Intro to Machine Learning Final Project

Classification of my Dogs

Abraham Alpuerto

Rensselaer Polytechnic Institute

Intro to Machine Learning Final Project

Classification of my Dogs

Introduction

My project focuses on the classification of my three dogs, based on a data set that I have personally collected for many years. Leveraging fundamental machine learning techniques we have or are going to cover in this course, I aim to explore different methods for effective image classification. To optimize computational efficiency, all images will be standardized to a lower resolution. In addition, to reduce computational complexity, the images will undergo a Principal Component Analysis (PCA) for dimensionality reduction and feature extraction. Following this, I will implement a Gaussian Mixture Model (GMM), K-Nearest Neighbors (KNN), and a Fully Connected Neural Network using the reduced feature set from PPCA. To finish it off, I will be comparing the results of this method against a Convolutional Neural Network (CNN) considered the best approach for image classification.

Data Collection

The dataset includes over 2,000 photos of my three dogs with a even spread between them. The images exhibit a high degree of variance in both background settings and the dogs' positions, resulting in significant differences from one picture to another. Due to this variability, methods other than a CNN may struggle to handle the data effectively. All images were captured using a phone camera at very high resolutions (ranging from 4K down to 1920x1080), making them challenging to process in their original form.

To reduce complexity, I resized the images to 256x256 pixels, with three color channels (Red, Green, and Blue, or RGB). After this transformation, each flattened image consists of 196,608 features, and the dataset's shape is [2003, 3, 256, 256].

Regarding ethical considerations, the dataset consists solely of photos of my own dogs, and no individuals outside my immediate family are included. Consequently, there are no permission-related issues or ethical concerns regarding the use of these images.

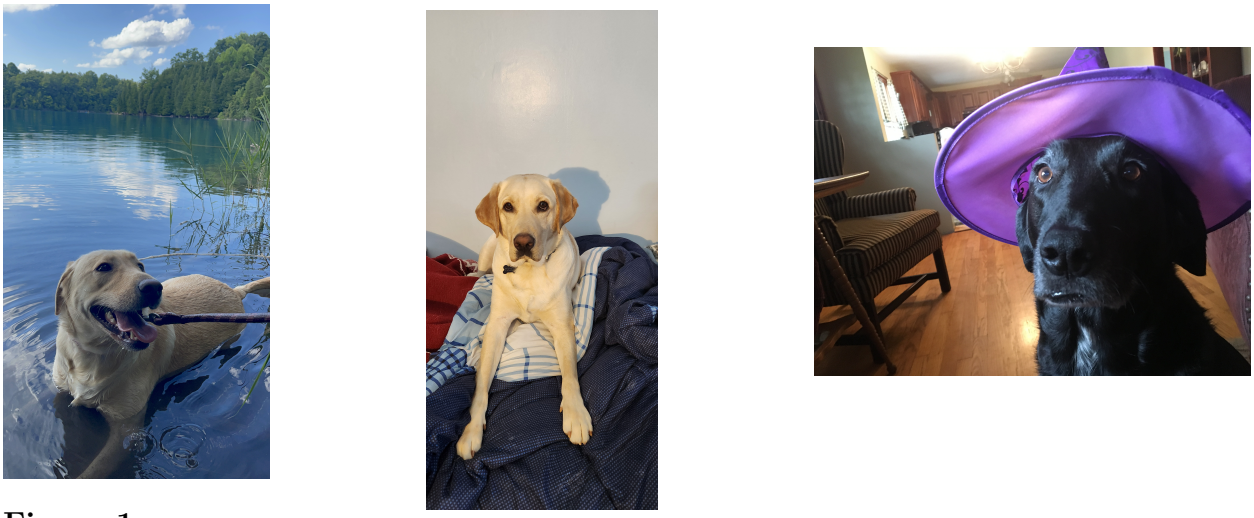

Figure 1

Pictures of my three dogs: Ellie, Tucker, Jessy.

