DOG BREED CLASSIFICATION

Stephen Parvaresh

Definition

Project Overview

Image classification is one of the major topics in the machine learning research and application field. This category of techniques provides an essential component for applications such as security, auto-pilot systems, product quality control, retail, and scientific research. Image classification is commonly achieved by use of convolutional neural networks (CNNs).
The purpose of this project is to use a CNN to classify dog breeds using images of dogs and humans.

Problem Statement

There are countless number of dog breeds in the world, and it can be very difficult to tell them apart. This project aims to develop a deep learning model using a convolutional neural network that can distinguish a breed of a dog given a picture of the dog. The finished model should firstly feature the ability to distinguish whether the supplied picture is a dog or a human. Secondly, the model should accurately identify dog breeds in images of dogs, or what dog breed the human would be in images of humans. The resulting model for this project accomplished both of those tasks with 76% accuracy in detecting the breed of the dog in the image provided to it.

Metrics

Because the data is imbalanced, a simple accuracy score is not a useful metric to determine model performance. Therefore, the metrics for judging how the CNN model performs are validation loss values and prediction accuracy against test dataset. The validation loss value is defined by cross- entropy loss, which is also called the log loss or multi-class loss. The loss value measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy values increase if the predicted probability diverges from the actual label.

Analysis

Data Exploration

The dataset is supplied by Udacity machine learning engineering nanodegree program. The supplied datasets include a series of dog pictures with corresponding breeds and a series of human pictures.
The dog breed dataset includes the following components:

1. Train

    The training dataset contains 133 dog breeds with 30-70 pictures for each breed, there are 6680 dog images in total for training.

2. Test

    The test dataset contains the same 133 dog breeds with 6-10 pictures for each breed, there are 836 dog images in total for testing.

3. Validation

    The validation dataset contains the same 133 dog breeds with 6-10 pictures for each breed, there are 836 dog images in total for validation.

The human dataset contains 13233 different pictures of humans. Figure 1 shows the distribution of dog breeds from the data provided.

The class distributions in test, train, and validation datasets seem quite uniform, with more uniformity in the validation dataset. Additionally, all the images are in RGB color space. All the human images feature a resolution of 250x250. Images of dogs feature various resolutions from ~200x~200 to ~1000x~1000.

