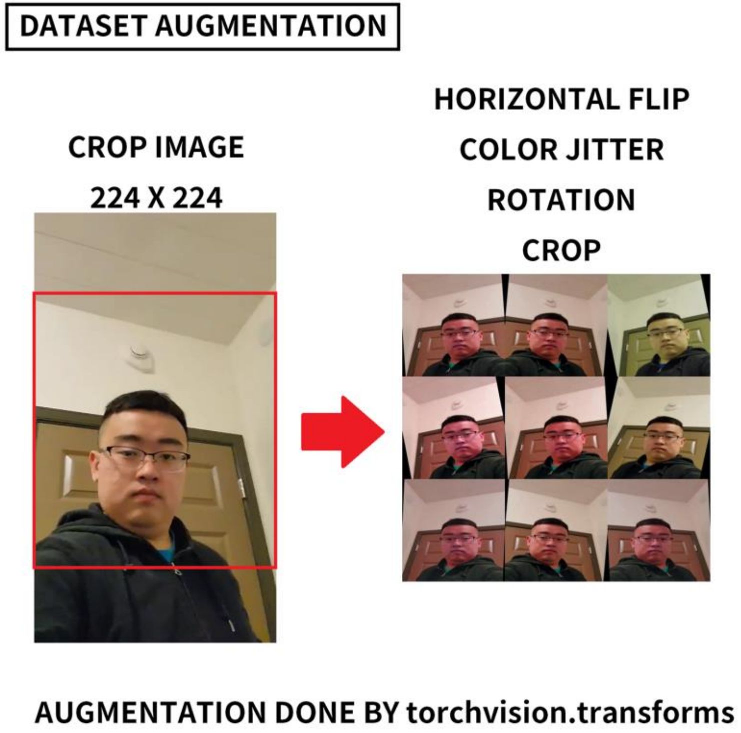


Augmentation

Data augmentation is a technique used in machine learning and deep learning, to increase the size and diversity of the training dataset by applying various transformations on the existing data which improves model's ability to generalize to new, unseen data.

Here we preprocessed images by cropping them into 224 X 224 size then applied various variations of augmentation for each image by using Pytorch transforms library.

After augmentation, we turn the images into the tensor object using PyTorch framework to leverage its optimized computational capabilities for efficient deep learning processing.



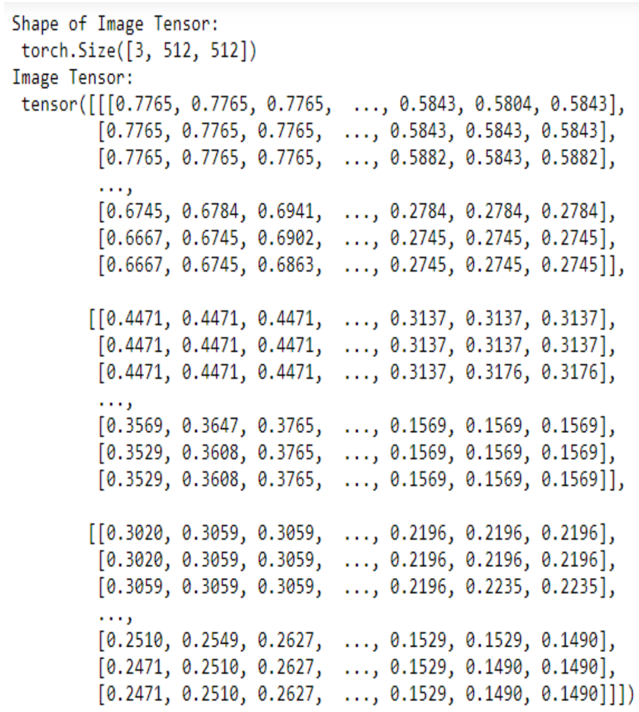
Tensor object and backpropagation

In PyTorch, a tensor is a multi-dimensional array that can store and process numerical data. Tensors are the fundamental building blocks of PyTorch and are used for computations in deep learning models. They have similar structure with arrays.

However, When you perform operations on tensors in PyTorch, the framework keeps track of the operations performed and creates a computational graph. This graph represents the operations as a directed acyclic graph (DAG), where each node represents an operation and edges represent the flow of data.

During backpropagation, the intermediate results stored during the forward pass are used to calculate the gradients of the model's parameters and update the weights accordingly.

We also normalize these tensors to improve its convergence and mitigate numerical instability.



