**Notes on the Cat/Dog Training**

**Objective**

The plan is to build a website that takes an image and seeks to determine if it is a dog, cat, or human by providing a percentage of what it likely is. The goal is to provide four options: dog, cat, human, or something else beyond the model’s capability.

**Steps**

1. Gather and assess the data.

2. Process the data to pick the best images for the model.

3. Split the data into training, validate, and test sets.

4. Normalize the data.

5. Train the model.

6. Test the model.

7. Build the website.

**Gather the Data**

**Data Sources**

1. <https://www.robots.ox.ac.uk/~vgg/data/pets/>

This website has over 7,000 images of cats and dogs of different breeds. These images have been well-sorted and prepared.

2. <https://www.kaggle.com/datasets/jessicali9530/lfw-dataset>

This website has over 13,000 human images already sorted with a focus on faces.

**Data Preprocessing**

This covered the work done in the following files:

1. data\_preprocessing\_1

2. data\_preprocessing\_2  
3. data\_preprocessing\_3

**Organizing the Data**

This is the distribution of the cat and dog breeds, along with the number of images in the initial data.

### ****Cat Breeds****:

1. **Abyssinian – 200 images**
2. **bengal – 200 images**
3. **birman – 200 images**
4. **bombay – 200 images**
5. **british** (British Shorthair) **– 200 images**
6. **egyptian** (Egyptian Mau) **– 200 images**
7. **maine** (Maine Coon) **– 200 images**
8. **persian – 200 images**
9. **ragdoll – 200 images**
10. **russian** (Russian Blue) **– 200 images**
11. **scottish** (Scottish Fold) **– 199 images**
12. **siamese – 200 images**
13. **sphynx – 200 images**

****Total Cat Images - 2599****

### ****Dog Breeds****:

1. **american** (American Bulldog or American Eskimo) **– 400 images**
2. **basset** (Basset Hound) **– 200 images**
3. **beagle – 200 images**
4. **boxer – 200 images**
5. **chihuahua – 200 images**
6. **english** (English Bulldog or English Springer Spaniel) **– 200 images**
7. **german** (likely German Shepherd) **– 200 images**
8. **great** (likely Great Dane) **– 200 images**
9. **havanese– 200 images**
10. **japanese** (Japanese Chin or Spitz) **– 200 images**
11. **keeshond – 200 images**
12. **leonberger – 200 images**
13. **miniature** (Miniature Schnauzer) **– 200 images**
14. **newfoundland – 200 images**
15. **pomeranian – 200 images**
16. **pug – 200 images**
17. **saint** (Saint Bernard) **– 200 images**
18. **samoyed – 200 images**
19. **shiba** (Shiba Inu) **– 200 images**
20. **staffordshire** (Staffordshire Bull Terrier) **– 191 images**
21. **wheaten** (Soft Coated Wheaten Terrier) **– 200 images**
22. **yorkshire** (Yorkshire Terrier) **– 200 images**

****Total Dog Images – 4791****

****Picking the Best Images to Use****

**Since the goal is to pick a cat,dog, human, I knew the best approach was to place all cat images in the same place and pick images randomly to train the model. To do this, I placed all of the cat, dog, and human images in their own folders. They would be placed in random order without acknowledgment of the different breeds. The breeds would be placed together without order.**

**Once I did that, I proceeded to apply the YOLOv5 on the cat and dog images. YOLOv5 is a deep learning-based object detection model that stands for "You Only Look Once" version 5. It is designed to detect and localize multiple objects within an image in a single forward pass, making it both fast and efficient for real-time applications. YOLOv5 is implemented in PyTorch and comes in different model sizes (e.g., YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x) to balance speed and accuracy depending on the use case. It outputs bounding boxes, class labels, and confidence scores for each detected object, and is widely used for tasks like surveillance, autonomous driving, and image dataset preprocessing.**