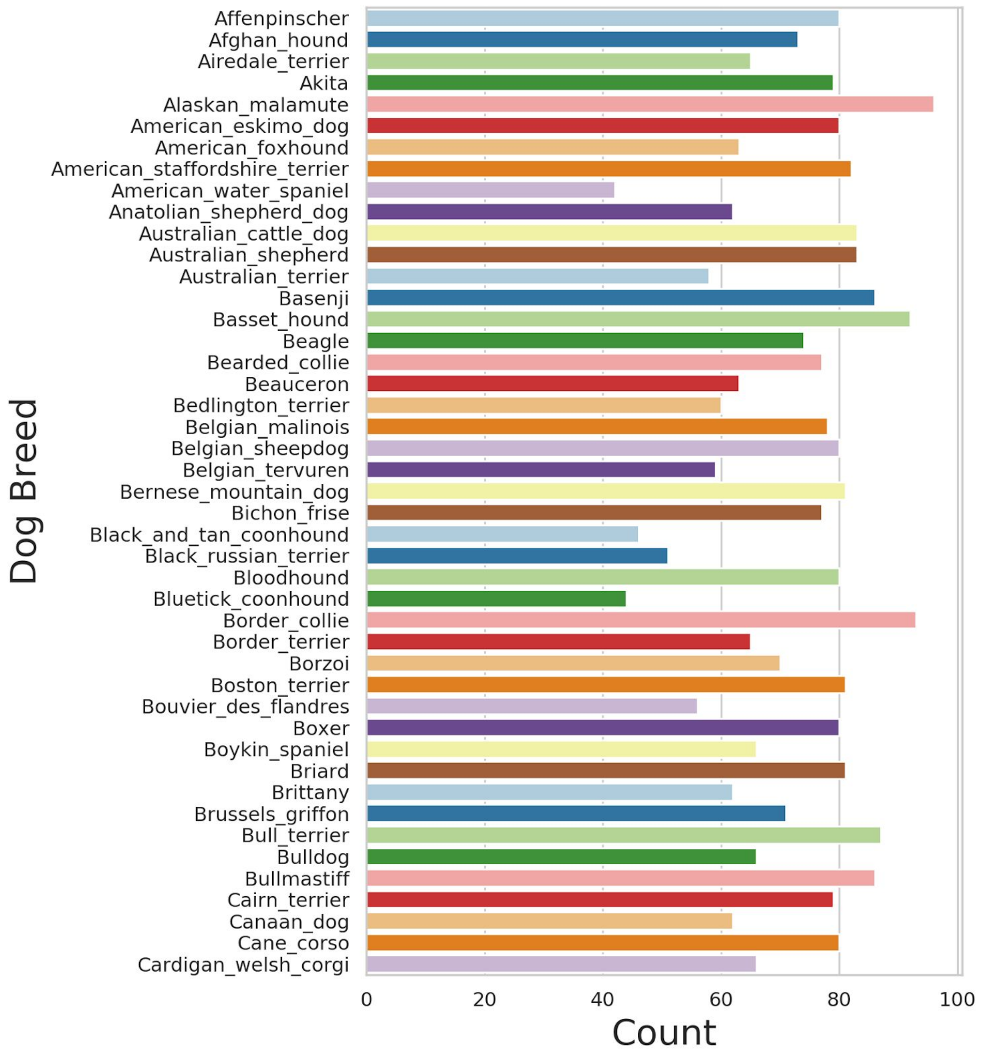


Figure 1. Distribution of dog breeds in datasets

Exploratory Visualization

The majority of the images contain a single portrait of a dog of the corresponding breed. However, the dataset contains a small portion of images that include both dogs and humans, multiple dogs with different breeds (Figure 2). The challenge is how to successfully detect whether there is a dog in the image and how not to identify humans as dogs, or vice versa.

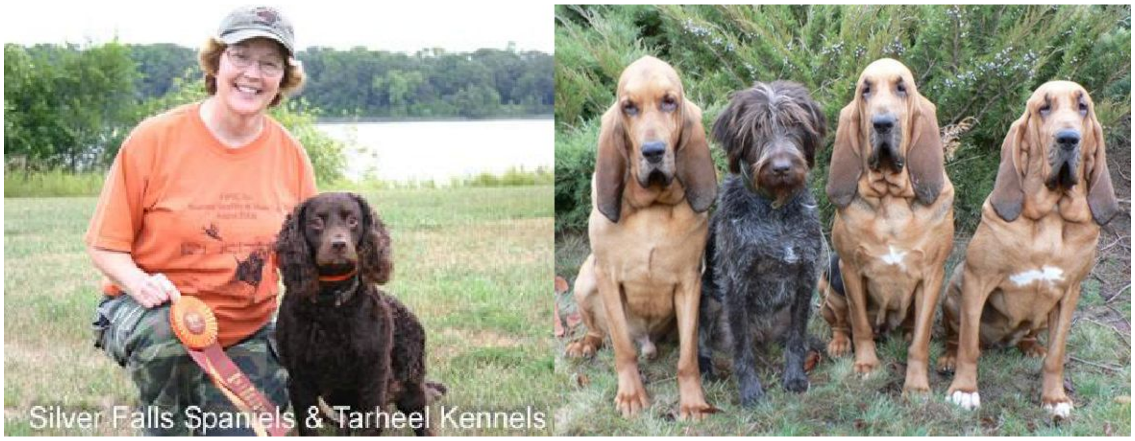


Figure 2. Images from the training dataset showing humans with dogs and multiple dogs with different breeds.

The human dataset is similarly structured with most pictures featuring a single human portrait in the middle of the frame. There are some pictures showing multiple humans in a frame, but one human is emphasized (Figure 3).

Figure 3. A picture in the human dataset containing multiple faces.