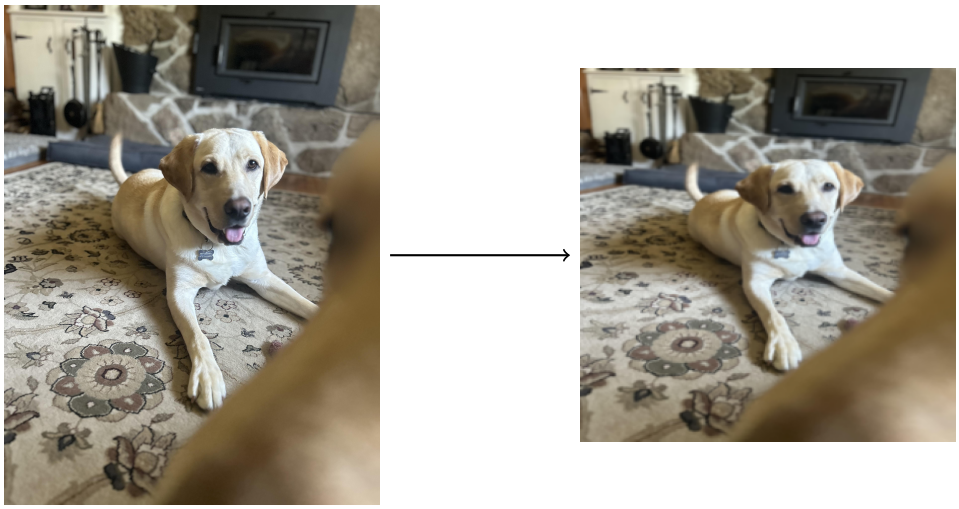

Figure 2

Left is a normal photo, right is a transformed photo.

Technical Approach

To classify the images of my dogs, I will be using a PCA algorithm to reduce the dimensionality of the data along with a GMM, KNN, and FCNN. This will be compared to a CNN which is one of the best in the image classification area. Each prediction algorithm is using a 90/10 data split with even fairly even split between all classes.

Principal Component Analysis (PCA)

The Principal component analysis is a reduction machine learning algorithm that takes the M most important components in the data and converts it to a different or lower dimension. This new dimension will keep a certain amount of the variance depending on the amount of M components that are used. The way this new state is calculated is through a series of equations:

1. Standardizing the dataset:

$$\mathbf{X}_{standardized} = \frac{\mathbf{X} - \mu}{\sigma} \tag{1}$$

2. Finding the covariance matrix Σ:

$$\mathbf{\Sigma} = \frac{1}{N-1}\mathbf{X^T X} \tag{2}$$

3. Finding the eigenvectors and eigenvalues of the covariance matrix:

$$\mathbf{\Sigma} v_i = \lambda_i v_i \tag{3}$$

4. Then take the top M eigenvectors $\mathbf{W}$ and project back onto the standardized data:

$$\mathbf{Z} = \mathbf{X}_{standardized}\mathbf{W} \tag{4}$$

Here, $\mathbf{Z}$ represents the data in a reduced-dimensional space, which will be used as input to the subsequent machine learning algorithms (GMM, KNN, and FCNN), excluding CNN. This is implemented using a standard PCA library to generate various datasets that hold a certain percentage of variance.

Gaussian Mixture Model (GMM)

. After doing PCA on the data, a Gaussian mixture model will be applied to classify the images. GMM is a probabilistic model assuming that the data has some sort of normal distributions. The highlighting of certain features from the PCA compliments a GMM which its clusters can target. GMM can leverage distinct visual characteristics of each dog to form clusters for classification.

1. Initialization

To compute the GMM we can apply a EM algorithm just like in the PCA. We start by initializing variables:

- π_k the probability of each cluster

- μ_k the mean vector which will just be random 3 data points

- $\mathbf{\Sigma}_k = \mathbf{I}$ the covariance matrix

The EM algorithm will iteratively refine these parameters to fit the GMM to the data.