**In my project, I use YOLOv5 object detection model to identify and extract images of cats and dogs. The human images were already processed and well-curated as the source is very reputable. The model detects objects in an image and provides bounding boxes with confidence scores. I filtered out any detections with a confidence score below 50% to ensure quality. For each valid detection, I cropped the image to the bounding box, resized it to 224×224 pixels, and saved the cleaned image into organized output folders. This process helped create a high-quality, standardized dataset for further use.**

**After this step, there were 1139 cat and 3282 dog images left. However, I was certain that these images were of great quality and that could be of great impact to the model I end up creating.**

****Splitting The Data into Training, Validation, and Test Sets****

**The next step was to split the data into these three groups. I chose a ratio of of 80%, 20%, and 20% for training, validation, and test sets respectively. At this point, the breakdown of the data in the train, validation, and test sets is as follows:**

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Folder Name | | |
|  | Cat | Dog | Human |
| Train | 911 | 2625 | 10586 |
| Validate | 113 | 328 | 1323 |
| Test | 115 | 329 | 1324 |

**This split shows that there is a great discrepancy in the training data. There are very few cat images and very many human images. The next step in this case is to try and address that. First, I wrote code that picked 2500 random human images. This would allow me to create training data that does not bias the data.**

****Augmenting the Training Data****

**In this case, I wanted to augment the cat images to get them to at least 3000 images for each to allow for better model training. This is where augmentation came in. Augmentation in image preparation refers to the process of artificially increasing the size and diversity of a dataset by applying various transformations to the original images. These transformations can include operations such as rotation, flipping, scaling, cropping, brightness adjustment, noise addition, and more. The goal of augmentation is to help a model generalize better by exposing it to a wider variety of image conditions, reducing the risk of overfitting and improving performance on unseen data. It is especially useful when working with limited datasets. With that, I have 3000 images of all three creatures for training.**

****Transforming The Data****

**After this, I placed the training, test, and validation folders under the same folder. I wanted to solely augment the training data while leaving the rest out of it. The next step is transformation. Transformation. in this context, refers to the preprocessing steps applied to each image before it is fed into the model. Specifically, the images are first converted from PIL format (or NumPy arrays) into PyTorch tensors using transforms.ToTensor(), which also scales pixel values to the [0, 1] range. Then, transforms.Normalize([0.5], [0.5]) standardizes the data by shifting the pixel values to the range [-1, 1] using a mean and standard deviation of 0.5. This normalization helps the model train more effectively by ensuring the input values are centered and scaled, which can lead to faster convergence and more stable gradients during backpropagation. Proper transformations also ensure that the input data format matches the model's expected input, which is crucial for correct and efficient training.**

**After transformation, I set up data loaders. Data loaders in PyTorch are utilities that handle the loading of datasets in manageable batches, optionally shuffling the data and loading it in parallel using multiple worker threads. In this case, DataLoader wraps around each dataset (train, validation, test) and creates an iterator that yields batches of 32 images at a time. This batching process is essential for training neural networks efficiently, especially when using GPUs, as it allows parallel computation across multiple samples. Additionally, shuffling the training data each epoch improves generalization by preventing the model from learning the order of the data. Using data loaders also helps manage memory efficiently, as images are only loaded as needed, rather than all at once into memory.**

****Summary Table****

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Train** | **Validation** | **Test** |
| Resize (to 224\*224) | ✅ | ✅ | ✅ |
| Augmentation | ✅ | X | X |
| ToTensor (Transformation) | ✅ | ✅ | ✅ |
| Normalize | ✅ | ✅ | ✅ |

**The Modeling Phase**

## **Summary of Client Upload Pipeline**

1. **Client uploads image**
2. **Run YOLOv5 to detect object**
3. **Crop image to bounding box**
4. **Resize to 224×224**
5. **Convert to tensor**
6. **Normalize**
7. **Feed into trained model**
8. **Get class probabilities (e.g., Cat: 85%, Dog: 10%, Human: 4%, Other: 1%)**
9. **Return label and confidence**