Algorithms and Techniques

The final model of the project included a trained detector using CNN to detect dog breeds based on the supplied picture. CNN (convolutional neural network) is a deep learning framework which is widely used in image classifications. Convolutional layers are one of the basic components of a CNN model. CNN uses multiple layers of convolutional filters to extract features out of an image such as vertical/horizontal edges, groups, certain shapes, and colors. A complete CNN model also features hyperparameters such as number of epochs, batch size, pooling, number of layers, weight, and dropouts. Example explanations of some hyperparameters are listed below:

1. Number of epochs: the number of times the entire training set passes through the neural network.
2. Number of hidden layers: the number of convolutional and linear layers specified in the CNN model, more layers can result in higher computational cost, less layers can result in underfitting.
3. Dropout: a preferable regularization technique to avoid overfitting in deep neural networks. The method simply drops out units in the neural network according to the desired probability.

The three hyperparameters listed above have been examined in this project.

CNNs work extremely well with image classification applications because it follows a hierarchical model which resembles the way the human brain processes images. The established model features fully connected layers where all the neurons are connected to each other with specified outputs. Images are composed of clouds, edges, colors, etc. These features are easily extractable by convolutional filters in the CNN models. Therefore, CNN is exceptionally effective in image classification applications.
Specifically, the input picture will go through two detectors, namely, human detector and dog detector. If human is detected in the picture, the algorithm will respond by displaying the image with a text describing the most resembled dog breed. If a dog is detected in the picture, the algorithm will display the image with a text indicating which breed the dog should belong to. If no human or dog can be recognized by the model, a text will be printed out indicating no human or dog is detected in the image. The human detector is constructed using a haar-cascade pre-trained model, the dog detector is constructed using a VGG16 pre-trained model. The dog breed classifier is a transferred pre-trained ResNet50 model. The flow chart is presented as Figure 4.

