Convolutional Neural Network for Dog Breed Classification

Udacity Machine Learning Engineer Nanodegree Capstone Project

Written by **Alex Augusto Costa Machado** on **December 10th 2021**

1 Definition

1.1 Project Overview

For computer vision tasks, such as a dog breed classification, an algorithm mostly used is a Convolutional Neural Network, or CNN for short, which is a model that extracts features, like texture and edges, from spatial data.

The history behind this algorithm started during the 1950s, but it is around the year 2012 that CNNs saw a huge surge in popularity after a CNN called AlexNet achieved state-of-the-art performance labeling pictures in the ImageNet challenge. Alex Krizhevsky et al. published the paper "ImageNet Classification with Deep Convolutional Neural Networks" describing the winning AlexNet model.

1.2 Problem Statement

The purpose of this project is to use a Convolutional Neural Network to classify dog breeds using images as input. For images that contain a human instead of a dog, the algorithm should display which dog breed resembles the human in the picture.

1.3 Metrics

The metric to evaluate the quality of the classifier is the accuracy of the dog breed predictions. The accuracy can be used in this case because the training data is not imbalanced. The definition of prediction accuracy is as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

2 Analysis

2.1 Data Exploration

There are two datasets for this project: one is a dog dataset containing 8351 images of 133 different dog breeds, and the other is a human dataset consisting of 13233 photos of 5749 different people. The dog dataset is divided in train, validation, and test datasets.

A sample of the dog dataset is presented in the Figure 1.

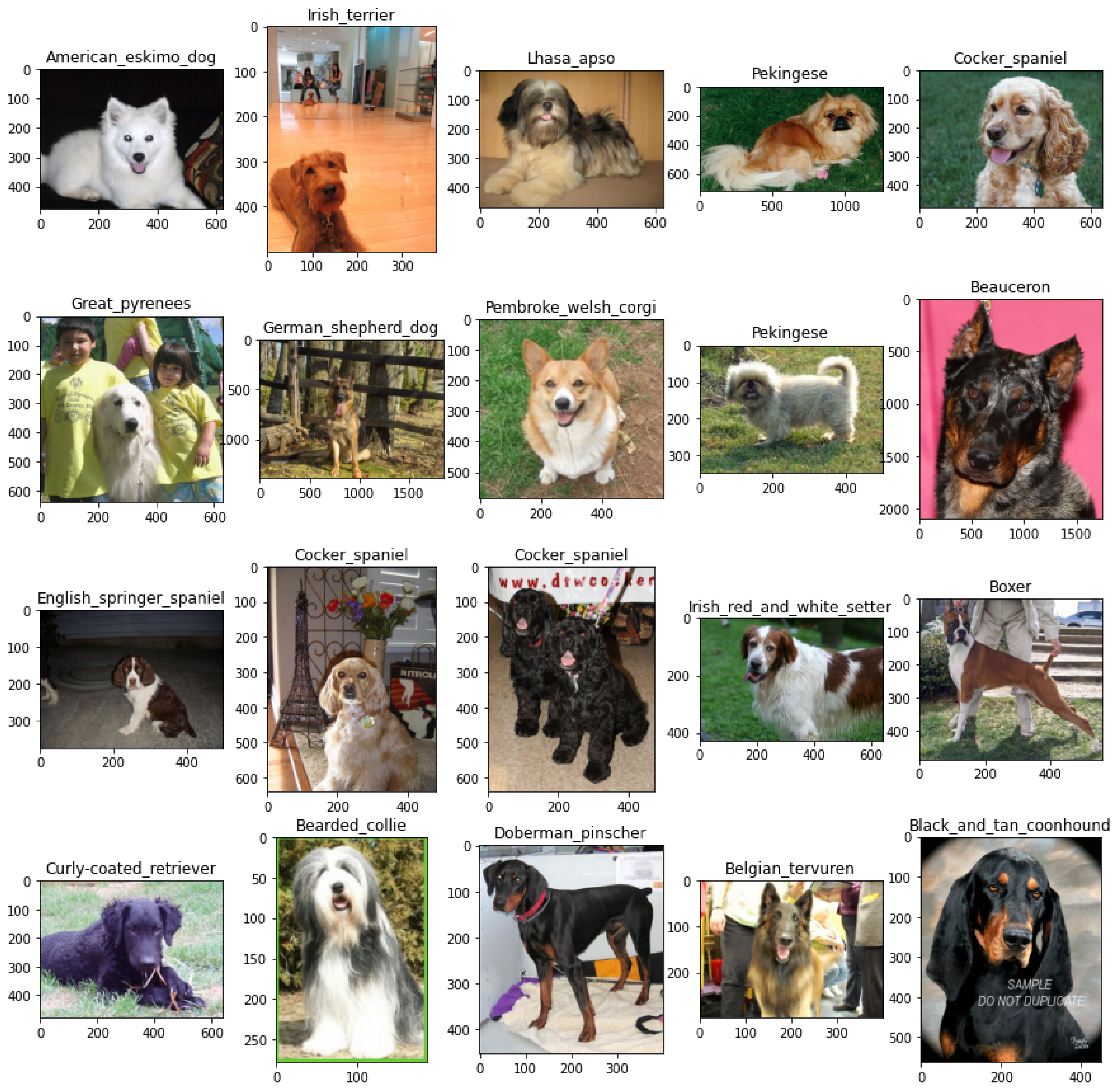


Figure 1: Dog dataset sample.