2. EM Algorithm

E-Step: Evaluate the responsibility for each data point.

$$\gamma_{nk} = \frac{\pi_k N(\mathbf{x}_n|\mu_k, \sum_k)}{\sum_{j=1}^{K} \pi_j N(\mathbf{x}_n|\mu_j, \sum_j)} \tag{5}$$

M-Step: Re-Estimate the GMM parameters given responsibility.

$$\mathbf{N}_k = \sum_{n=1}^{N} \gamma_{nk} \tag{6}$$

$$\mu_k^{new} = \frac{1}{\mathbf{N}_k} \sum_{n=1}^{N} \gamma_{nk} \mathbf{x}_n \tag{7}$$

$$\mathbf{\Sigma}_k^{new} = \frac{1}{\mathbf{N}_k} \sum_{n=1}^{N} \gamma_{nk} (\mathbf{x}_n - \mu_k^{new})(\mathbf{x}_n - \mu_k^{new})^T \tag{8}$$

$$\pi_k^{new} = \frac{\mathbf{N}_k}{N} \tag{9}$$

Iterations and End: The EM-Algorithm will go through multiple iterations to tune the GMM. After convergence we can take a look at each data point in the cluster. We will take the majority label of each cluster then give each cluster a label.

K-Nearest Neighbors

K-Nearest Neighbors is a very simple machine learning algorithm that works exceptionally well for the complexity of the algorithm. As it states in the name a KNN will

classify using a certain K nearest neighbors. This was implemented along with the PCA reduced data to provide a simple classification. Multiple neighbors are tried to generate best possible model for example:

```
for i in range(1,5):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train,y_train)
    score = neigh.score(X_test,y_test)
    print(f"Accuracy with n = {i} neighbors: {score}")
```

Fully Connected Neural Network (FCNN)

A Fully Connected Neural Network (FCNN) is a type of Multi-Layer Perceptron (MLP) in which every neuron in one layer is connected to every neuron in the next layer. A perceptron, a fundamental building block of neural networks, takes input values, applies learned weights and biases, and then uses an activation function to determine if it should 'fire.' During training, the weights and biases are adjusted to improve the network's predictions.

The following activation functions were used in the network:

$$ReLU(x) = max(0, x) = \frac{x + |x|}{2} = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases} \tag{10}$$

$$softmax = \sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{11}$$

ReLU and softmax are two of the most commonly used activation functions in neural network architectures. ReLU helps the network model complex relationships and mitigates the vanishing gradient problem. Softmax is typically used at the output layer in multi-class classification problems to normalize the output values into a probability distribution over classes.

The loss optimizer used to implement this neural network was Adam. Adam stands for adaptive moment estimation, a specialized technique for gradient descent. It combines an adaptive learning rate and momentum, enabling it to converge more quickly. In this case, PyTorch's Adam optimizer was implemented along with L2 regularization:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{original}} + \lambda \sum_{i} \|\mathbf{w}_i\|_2^2 \tag{12}$$

Where:

- $\mathcal{L}_{\text{total}}$ is the total loss with L_2 regularization.
- $\mathcal{L}_{\text{original}}$ is the original loss function (e.g., cross-entropy or MSE).
- λ is the regularization parameter (weight decay factor) controlling the strength of regularization.
- $\mathbf{w}_i$ represents the weights of the i-th layer.
- $\|\mathbf{w}_i\|_2^2$ is the squared L_2-norm of the weights, computed as $\|\mathbf{w}_i\|_2^2 = \sum_j w_{ij}^2$.

The built-in neural network library was used. The network had one hidden layer with 256 neurons and dropout between each layer, for a total of two layers with dropout. Dropout deactivates a random number of neurons during training. In this network, the input and hidden layers used ReLU activation functions and dropout, while the output layer used softmax. A learning rate of 0.0005 and a batch size of 512 (to help prevent over fitting) over 2500 epochs was employed. The model with the best testing set value was saved.