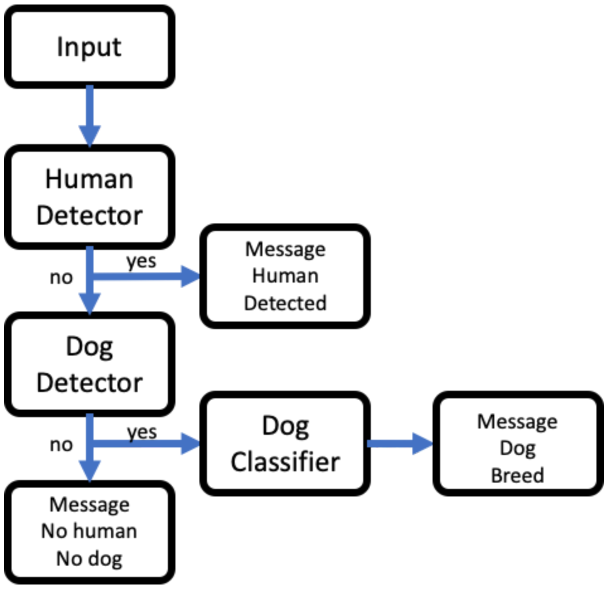


Figure 4. Dog classifier flow chart

Benchmarks

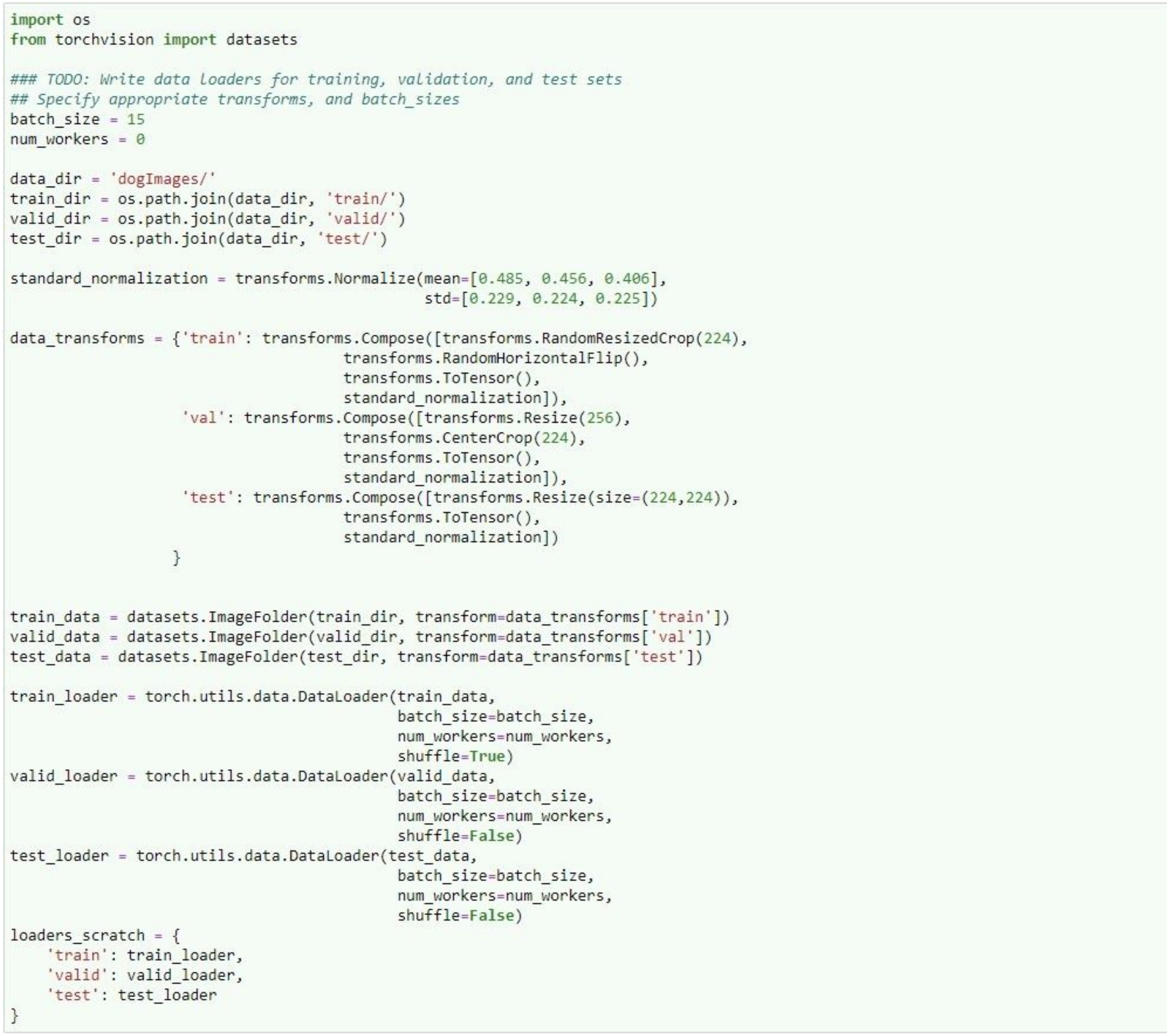
The benchmark for the model will be 90% prediction accuracy, which will be used as the upper limit. The benchmark set by Udacity will be 60% prediction accuracy, which will be used as the lower limit. The final performance of the model will sit in between the two limits.

Methodology

Data Preprocessing

The training, test, validation images are resized and center cropped into 224x224 pixels, then randomly flipped in the horizontal direction before transforming into tensors. The transformed data are organized into train, test, and validation directories, respectively. The corresponding reprocessing code blocks are shown in Figure 5.

The reason for resizing, cropping is to achieve a uniform input data format. The random flipping increases the data variation in terms of features, thus making the dataset more robust.