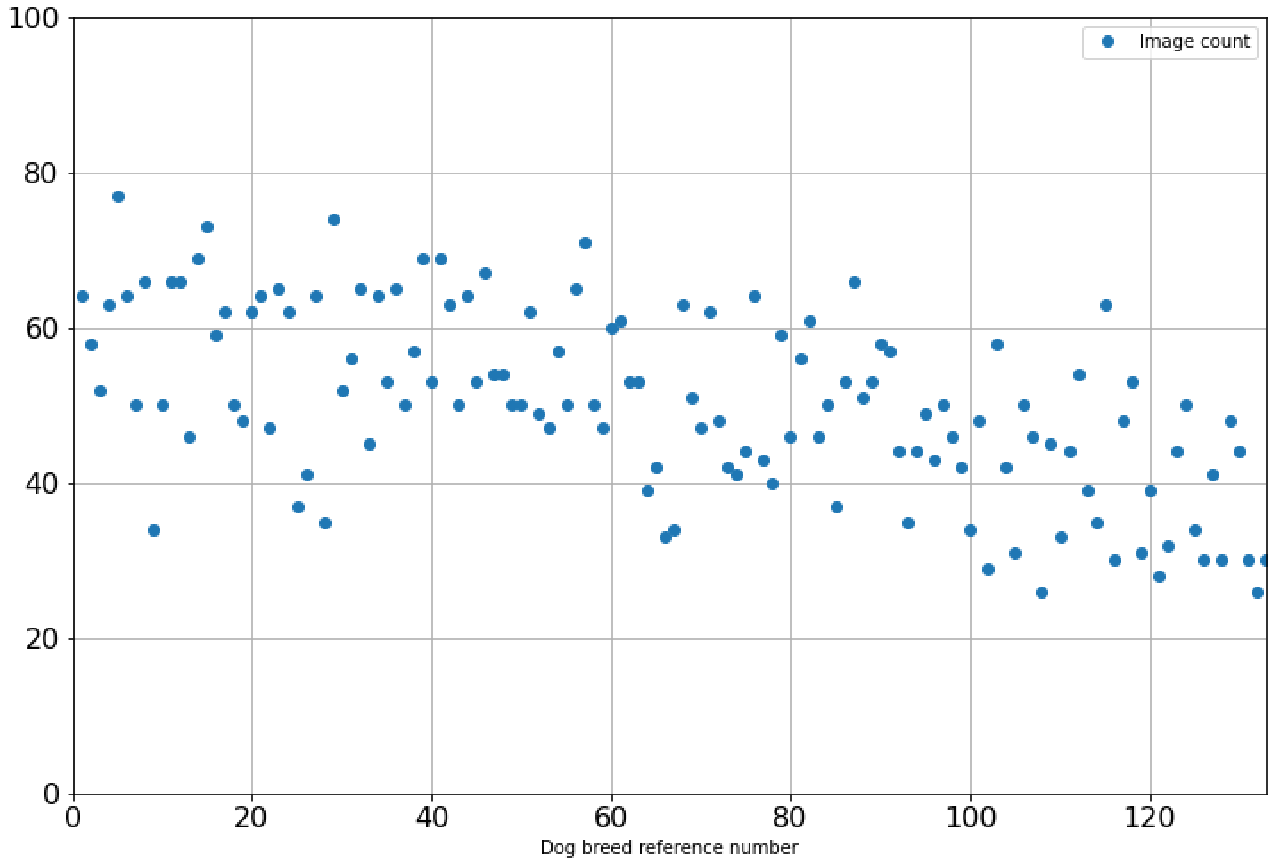


Figure 2: Number of images for each dog breed.

2.2 Exploratory Visualization

A way to explore this image dataset is to verify how many images there are for each of the dog breeds. The Figure 2 presents a graph of image count for each dog breed.

The Table 1 shows some statistical analysis of the number of images for each dog breed.

	Value
Minimum	26
Maximum	77
Mean	50.23
Median	50

Table 1: Statistical analysis of data.

2.3 Algorithms and Techniques

Three main algorithms will be used for this project: a human face detector, a dog detector, and a dog breed classifier.

For the human face detector, I used an OpenCV's implementation of Haar feature-based cascade classifier. The pre-trained face detector is stored as XML file on GitHub.

The dog detector and the dog breed classifier use VGG (Visual Geometry Group) pre-trained models with weights that have been trained on ImageNet, a very popular dataset used for image classification and other vision tasks.

More about these two techniques is presented on the following sections.

2.3.1 Haar Feature-Based Cascade Classifiers

Haar feature-based cascade classifiers is an effective method for object detection proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then, the features are extracted from it using Haar features presented on Figure 3. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.

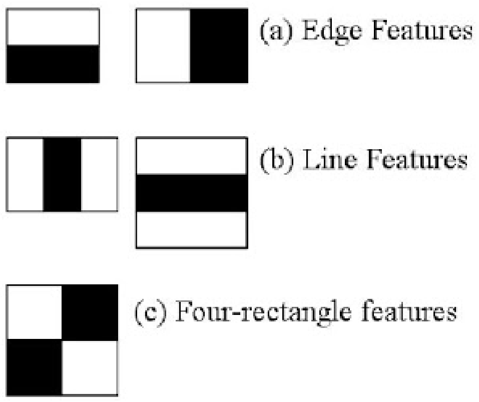


Figure 3: Haar features.