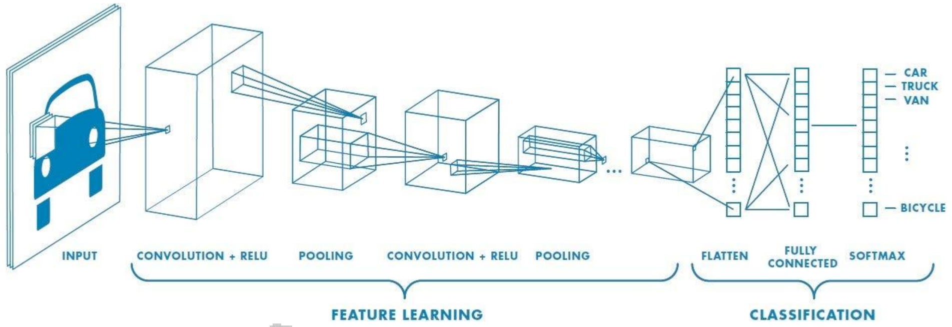
```
Shape of Image Tensor:
 torch.Size([3, 512, 512])
Image Tensor:
 tensor([[[0.7765, 0.7765, 0.7765,  ..., 0.5843, 0.5804, 0.5843],
         [0.7765, 0.7765, 0.7765,  ..., 0.5843, 0.5843, 0.5843],
         [0.7765, 0.7765, 0.7765,  ..., 0.5882, 0.5843, 0.5882],
         ...,
         [0.6745, 0.6784, 0.6941,  ..., 0.2784, 0.2784, 0.2784],
         [0.6667, 0.6745, 0.6902,  ..., 0.2745, 0.2745, 0.2745],
         [0.6667, 0.6745, 0.6863,  ..., 0.2745, 0.2745, 0.2745]],

        [[0.4471, 0.4471, 0.4471,  ..., 0.3137, 0.3137, 0.3137],
         [0.4471, 0.4471, 0.4471,  ..., 0.3137, 0.3137, 0.3137],
         [0.4471, 0.4471, 0.4471,  ..., 0.3137, 0.3176, 0.3176],
         ...,
         [0.3569, 0.3647, 0.3765,  ..., 0.1569, 0.1569, 0.1569],
         [0.3529, 0.3608, 0.3765,  ..., 0.1569, 0.1569, 0.1569],
         [0.3529, 0.3608, 0.3765,  ..., 0.1569, 0.1569, 0.1569]],

        [[0.3020, 0.3059, 0.3059,  ..., 0.2196, 0.2196, 0.2196],
         [0.3020, 0.3059, 0.3059,  ..., 0.2196, 0.2196, 0.2196],
         [0.3059, 0.3059, 0.3059,  ..., 0.2196, 0.2235, 0.2235],
         ...,
         [0.2510, 0.2549, 0.2627,  ..., 0.1529, 0.1529, 0.1490],
         [0.2471, 0.2510, 0.2627,  ..., 0.1529, 0.1490, 0.1490],
         [0.2471, 0.2510, 0.2627,  ..., 0.1529, 0.1490, 0.1490]]])
```

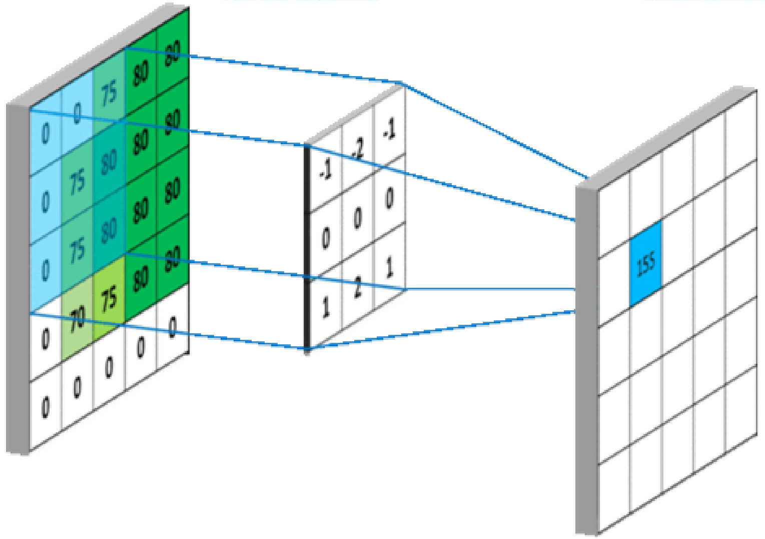
Algorithm: Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a type of deep learning algorithm commonly used for image classification and object recognition. The basic idea behind CNNs is to use layers of interconnected neurons that can automatically learn and identify features in images.

At a high level, a CNN consists of several layers that perform different functions. The input layer takes in the image data, which is then passed through a series of convolutional layers. These layers apply filters to the input data in order to detect features such as edges, corners, and textures.

During the training process, the weights of the filters in the convolutional layers are adjusted through a process of backpropagation, which involves calculating the gradient of the loss function with respect to the weights and updating them accordingly. This process continues until the network converges on a set of weights that minimizes the loss function.

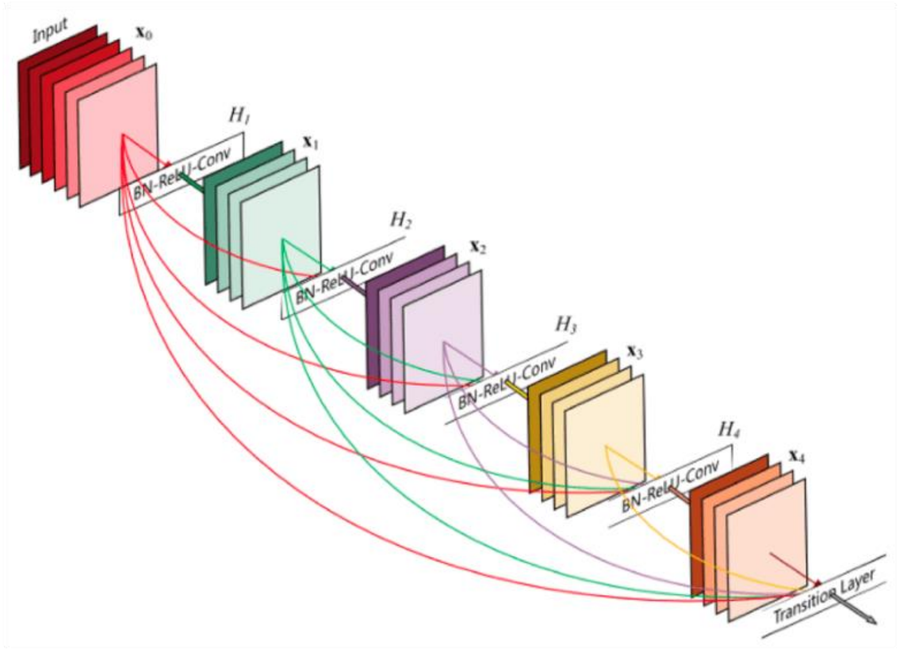




Algorithm: Residual Neural Network

Residual Neural Networks (ResNets) are a type of neural network architecture that address the problem of vanishing gradients in very deep neural networks. Vanishing gradients can occur in deep networks when the gradient signal becomes too small as it propagates backwards through the network during training, which can result in slower convergence or even complete failure to converge.