Convolutional Neural Network (CNN)

I designed a Convolutional Neural Network (CNN) to classify the dogs in the images. A CNN processes input images through a series of convolutional layers, each layer using small filters to identify patterns within the data. The network was custom-built using PyTorch's libraries and consists of five convolutional layers followed by a fully

connected layer with dropout. All layer activation functions, except for the output layer, use ReLU. Batch normalization and average pooling were applied between convolutional layers to aid in regularization. The following is the actual code to my final CNN after multiple testing iterations:

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, 5)
        self.bn1 = nn.BatchNorm2d(16)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, 3)
        self.bn2 = nn.BatchNorm2d(32)
        self.conv3 = nn.Conv2d(32, 64, 5)
        self.bn3 = nn.BatchNorm2d(64)
        self.conv4 = nn.Conv2d(64, 128, 3)
        self.bn4 = nn.BatchNorm2d(128)
        self.conv5 = nn.Conv2d(128, 256, 5)
        self.bn5 = nn.BatchNorm2d(256)

        self.global_avg_pool = nn.AdaptiveAvgPool2d((1, 1))

        self.fc1 = nn.Sequential(nn.Linear(256, 1024),
                                 nn.ReLU(), nn.Dropout(0.8))
        self.fc2 = nn.Linear(1024,3)

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
```

```
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = self.pool(F.relu(self.bn3(self.conv3(x))))
        x = self.pool(F.relu(self.bn4(self.conv4(x))))
        x = self.pool(F.relu(self.bn5(self.conv5(x))))
        x = self.global_avg_pool(x)