For each feature, it finds the best threshold which will classify the faces to positive and negative. As there are errors or misclassifications, the features with minimum error rate are selected, which means they are the features that most accurately classify the face and non-face images. The final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper mentioned above states that even 200 features provide detection with 95% accuracy.

2.3.2 Visual Geometry Group

Before talking about VGG, it is important to comment about AlexNet, which is a CNN used for image recognition that came out in 2012. It takes into account overfitting by using data augmentation and dropout. It replaces tanh activation function with ReLU by encapsulating its distinct features for over-pooling.

VGG came into picture as it addresses the depth of CNNs. It is a pre-trained model and contains the weights that represent the features of whichever dataset it was trained on. More about the VGG architecture is presented below:

- Input: VGG takes in a 224x224 pixel RGB image. For the ImageNet competition, the authors cropped out the center 224x224 patch in each image to keep the input image size consistent.
- Convolutional Layers: the convolutional layers in VGG use a very small receptive field (3x3, the smallest possible size that still captures left/right and up/down). There are also 1x1 convolution filters which act as a linear transformation of the input, which is followed by a ReLU unit. The convolution stride is fixed to 1 pixel so that the spatial resolution is preserved after convolution.
- Fully-Connected Layers: VGG has three fully-connected layers: the first two have 4096 channels each and the third has 1000 channels, 1 for each class.
- Hidden Layers: all of VGG's hidden layers use ReLU. VGG does not generally use Local Response Normalization (LRN), as LRN increases memory consumption and training time with no particular increase in accuracy.

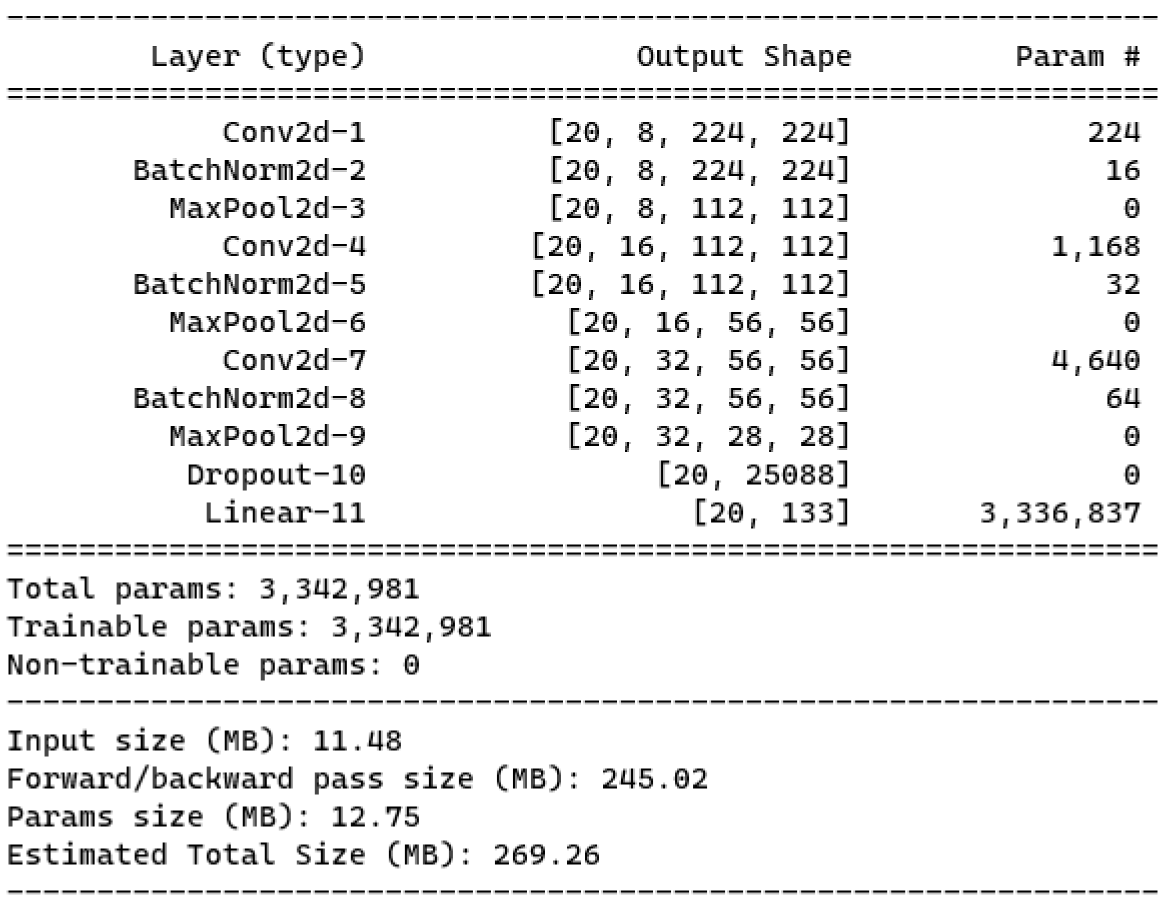
VGG, while based off of AlexNet, has several differences that separates it from other competing models:

- Instead of using large receptive fields like AlexNet (11x11 with a stride of 4), VGG uses very small receptive fields (3x3 with a stride of 1). Because there are now three ReLU units instead of just one, the decision function is more discriminative.
- VGG incorporates 1x1 convolutional layers to make the decision function more non-linear without changing the receptive fields.
- The small-size convolution filters allows VGG to have a large number of weight layers.