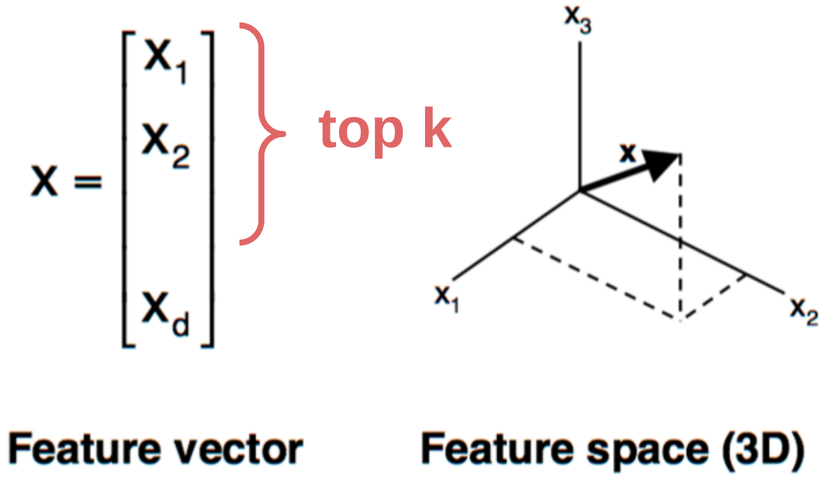
The basic idea behind ResNets is to introduce skip connections, also known as residual connections, that allow the network to bypass one or more layers and directly propagate information from earlier layers to later layers. This makes it easier for the gradient signal to flow backwards through the network during training, which can help to mitigate the vanishing gradients problem and allow for deeper networks to be trained more effectively.



Algorithm: Top k Features

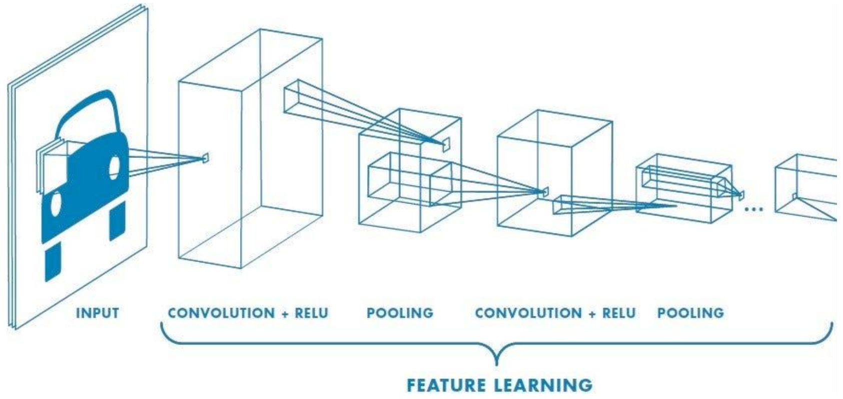
We also created an algorithm to extract the top k features from a feature vector by selecting a subset of k features from the original set of features that are most informative or most correlated with the target variable. We did this to reduce the computational complexity of a machine learning algorithm and to improve its performance by removing irrelevant or redundant features.

The algorithm is a correlation-based feature selection method. It first **calculates the correlation matrix** of the input data and then selects the top k features that have the **highest correlation values** with each other. Specifically, it creates a mask of ones for the lower triangle of the correlation matrix and sets the upper triangle to NaN values to avoid selecting the same feature twice. Then it sorts the resulting correlation values in **descending order and selects the top k features** based on the sorted correlation values.

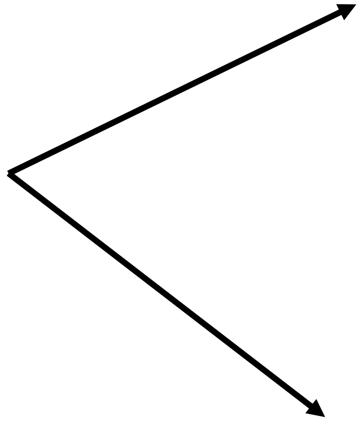


$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_d \end{bmatrix}$
top k
x_3
x
x_1
x_2
Feature vector
Feature space (3D)

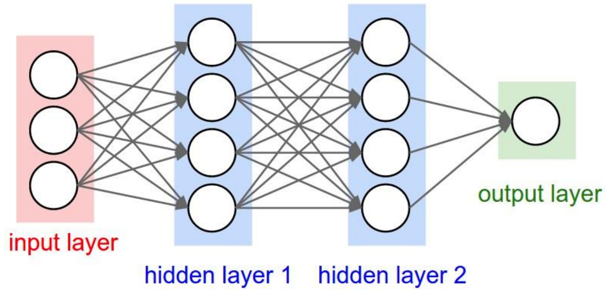
Algorithm: Neural Network & Decision Tree



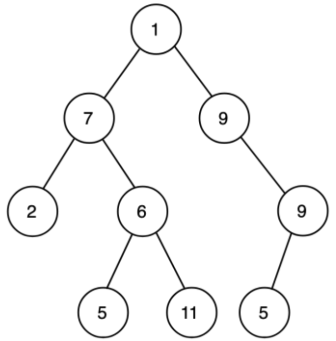
Convolutional Neural Network (CNN)