        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        return x
```

For CNN, a stochastic gradient descent was used for the loss function. This is due to the fact that SGD performs better on multiple trails than Adam. The optimization algorithm with momentum updates the parameters of the network as follows:

1. Velocity update with momentum:

$$v_t = \mu v_{t-1} + \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{13}$$

2. Parameter update:

$$\theta_{t+1} = \theta_t - v_t \tag{14}$$

Where:

- θ_t represents the parameters (weights) at iteration t.
- $\mathcal{L}(\theta_t)$ is the loss function evaluated at θ_t.
- $\nabla_\theta \mathcal{L}(\theta_t)$ is the gradient of the loss with respect to the parameters.
- v_t is the velocity (accumulated gradient) at iteration t.
- μ is the momentum factor (typically between 0 and 1).

- η is the learning rate.

This all ran for 100 epochs with a 0.0008 learning rate and a momentum of 0.9 in the SGD optimizer. The model was also trained on a batch size of 64.

Data Augmentation was also a method that was used in a separate model. The data was augmented dynamically so every training iteration was different. Transformation of each picture with horizontal flips with a probability of 25%, rotation of 10 degrees, and random color jitter with every training step.

Evaluation and Comparison

To compare the effectiveness of the different techniques implemented, I evaluated each model on a 90/10 train-test split of the dataset, ensuring an even distribution across all classes.

In terms of classification, a CNN is expected to achieve the highest accuracy. However, this comes with significant computational demands and time spent tuning hyper-parameters. Training the CNN can take up to an hour, depending on the dataset size and system specifications. For the case of using data augmentation the model training then takes up to 2 hours for training. In contrast, other methods, which are not expected to perform as well, complete training in under two minutes.

Results

PCA Reductions

First, to determine the optimal amount of variance to retain for each model, I conducted tests on datasets with different levels of dimensionality reduction. Among the various trials, the best results were achieved when 60% of the variance was preserved in the images. Notably, the KNN model, using only the nearest neighbor, performed the best among these models despite being the simplest algorithm implemented. In this case, the most effective approach was essentially just a "Nearest Neighbor algorithm."

Table 1

Classification Accuracies at Different PCA Variance Levels

PCA Variance	**Method**	**Parameters**	**Accuracy**
90% Reduced dimension: 551	KNN	k=1	86%
	KNN	k=2	79%