2.4 Benchmark

The benchmark for the dog breed classifier will be a simple CNN model designed from scratch that will be compared to the final CNN obtained from transfer learning using a pre-trained VGG-11 model.

The baseline model contains three convolutional blocks, each presenting a convolutional layer, a batch normalization layer, and a pooling layer. After these blocks, there is a dropout layer and a fully connected linear layer. A summary of the model is presented on Figure 4.



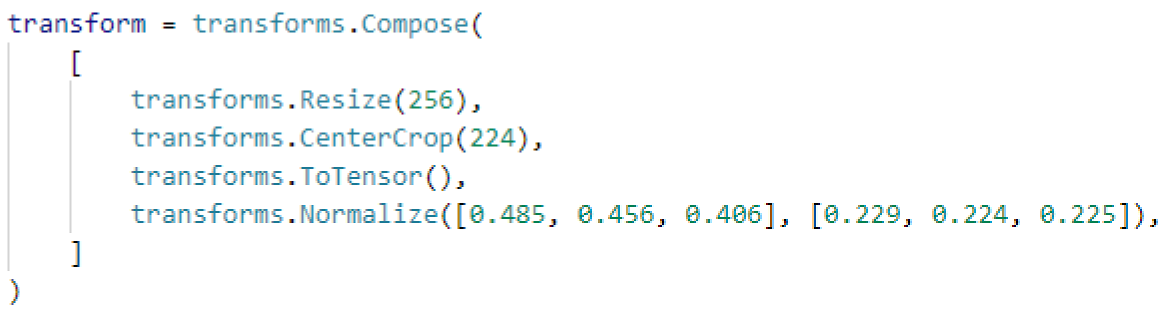
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [20, 8, 224, 224]             224
       BatchNorm2d-2         [20, 8, 224, 224]              16
         MaxPool2d-3         [20, 8, 112, 112]               0
            Conv2d-4        [20, 16, 112, 112]           1,168
       BatchNorm2d-5        [20, 16, 112, 112]              32
         MaxPool2d-6          [20, 16, 56, 56]               0
            Conv2d-7          [20, 32, 56, 56]           4,640
       BatchNorm2d-8          [20, 32, 56, 56]              64
         MaxPool2d-9          [20, 32, 28, 28]               0
          Dropout-10               [20, 25088]               0
           Linear-11                 [20, 133]       3,336,837
================================================================
Total params: 3,342,981
Trainable params: 3,342,981
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 11.48
Forward/backward pass size (MB): 245.02
Params size (MB): 12.75
Estimated Total Size (MB): 269.26
----------------------------------------------------------------
```

Figure 4: Summary of the benchmark model.

3 Methodology

3.1 Data Preprocessing

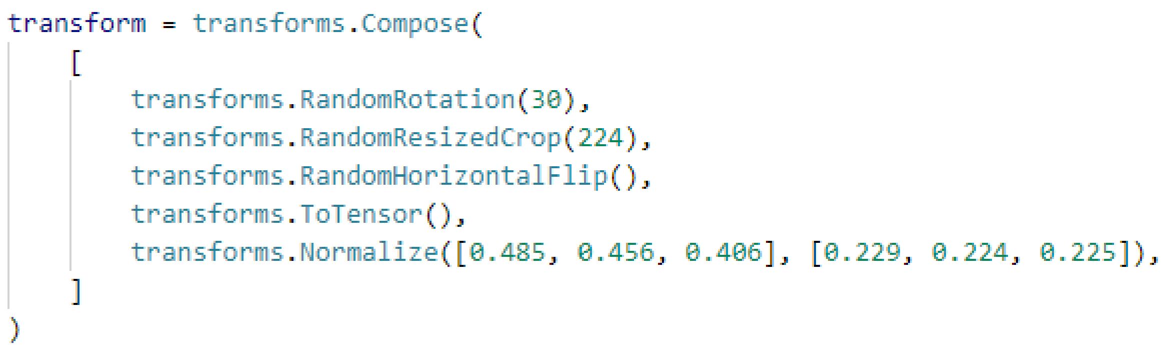
Since the images have all different sizes, it was necessary to resize them. The images on the validation and testing datasets were resized to 256 pixels and cropped to square shapes of 224x224 pixels about the center and transformed into tensors, then the resulting tensors were normalized. A code snippet is presented on Figure 5.



```
transform = transforms.Compose(
    [
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
    ]
)
```

Figure 5: Code for validation and testing transform.

For the training data, in addition the resizing, cropping and normalization, data augmentation was applied, so the images were also randomly rotated and horizontally flipped. The Figure 6 presents the code used for this transformation.