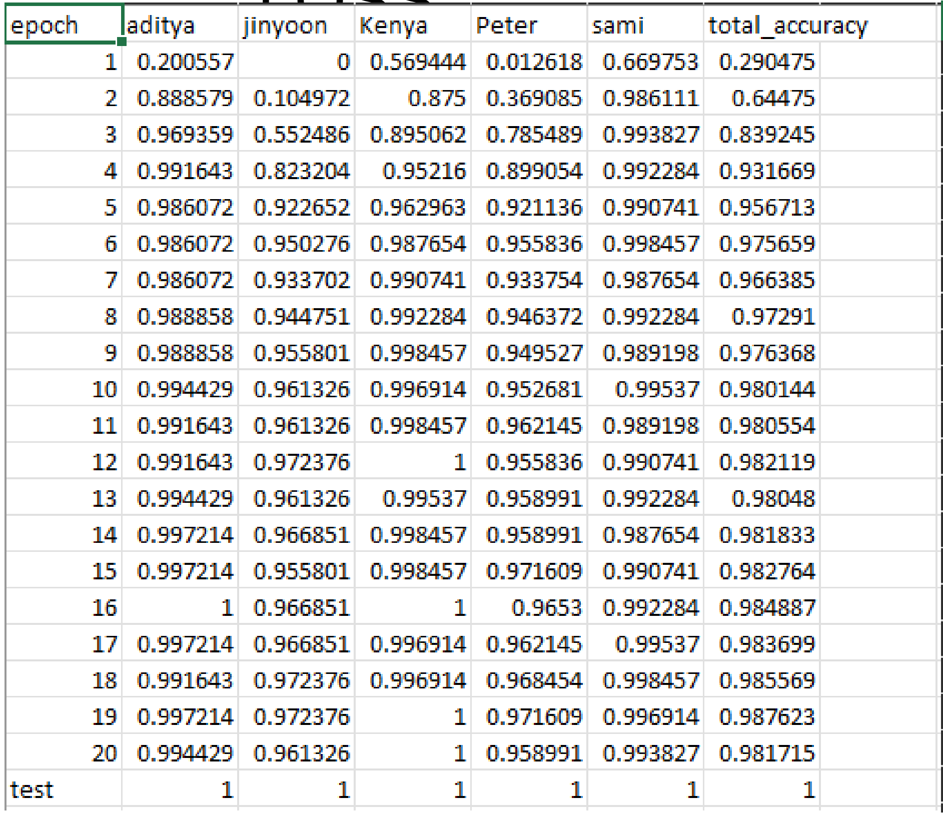
Neural Network (NN)