	KNN	k=3	81%
	KNN	k=4	79%
	FCNN	-	83%
	GMM	-	52%
80% Reduced dimension: 207	KNN	k=1	88%
	KNN	k=2	86%
	KNN	k=3	84%
	KNN	k=4	83%
	FCNN	-	82%
	GMM	-	64%
70% Reduced dimension: 85	KNN	k=1	91%
	KNN	k=2	89%
	KNN	k=3	88%
	KNN	k=4	87%
	FCNN	-	85%
	GMM	-	65%
60% Reduced dimension: 38	KNN	k=1	93%
	KNN	k=2	89%
	KNN	k=3	88%
	KNN	k=4	88%
	FCNN	-	92%

PCA Variance	Method	Parameters	Accuracy
	GMM	-	66%

Training Results

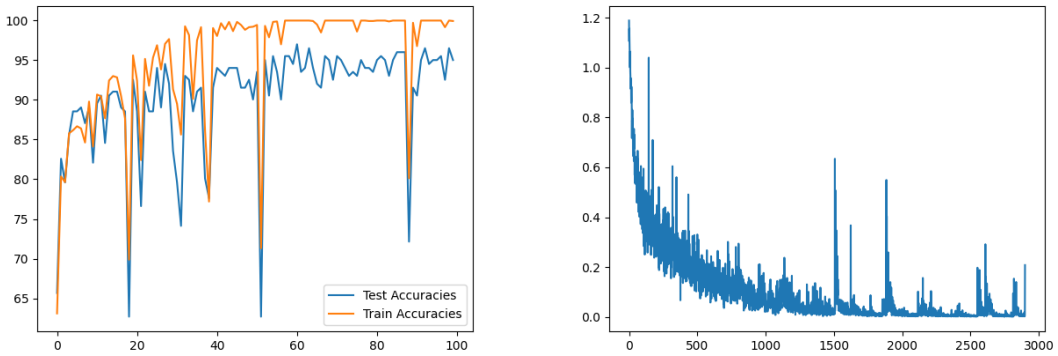

Figure 3

CNN training loss and accuracy graphs

Shown above is the training accuracies and loss graphs. From this it is apparent the use of SGD opposed to Adam as it is known to be more spiky as it jumps in and out of local minima. The peak for testing data was shown to peak at around 60 epochs which is where the model was saved before eventually over fitting.

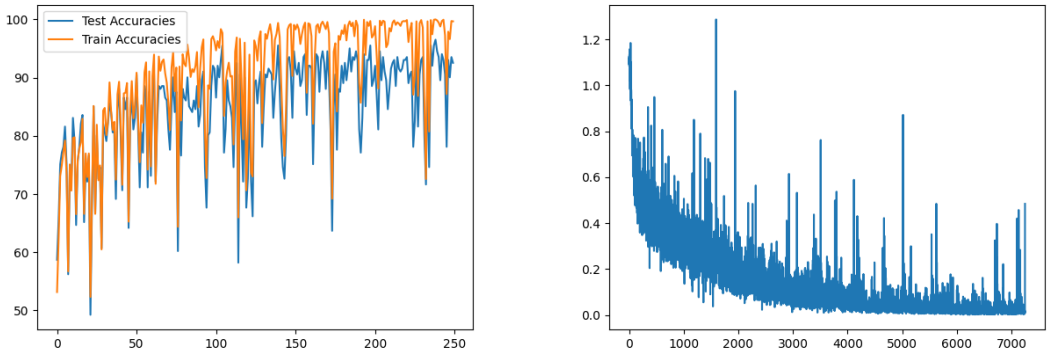

Figure 4

CNN with data augmentation training loss and accuracy graphs

The next set of graphs show the data augmentation CNN mode. As seen the data augmentation made the convergence take much longer to converge as it ran for 250 epochs. The same idea though as we can see from the graphs shape that SGD is being used and that the pictures must be changing as the accuracy fluctuates lots. The peak can be seen around 230 epochs while in the midst of possibly over fitting.

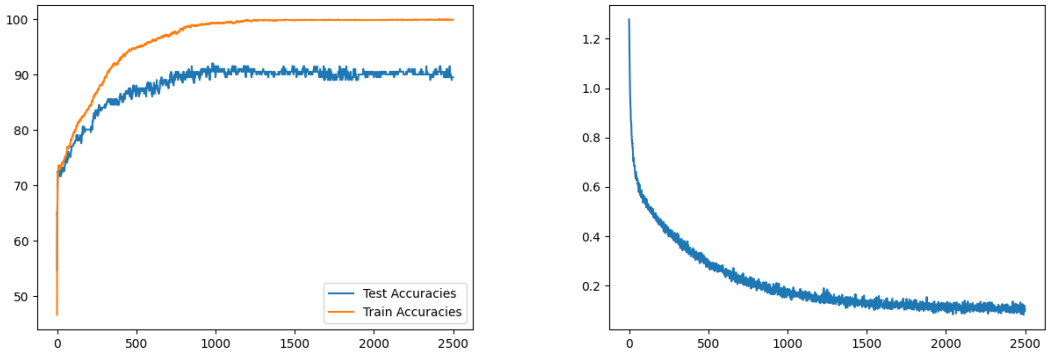

Figure 5

FCNN training loss and accuracy graphs

These last pictures of training loss and accuracy graphs are from the FCNN. As can be seen the network seems to over fit fairly easily even with many attempts at tuning to prevent this over fitting this was the best results for the following. Looking at the loss graph we can see a very smooth graph indicating the Adam optimizer compared to SGD like in the other two graphs.

Models Results

Overall, the models performed as expected, with the CNN achieving the best performance at 97% accuracy. However, the data augmentation approach did not perform as well and required nearly five times the training time. Realistically, I believe that more diverse data with different backgrounds would have been needed, rather than augmented data that only slightly altered properties like tilt or color. The large amount of variance in the dataset likely stems from the backgrounds rather than minor variations in the images.

Table 2

Comparison of Models

Model	**Test Accuracy**	**Training Time**
CNN	97.02%	35 minutes
CNN + Data Augmentation	95.62%	2.5 hours
FCNN	92.04%	2 minutes
GMM	64.02%	$<$ 1 minute
KNN k = 1	93.03%	$<$ 1 minute
KNN k = 2	89.55%	$<$ 1 minute
KNN k = 3	88.55%	$<$ 1 minute
KNN k = 3	88.05%	$<$ 1 minute