```
transform = transforms.Compose(
    [
        transforms.RandomRotation(30),
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
    ]
)
```

Figure 6: Code for training transform.

3.2 Implementation

3.2.1 Human Face Detector

For the implementation of this algorithm I used an OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

The cascade classifier was downloaded from OpenCV's GitHub page and attached to a model. Then, the images were read and converted to grayscale. These images were then passed through to the model so that it could look up for faces in the pictures.

After this, a bounding box was drawn around each face and the images were converted back to RGB so they could be plotted.

No hyperparameters were changed for this, since there was no training needed for this model.

3.2.2 Dog Detector

A pre-trained VGG-16 model was used for this implementation. This model was downloaded along with weights that have been trained on ImageNet, a very large, very popular dataset used for image classification and other vision tasks. ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories.

No hyperparameters were changed for this, since there was no training needed for this model. Before sending the images to the model, a set of transformations were necessary:

- Resize to 256 pixel.
- Crop to a square of 224x224 pixels.
- Convert to tensor.
- Normalize tensors.

After this, the model would return one of the 1000 indices of classification. Looking at the dictionary, one can notice that the categories corresponding to dogs appear in an uninterrupted sequence and correspond to dictionary keys 151-268, inclusive, to include all categories from "Chihuahua" to "Mexican hairless". Thus, in order to check to see if an image is predicted to contain a dog by the pre-trained VGG-16 model, I needed only to check if the pre-trained model predicts an index between 151 and 268 (inclusive).

3.2.3 Dog Breed Classifier CNN from scratch