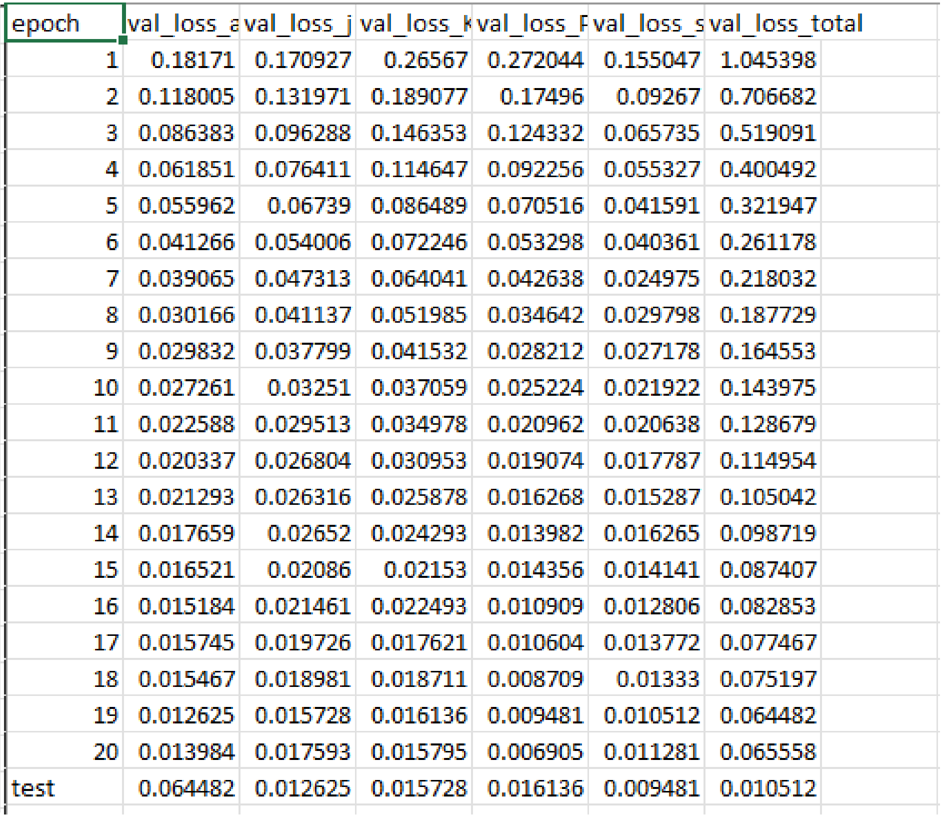
Decision Tree



ACCURACY
LOSS



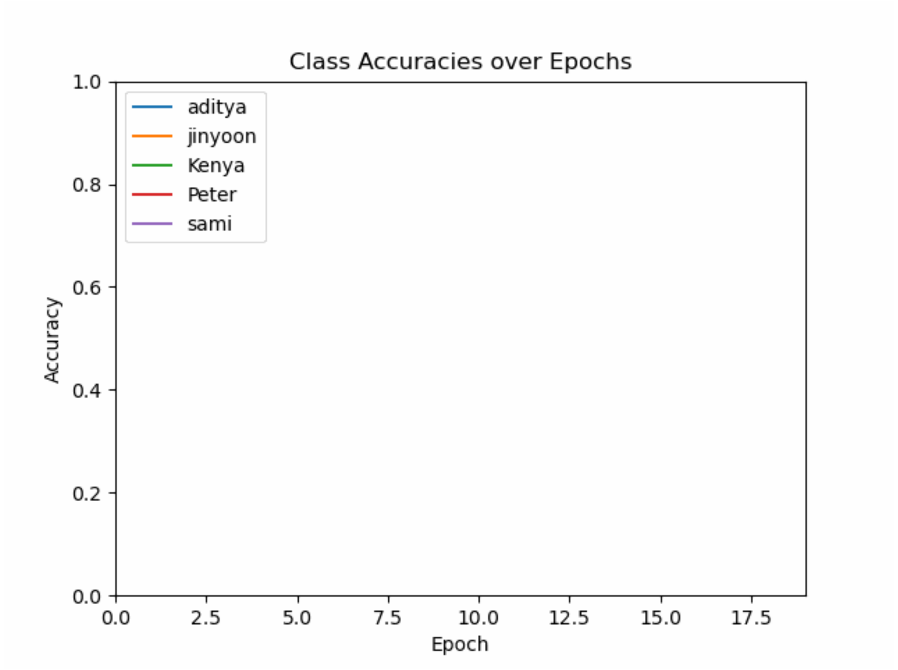
epoch	aditya	jinyoon	Kenya	Peter	sami	total_accuracy
1	0.200557	0	0.569444	0.012618	0.669753	0.290475
2	0.888579	0.104972	0.875	0.369085	0.986111	0.64475
3	0.969359	0.552486	0.895062	0.785489	0.993827	0.839245
4	0.991643	0.823204	0.95216	0.899054	0.992284	0.931669
5	0.986072	0.922652	0.962963	0.921136	0.990741	0.956713
6	0.986072	0.950276	0.987654	0.955836	0.998457	0.975659
7	0.986072	0.933702	0.990741	0.933754	0.987654	0.966385
8	0.988858	0.944751	0.992284	0.946372	0.992284	0.97291
9	0.988858	0.955801	0.998457	0.949527	0.989198	0.976368
10	0.994429	0.961326	0.996914	0.952681	0.99537	0.980144
11	0.991643	0.961326	0.998457	0.962145	0.989198	0.980554
12	0.991643	0.972376	1	0.955836	0.990741	0.982119
13	0.994429	0.961326	0.99537	0.958991	0.992284	0.98048
14	0.997214	0.966851	0.998457	0.958991	0.987654	0.981833
15	0.997214	0.955801	0.998457	0.971609	0.990741	0.982764
16	1	0.966851	1	0.9653	0.992284	0.984887
17	0.997214	0.966851	0.996914	0.962145	0.99537	0.983699
18	0.991643	0.972376	0.996914	0.968454	0.998457	0.985569
19	0.997214	0.972376	1	0.971609	0.996914	0.987623
20	0.994429	0.961326	1	0.958991	0.993827	0.981715
test	1	1	1	1	1	1

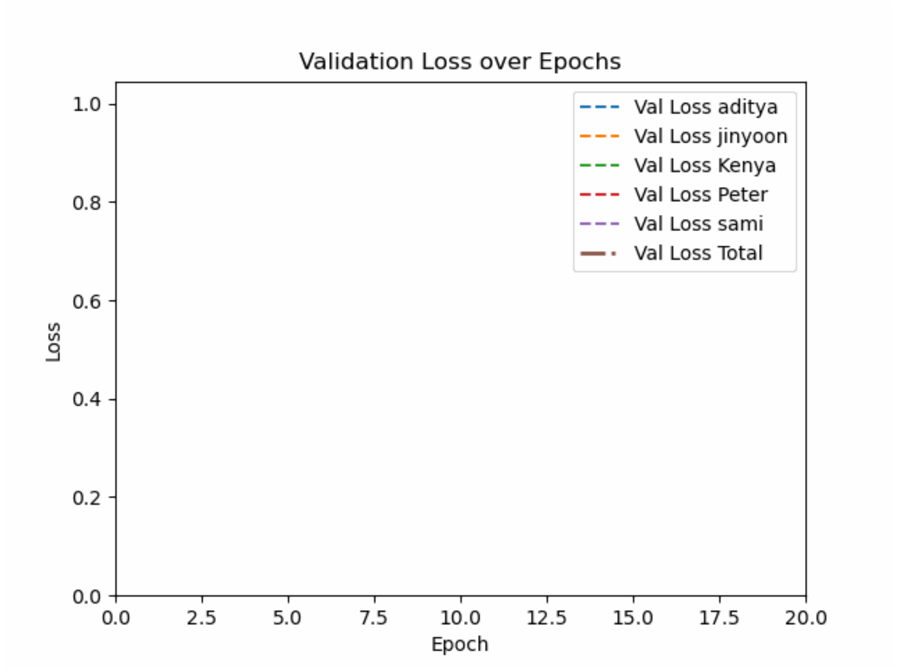


epoch	val_loss_a	val_loss_j	val_loss_k	val_loss_P	val_loss_s	val_loss_total
1	0.18171	0.170927	0.26567	0.272044	0.155047	1.045398
2	0.118005	0.131971	0.189077	0.17496	0.09267	0.706682
3	0.086383	0.096288	0.146353	0.124332	0.065735	0.519091
4	0.061851	0.076411	0.114647	0.092256	0.055327	0.400492
5	0.055962	0.06739	0.086489	0.070516	0.041591	0.321947
6	0.041266	0.054006	0.072246	0.053298	0.040361	0.261178
7	0.039065	0.047313	0.064041	0.042638	0.024975	0.218032
8	0.030166	0.041137	0.051985	0.034642	0.029798	0.187729
9	0.029832	0.037799	0.041532	0.028212	0.027178	0.164553
10	0.027261	0.03251	0.037059	0.025224	0.021922	0.143975
11	0.022588	0.029513	0.034978	0.020962	0.020638	0.128679
12	0.020337	0.026804	0.030953	0.019074	0.017787	0.114954
13	0.021293	0.026316	0.025878	0.016268	0.015287	0.105042
14	0.017659	0.02652	0.024293	0.013982	0.016265	0.098719
15	0.016521	0.02086	0.02153	0.014356	0.014141	0.087407
16	0.015184	0.021461	0.022493	0.010909	0.012806	0.082853
17	0.015745	0.019726	0.017621	0.010604	0.013772	0.077467
18	0.015467	0.018981	0.018711	0.008709	0.01333	0.075197
19	0.012625	0.015728	0.016136	0.009481	0.010512	0.064482
20	0.013984	0.017593	0.015795	0.006905	0.011281	0.065558
test	0.064482	0.012625	0.015728	0.016136	0.009481	0.010512