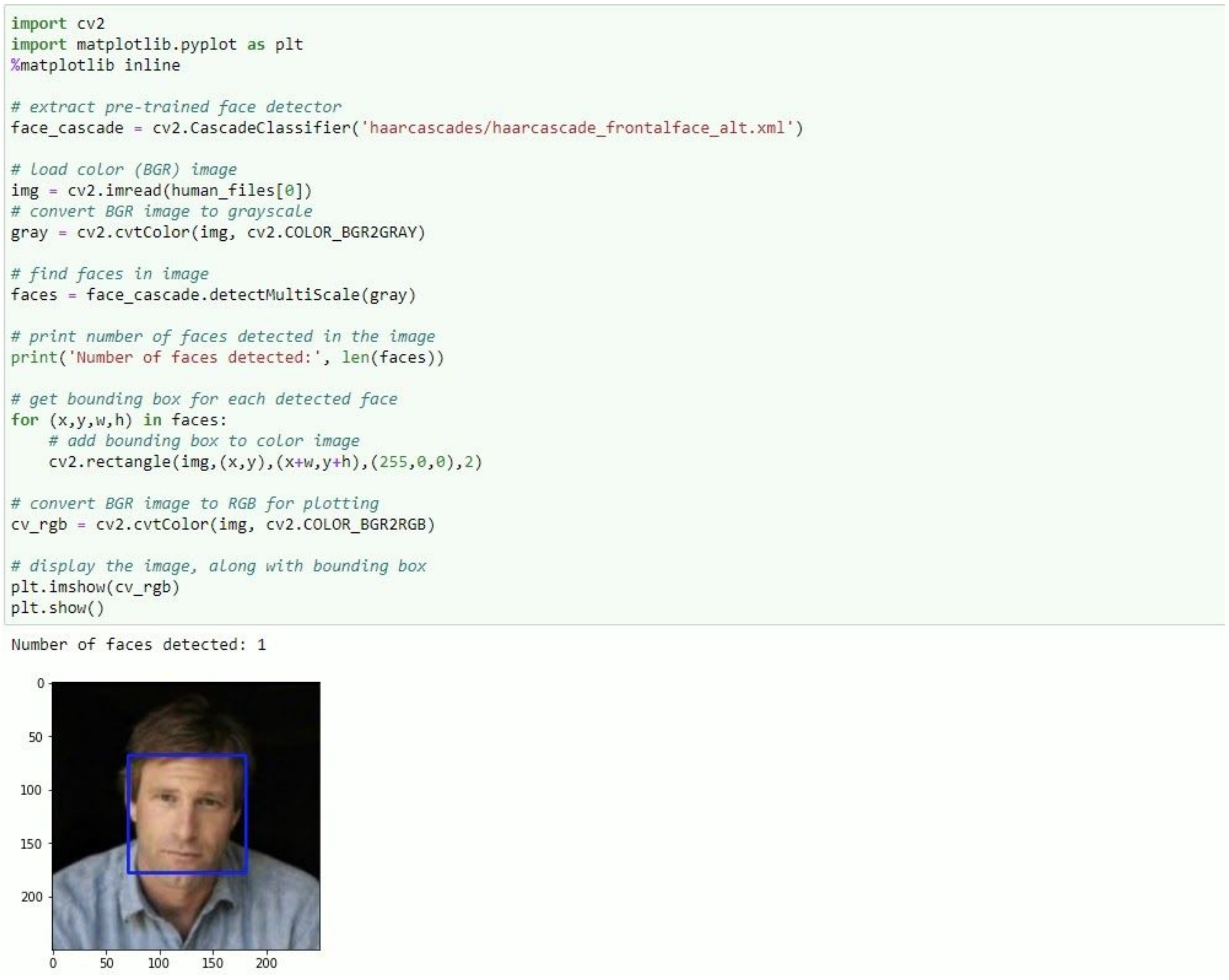


```python
import os
from torchvision import datasets

### TODO: Write data loaders for training, validation, and test sets
## Specify appropriate transforms, and batch_sizes
batch_size = 15
num_workers = 0

data_dir = 'dogImages/'
train_dir = os.path.join(data_dir, 'train/')
valid_dir = os.path.join(data_dir, 'valid/')
test_dir = os.path.join(data_dir, 'test/')

standard_normalization = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                              std=[0.229, 0.224, 0.225])

data_transforms = {'train': transforms.Compose([transforms.RandomResizedCrop(224),
                                     transforms.RandomHorizontalFlip(),
                                     transforms.ToTensor(),
                                     standard_normalization]),
                   'val': transforms.Compose([transforms.Resize(256),
                                     transforms.CenterCrop(224),
                                     transforms.ToTensor(),
                                     standard_normalization]),
                   'test': transforms.Compose([transforms.Resize(size=(224,224)),
                                     transforms.ToTensor(),
                                     standard_normalization])
                  }


train_data = datasets.ImageFolder(train_dir, transform=data_transforms['train'])
valid_data = datasets.ImageFolder(valid_dir, transform=data_transforms['val'])
test_data = datasets.ImageFolder(test_dir, transform=data_transforms['test'])

train_loader = torch.utils.data.DataLoader(train_data,
                                           batch_size=batch_size,
                                           num_workers=num_workers,
                                           shuffle=True)
valid_loader = torch.utils.data.DataLoader(valid_data,
                                           batch_size=batch_size,
                                           num_workers=num_workers,
                                           shuffle=False)
test_loader = torch.utils.data.DataLoader(test_data,
                                           batch_size=batch_size,
                                           num_workers=num_workers,
                                           shuffle=False)
loaders_scratch = {
    'train': train_loader,
    'valid': valid_loader,
    'test': test_loader
}
```