This dog breed classifier is a simple CNN model designed from scratch that will be compared to the final CNN obtained from transfer learning using a pre-trained VGG-11 model.

At first, I started with one convolutional block consisting of a convolutional layer, a batch normalization layer and a pooling layer. The output of this block would pass through a dropout layer and finally a fully connected layer. I then realized that adding more convolutional blocks increasing the output channels would improve the performance, so I ended up with three of these convolutional blocks. The code for this model is presented on Figure 7.

The hyperparameters used for this model are listed on Table 2.

Hyperparameter	Value
input-dim	3
conv1-out-dim	8
conv2-out-dim	16
conv3-out-dim	32
conv-kernel-size	3
pool-kernel-size	2
drop-prob	0.5
hidden-dim	25088
output-dim	133

Table 2: Hyperparameters for CNN model from scratch.

For the loss function I used the cross entropy loss, and for optimization I used SGD with lr = 0.001 and momentum = 0.9.

Each epoch of the training algorithm consists of a training step and a validation step, if the model presents a reduction on the validation loss, then the new parameters are saved.

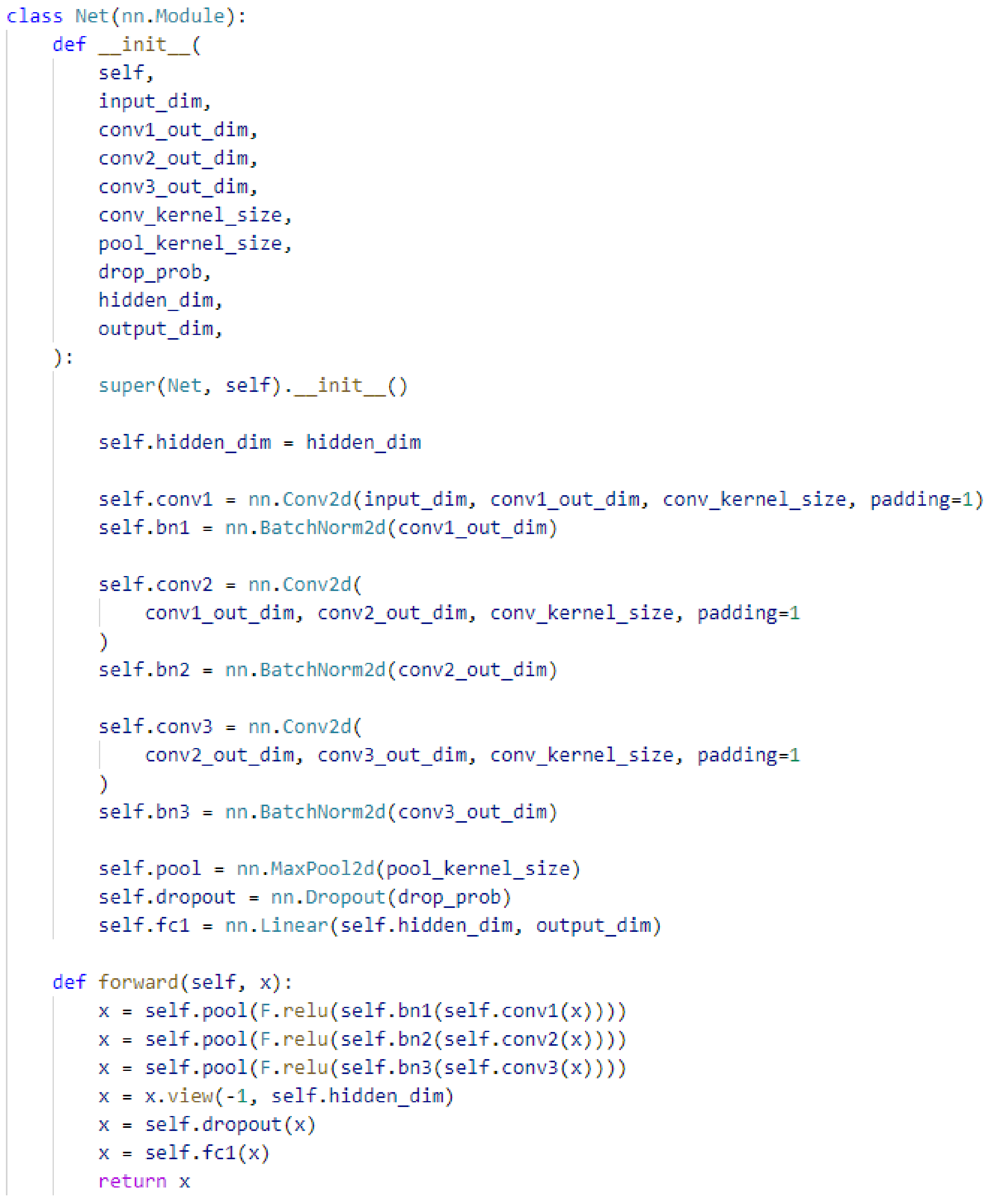
This model was trained for 80 epochs, reaching a validation loss of 3.561250 and an accuracy of 20% when exposed to the test dataset.

3.2.4 Dog Breed Classifier CNN with Transfer Learning

A pre-trained VGG-11 model was used for this implementation. This model was downloaded along with weights that have been trained on ImageNet.

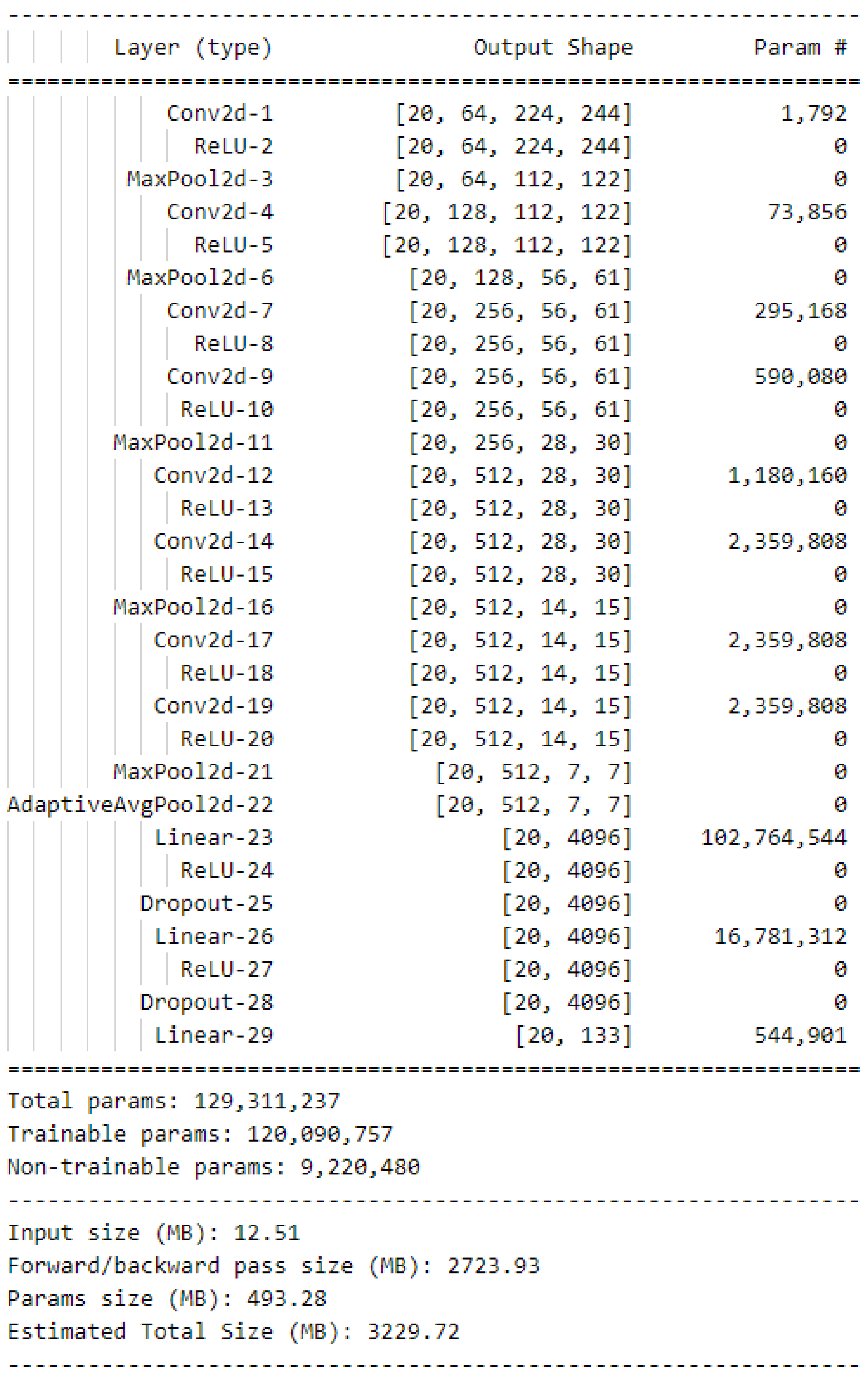
After loading the model, it was necessary to freeze the existing parameters as to avoid destroying any of the information they contain during future training rounds.

The final layer was overwritten, changing the output dimension to 133, which is the number of dog breeds in the training dataset. A summary for this model is presented on Figure 8.