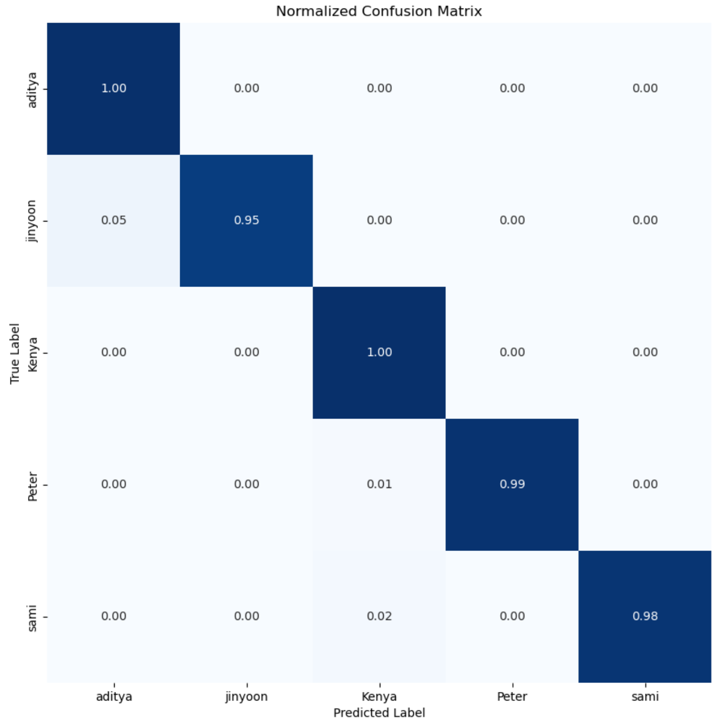
Plotting graph

This is the visualized animation of training process, we can see that accuracy rapidly increases over the start of the training epochs, but gradually decreases acceleration over time.





Results: Neural Network Confusion matrix (10 epochs)



	aditya	jinyoon	Kenya	Peter	sami
aditya	1.00	0.00	0.00	0.00	0.00
jinyoon	0.05	0.95	0.00	0.00	0.00
Kenya	0.00	0.00	1.00	0.00	0.00
Peter	0.00	0.00	0.01	0.99	0.00
sami	0.00	0.00	0.02	0.00	0.98

Confusion_matrix[a][j] shows the percentage of times class ‘j’(column) was predicted when the true class was ‘a’(row)

Accuracy(Aditya) = 1

Accuracy(Jinyoon) = 0.95

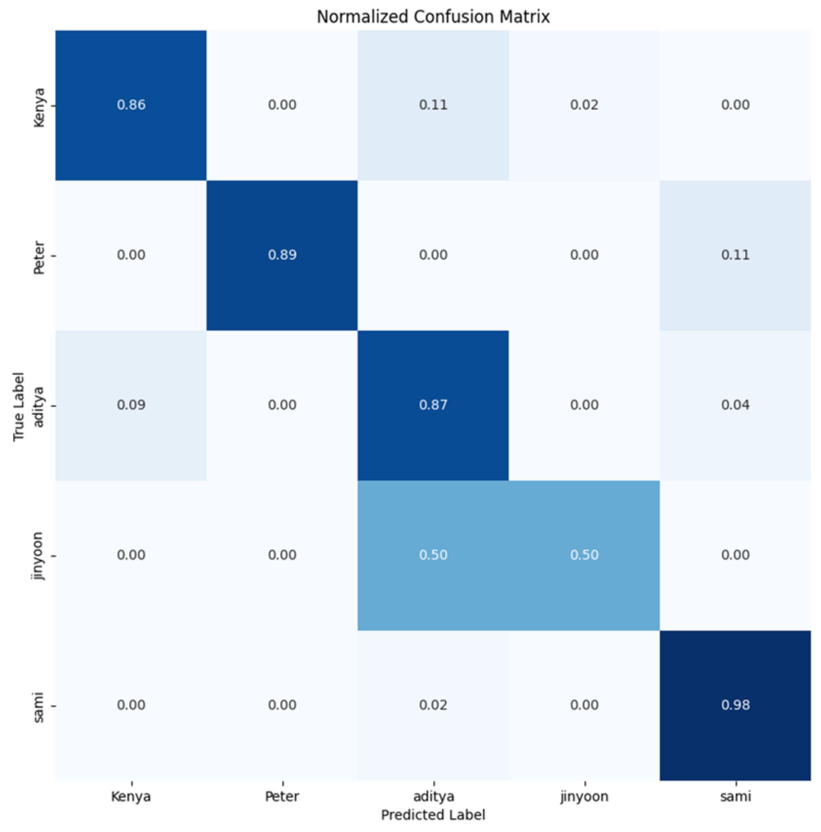
Accuracy(Kenya) = 1.00

Accuracy(Peter) = 0.99

Accuracy(Sami) = 0.99

AccuracyTotal = 0.98

Results: Decision Tree



True \ Predicted	Kenya	Peter	aditya	jinyoon	sami
Kenya	0.86	0.00	0.11	0.02	0.00
Peter	0.00	0.89	0.00	0.00	0.11
aditya	0.09	0.00	0.87	0.00	0.04
jinyoon	0.00	0.00	0.50	0.50	0.00
sami	0.00	0.00	0.02	0.00	0.98

Decision tree model produces less accurate results than the CNN, but still produces favorable results.