Figure 5. Data preprocessing code block

Implementation

Human detector: The image is firstly fed to a human detector function to detect if a human face is presented. The pre-trained model haar-cascade classifiers is implemented to achieve this functionality. Figure 6 shows the code block defining the human detector and a sample result.



```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

# extract pre-trained face detector
face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')

# load color (BGR) image
img = cv2.imread(human_files[0])
# convert BGR image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# find faces in image
faces = face_cascade.detectMultiScale(gray)

# print number of faces detected in the image
print('Number of faces detected:', len(faces))

# get bounding box for each detected face
for (x,y,w,h) in faces:
    # add bounding box to color image
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

# convert BGR image to RGB for plotting
cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# display the image, along with bounding box
plt.imshow(cv_rgb)
plt.show()
```

```
Number of faces detected: 1
```

Figure 6. Human detector function and a sample result.

Dog detector: If no human face is detected by the human detector, the image will be passed to the dog detector to see if a dog is presented. The dog detector is constructed using a VGG16 pre-trained model which can identify classes in the 1000 categories. Additionally, VGG16 can be used directly for dog breed identification, however, ResNet50 is going to be used as the pre-trained transfer training model in the final dog breed classifier.

Dog breed classifier: if a dog is detected by the dog detector, the image will be forwarded to the dog breed classifier. In this project, a CNN model for predicting the dog breed is created first. The structure for the scratch CNN model is described below:

1. First convolution layer uses 32 filters, max pooling and stride reduced the image size to 56x56
2. Second convolution layer uses 64 filters, max pooling and stride reduced the image size to 14x14
3. Third convolution layer uses 128 filters and max pooling reduced the image size to 7x7
4. Two linear layers are assigned, with 500 as the output size.
5. Dropout is set to be 0.3 to avoid overfitting.

However, the scratch model is a simple CNN which has a high possibility of not achieving the accuracy metric (60%-90%) that is proposed in this project.

Refinement
Several structures have been tested with the scratch CNN model. Specifically, the number of filters (8 vs 16 in the first layer), pooling and striding (striding=0 vs striding=2). The best accuracy after 100 epochs was 22%. No drastic improvement was found using limited filters variations and pooling strategies. Therefore, it was concluded that the structure of the scratch CNN is too basic for achieving higher accuracy. Because the accuracy metric achieved by the scratch CNN model is a far cry from the proposed accuracy (60%-90%) in this project. Therefore, using a more established pre- trained model makes sense. In this project, ResNet50 pre-trained model is used to serve as the real dog breed classifier. The ResNet50 model is tuned to match the 133 class outputs for this project. The ResNet50 model should theoretically produce much higher accuracy.

Results

Model Evaluation and Validation
After 100 epochs of training, the training loss and validation loss are reduced to 2.64 and 3.79, respectively. The results indicate that the scratch model is too simple in terms of architecture. Additionally, the size of training data can be a caveat to a fresh model like this.
However, by implementing the ResNet50 model, after 20 epochs of training. Training loss and validation loss are reduced to 2.13 and 1.45, which is a significant improvement compared to the scratch CNN model. The transferred model also shows a test loss of 1.52 and an accuracy of 76%. This result suggests a significant advantage of using a pre-trained CNN model in terms of general image classification than building a CNN model from scratch.

Justification
The best prediction accuracy achieved was 76% and the best test loss score was 1.45. These results are still a fair distance away from the proposed performance, however they exceed the 60% threshold that was set, meaning this model performed very well. The following are ways to potentially improve the accuracy:

1. Given more training time and epochs, the accuracy could be improved.
2. Larger input dataset for training can improve the accuracy of the model.
3. Manipulation on the training images to make the input dataset more robust.
4. Play with the layer structures and hyperparameter tuning.
5. Examine the relevance of a different pretrained model.