```python
class Net(nn.Module):
    def __init__(
        self,
        input_dim,
        conv1_out_dim,
        conv2_out_dim,
        conv3_out_dim,
        conv_kernel_size,
        pool_kernel_size,
        drop_prob,
        hidden_dim,
        output_dim,
    ):
        super(Net, self).__init__()

        self.hidden_dim = hidden_dim

        self.conv1 = nn.Conv2d(input_dim, conv1_out_dim, conv_kernel_size, padding=1)
        self.bn1 = nn.BatchNorm2d(conv1_out_dim)

        self.conv2 = nn.Conv2d(
            conv1_out_dim, conv2_out_dim, conv_kernel_size, padding=1
        )
        self.bn2 = nn.BatchNorm2d(conv2_out_dim)

        self.conv3 = nn.Conv2d(
            conv2_out_dim, conv3_out_dim, conv_kernel_size, padding=1
        )
        self.bn3 = nn.BatchNorm2d(conv3_out_dim)

        self.pool = nn.MaxPool2d(pool_kernel_size)
        self.dropout = nn.Dropout(drop_prob)
        self.fc1 = nn.Linear(self.hidden_dim, output_dim)

    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 = x.view(-1, self.hidden_dim)
        x = self.dropout(x)
        x = self.fc1(x)
        return x
```

Figure 7: Code for CNN model made from scratch.