Accuracy(Aditya) = 0.85

Accuracy(Jinyoon) = 0.89

Accuracy(Kenya) = 0.96

Accuracy(Peter) = 0.98

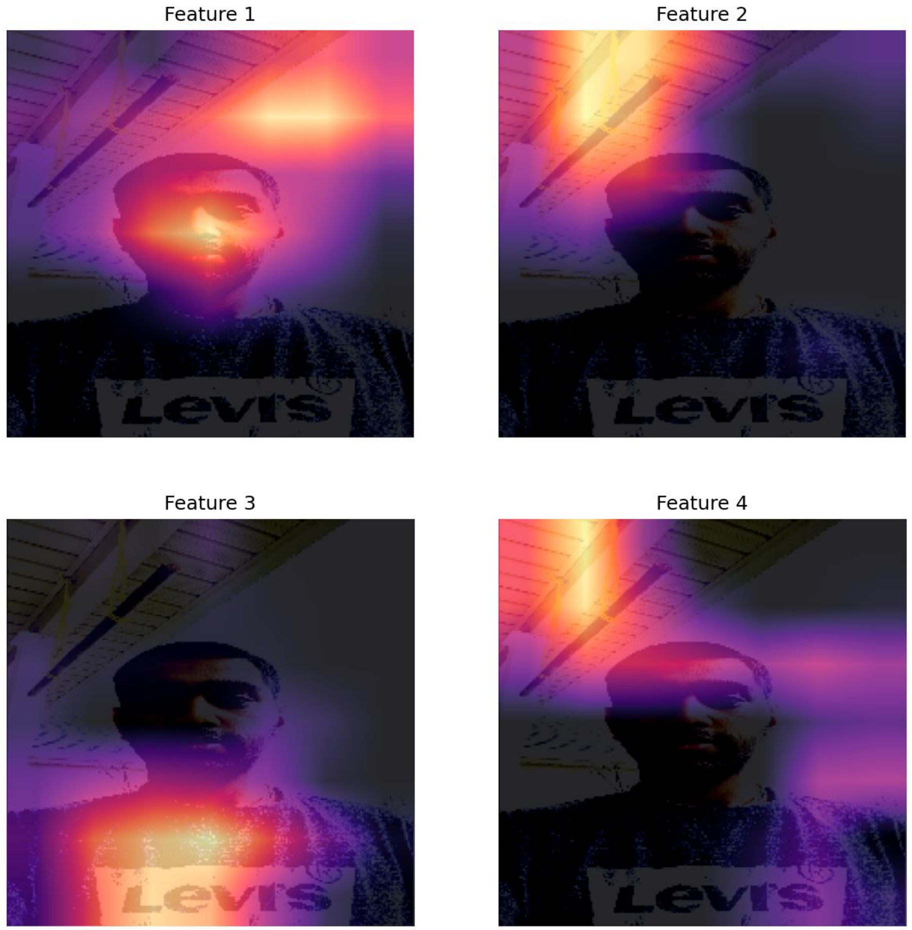
Accuracy(Sami) = 0.97

AccuracyTotal = 0.93

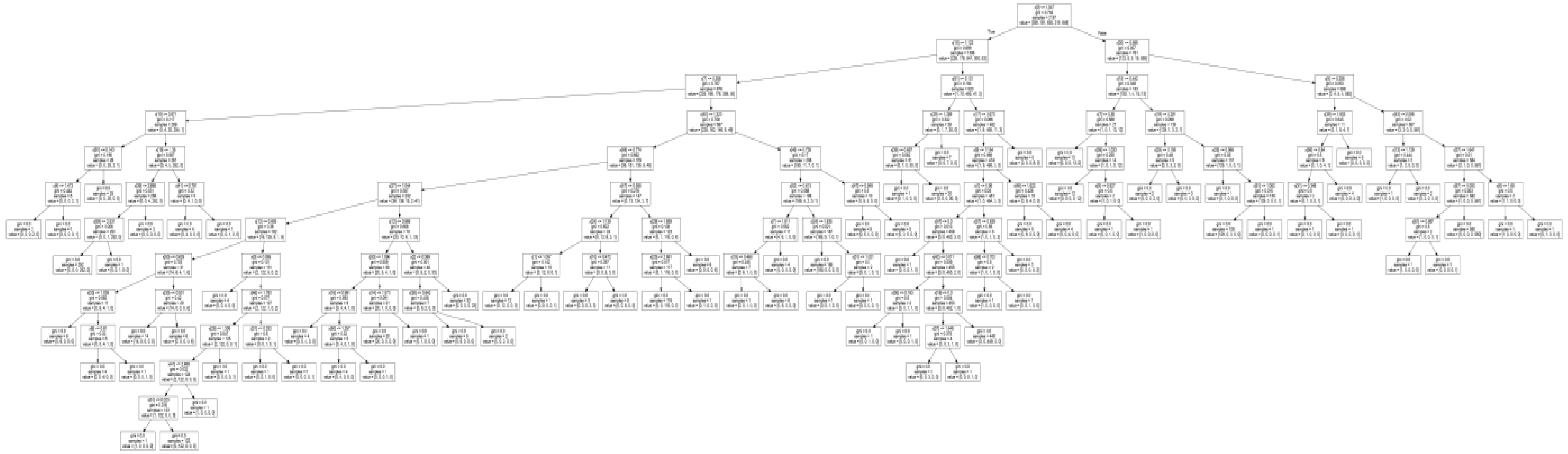
Interpretability: CNN

Class activation maps (CAM) are a visualization technique that allows us to see which parts of an input image a convolutional neural network (CNN) is using to make a prediction. CAMs are useful for understanding how a CNN is making its predictions and for identifying which parts of an image are most important for a particular class.

The basic idea behind CAMs is to generate a heatmap that shows the contribution of each pixel in the input image to the final prediction of a particular class. This can be done by taking the output of the last convolutional layer in the CNN and using global average pooling to reduce the spatial dimensions of the feature map to a single value. The resulting vector can then be used as input to a fully connected layer that predicts the class probabilities.