A surprising result came from the KNN model, particularly with the nearest neighbor approach, which performed exceptionally well. It achieved 93% accuracy on the test set when the data was reduced using PCA with 60% variance retained. The GMM, on the other hand, significantly underperformed compared to the other models, with a low score of 64%. This may be due to poor initial cluster placements, causing the clusters to converge on suboptimal configurations. Overall, I believe the KNN provided the best balance of accuracy (93%), ease of implementation, extremely minimal hyper-parameter tuning, and quick training time.

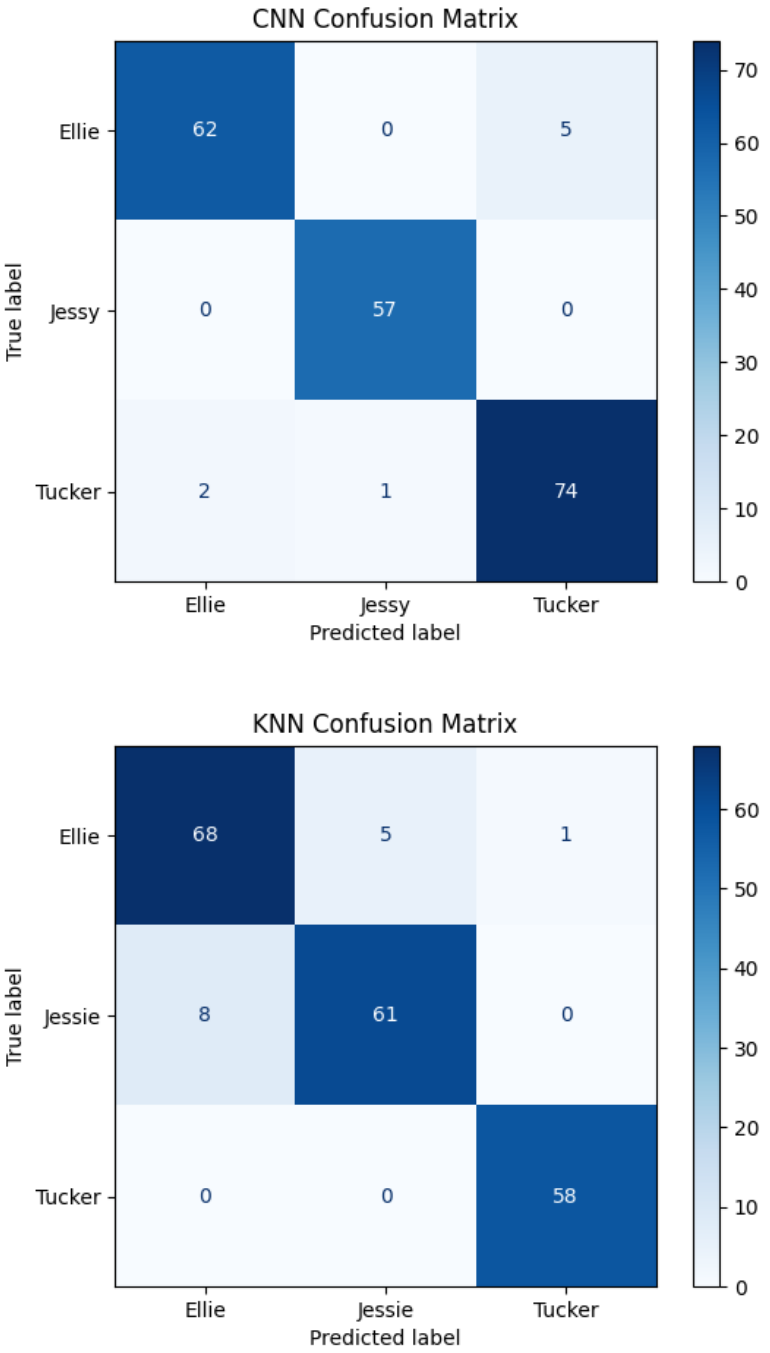

Figure 6

Confusion Matrix of the top 2 Algorithms

Looking deeper into the results for the CNN confusion matrix, Ellie and Tucker were the most frequently misclassified, which aligns with expectations given their similar colors

and features compared to Jessie. For the KNN, the results presented a different challenge. Misclassifications between Ellie and Jessie are particularly puzzling, as they look nothing alike. This may be attributed to the parts of the images that PCA retained, such as their collars occasionally being the same color or similarities in some backgrounds. However, this remains a "black box" situation, where the internal reasoning of the model is unclear.

Discussion and Future Work

Overall, I believe this project was successful, and I achieved very promising results. However, I recognize there is room for improvement, particularly through additional time spent tuning hyperparameters and optimizing the CNN's loss function to find the global maximum. If I had more time for this project, one key improvement would be to expand the dataset. Popular image datasets like MNIST or ImageNet are extremely large, and I noticed that as I added more images to my dataset, the accuracy improved across all algorithms. Another potential improvement would be to give data augmentation more attention, experimenting with different levels of transformation to bring its performance closer to that of the standard CNN.

For future work or a different direction, I think adding predictions from videos could be an interesting extension. Instead of single tensors of shape [RGB, 256, 256], the data could be structured as [Frames, RGB, 256, 256]. From here maybe the way the dog moves on the videos a CNN or different types of models would be able to tell which dog it is with higher accuracy. I would be especially interested in how a KNN would do on this and with adding PCA into this. The only problem with this is the complexity increases extremely quickly and to handle this data I would need more computer power and use some sort of cluster. These extensions would build upon topics from this course like PCA, KNN, and Neural Networks in a challenging but an exciting way.