```
----------------------------------------------------------------
|  |  |  |  Layer (type)               Output Shape         Param #
================================================================
|  |  |  |  |  |  Conv2d-1       [20, 64, 224, 244]           1,792
|  |  |  |  |  |  |  ReLU-2      [20, 64, 224, 244]               0
|  |  |  |  |  MaxPool2d-3       [20, 64, 112, 122]               0
|  |  |  |  |  |  Conv2d-4      [20, 128, 112, 122]          73,856
|  |  |  |  |  |  |  ReLU-5     [20, 128, 112, 122]               0
|  |  |  |  |  MaxPool2d-6        [20, 128, 56, 61]               0
|  |  |  |  |  |  Conv2d-7       [20, 256, 56, 61]          295,168
|  |  |  |  |  |  |  ReLU-8      [20, 256, 56, 61]               0
|  |  |  |  |  |  Conv2d-9       [20, 256, 56, 61]          590,080
|  |  |  |  |  |  |  ReLU-10     [20, 256, 56, 61]               0
|  |  |  |  |  MaxPool2d-11       [20, 256, 28, 30]               0
|  |  |  |  |  |  Conv2d-12      [20, 512, 28, 30]        1,180,160
|  |  |  |  |  |  |  ReLU-13     [20, 512, 28, 30]               0
|  |  |  |  |  |  Conv2d-14      [20, 512, 28, 30]        2,359,808
|  |  |  |  |  |  |  ReLU-15     [20, 512, 28, 30]               0
|  |  |  |  |  MaxPool2d-16       [20, 512, 14, 15]               0
|  |  |  |  |  |  Conv2d-17      [20, 512, 14, 15]        2,359,808
|  |  |  |  |  |  |  ReLU-18     [20, 512, 14, 15]               0
|  |  |  |  |  |  Conv2d-19      [20, 512, 14, 15]        2,359,808
|  |  |  |  |  |  |  ReLU-20     [20, 512, 14, 15]               0
|  |  |  |  |  MaxPool2d-21         [20, 512, 7, 7]               0
AdaptiveAvgPool2d-22                 [20, 512, 7, 7]               0
|  |  |  |  |  |  Linear-23             [20, 4096]     102,764,544
|  |  |  |  |  |  |  ReLU-24            [20, 4096]               0
|  |  |  |  |  |  Dropout-25            [20, 4096]               0
|  |  |  |  |  |  Linear-26             [20, 4096]      16,781,312
|  |  |  |  |  |  |  ReLU-27            [20, 4096]               0
|  |  |  |  |  |  Dropout-28            [20, 4096]               0
|  |  |  |  |  |  Linear-29              [20, 133]         544,901
================================================================
Total params: 129,311,237
Trainable params: 120,090,757
Non-trainable params: 9,220,480
----------------------------------------------------------------
Input size (MB): 12.51
Forward/backward pass size (MB): 2723.93
Params size (MB): 493.28
Estimated Total Size (MB): 3229.72
----------------------------------------------------------------
```

Figure 8: Summary of the final model.

For the loss function I used the cross entropy loss, and for optimization I used SGD with lr = 0.001 and momentum = 0.9.

The training algorithm is the same as in the previous section: each epoch of consists of a training step and a validation step, every time the model presents a reduction on the validation loss, the new parameters are saved.

This model was trained for 20 epochs, reaching a validation loss of 0.534802 and an accuracy of 86% when exposed to the test dataset.

3.3 Refinement

When using transfer learning with the VGG-11 pre-trained model, at first it only reached 46% of accuracy. Even trying to change the hyperparameters and batch size would make no difference on the final accuracy. What really made it reach 86% was shuffling the training data before feeding it to the model.

4 Results

4.1 Model Evaluation and Validation

The model achieved a prediction accuracy of 86% when exposed to the test dataset. Using images of dogs chosen randomly on the Internet, I obtained the results presented on Table 3. Only two cases did not obtain the expected result (pictures 5 and 6), but the predicted dog breed is very similar to the target breed.

On Figure 9 we can see the images for both cases. A Shih tzu (Figure 9a) was provided as input, but a Lhasa apso (Figure 9b) was predicted. The same happened for the Staffordshire bull terrier (Figure 9c) used as input and the American staffordshire terrier (Figure 9d) received as prediction.

	Target	**Prediction**
Picture 1	Alaskan malamute	Alaskan malamute
Picture 2	French bulldog	French bulldog
Picture 3	Afghan hound	Afghan hound
Picture 4	Boxer	Boxer
Picture 5	Shih tzu	Lhasa apso
Picture 6	Staffordshire bull terrier	American staffordshire terrier

Table 3: Comparison between target values and prediction results.

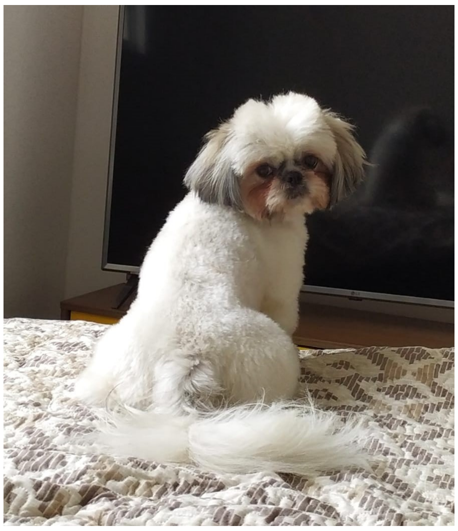
4.2 Justification

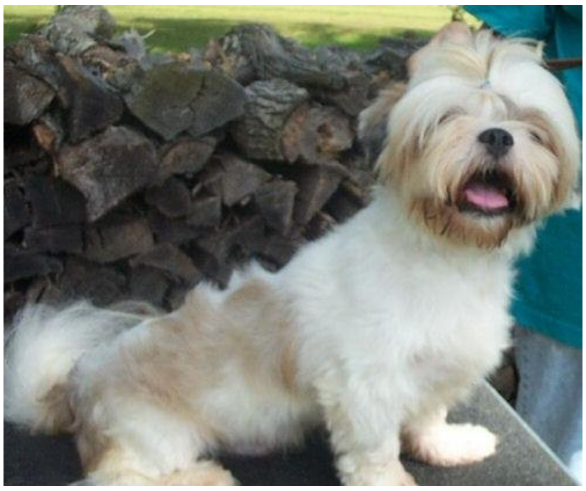
The Table 4 presents a comparison between the results obtained for the proposed model and the benchmark model.

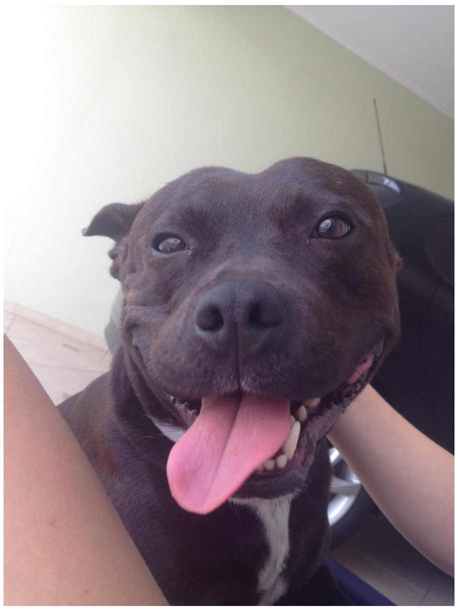
	Benchmark model	**Proposed model**
Epochs	80	20
Training loss	3.437396	0.830221
Validation loss	3.561250	0.534802
Test accuracy	20%	86%

Table 4: Comparison between benchmark and proposed models.

It is possible to see that the proposed model performs a lot better than the benchmark model. As we used transfer learning, the model only required 20 epochs to reach an accuracy of 86%, which is a satisfactory result.


(a) Target: Shih tzu


(b) Prediction: Lhasa apso


(c) Target: Staffordshire bull terrier

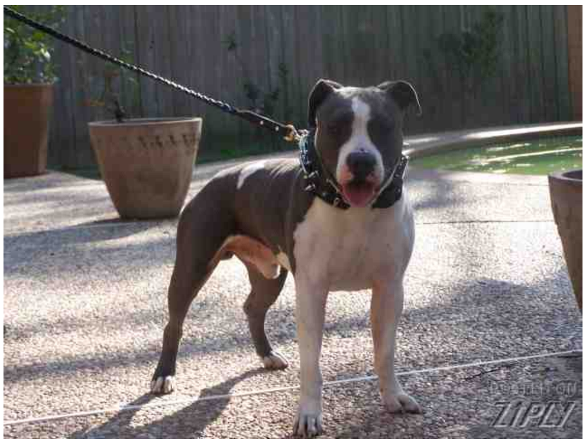

(d) Prediction: American staffordshire terrier

Figure 9: Comparison for wrongly classified dog breeds.

5 Conclusion

5.1 Reflection

This project was very important for me to understand the different steps of creating a machine learning model. From data exploration, to preprocessing, training, and testing, all the steps were included in this project.

The most difficulty I had was making the configuration of the Convolutional Neural Network for the Python program to run correctly without any exceptions being raised.

5.2 Improvement

Some possible points of improvement are:

- Add functionality to detect dog mutts;
- Improve human face detection, as there were human faces detected on some dog images;
- Change the algorithm to try and reach over 90% accuracy.