Interpretability: Decision Tree



Decision Tree model with a max depth of 12

Conclusion

In conclusion, it was seen that:

- The neural network reached near 100% accuracy, while the decision tree was slightly less successful

This is likely a result of differences in complexity:

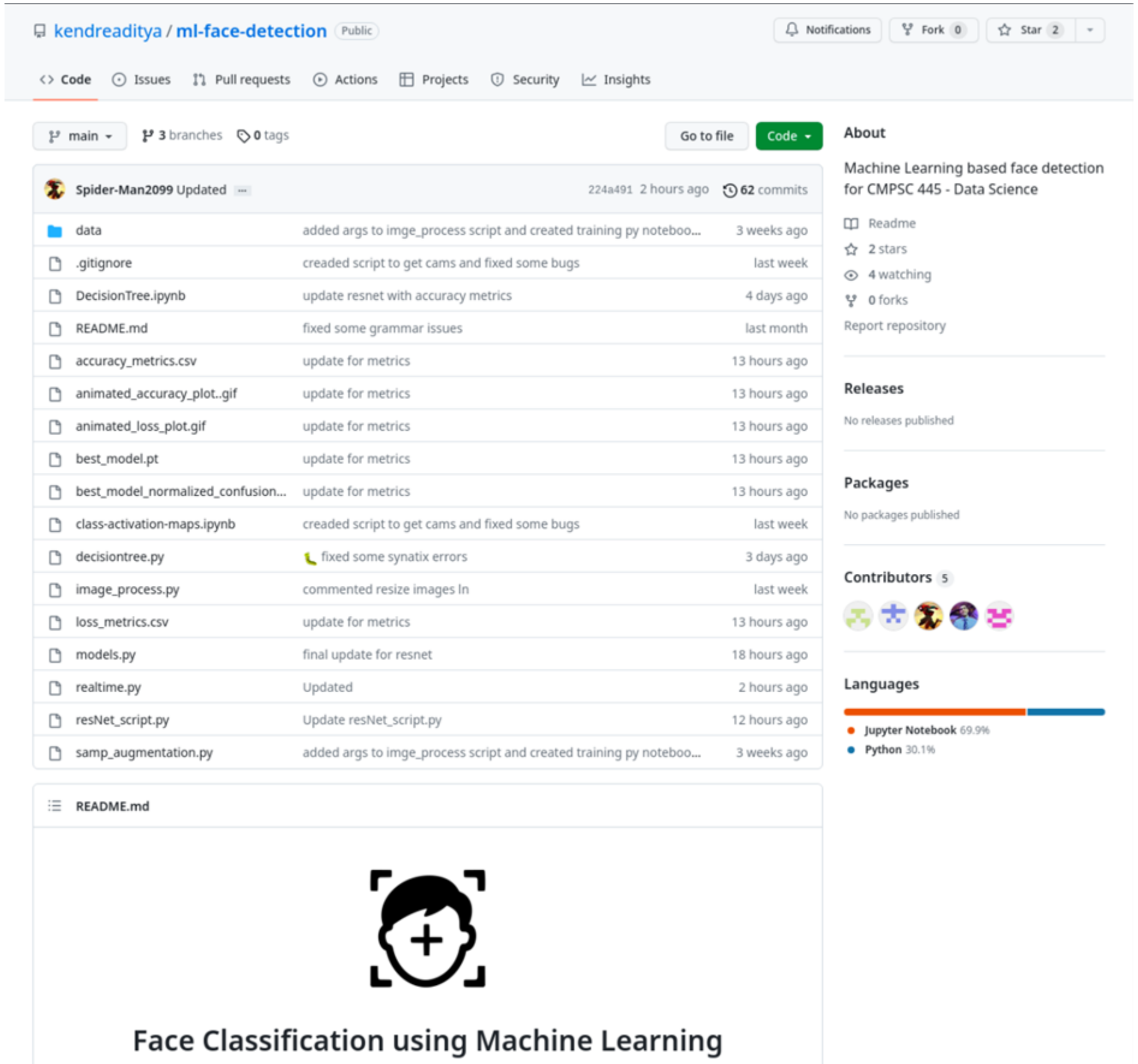
- Decision trees are essentially a linear series of if-statements that lose features as they grow closer to a decision
- Neural networks are much more refined as the data is passed back and forth
- In the case of image classification, a complex and nonlinear relationship between inputs and outputs, neural networks are the better option for classification

Therefore, to further evaluate the model we will be testing the model on a separate and completely new dataset distribution.

Code Availability

Our code is also available on GitHub for the project, allowing for easy reproducibility and transparency in our research.

github.com/kendreaditya/ml-face-detection



kendreaditya / ml-face-detection Public

Spider-Man2099 Updated ··· 224a491 2 hours ago 62 commits

data	added args to imge_process script and created training py noteboo...	3 weeks ago
.gitignore	creaded script to get cams and fixed some bugs	last week
DecisionTree.ipynb	update resnet with accuracy metrics	4 days ago
README.md	fixed some grammar issues	last month
accuracy_metrics.csv	update for metrics	13 hours ago
animated_accuracy_plot..gif	update for metrics	13 hours ago
animated_loss_plot.gif	update for metrics	13 hours ago
best_model.pt	update for metrics	13 hours ago
best_model_normalized_confusion...	update for metrics	13 hours ago
class-activation-maps.ipynb	creaded script to get cams and fixed some bugs	last week
decisiontree.py	🐛 fixed some synatix errors	3 days ago
image_process.py	commented resize images In	last week
loss_metrics.csv	update for metrics	13 hours ago
models.py	final update for resnet	18 hours ago
realtime.py	Updated	2 hours ago
resNet_script.py	Update resNet_script.py	12 hours ago
samp_augmentation.py	added args to imge_process script and created training py noteboo...	3 weeks ago

README.md

Face Classification using Machine Learning

About

Machine Learning based face detection for CMPSC 445 - Data Science

Readme
2 stars
4 watching
0 forks
Report repository

Releases

No releases published

Packages

No packages published

Contributors 5

Languages

- Jupyter Notebook 69.9%
- Python 30.1%