Machine Learning Engineer Nanodegree

Capstone Project

Dog Breed Classifier

Abdolmajid Yolmeh

1. Problem Definition

The image classification task is to predict a label to a given image. This domain has seen many applications in various areas such as identifying hand-written digits [1], face recognition [2], and early disease diagnosis via medical imaging [3]. In this domain, neural networks such as the convolutional neural networks (CNNs), have been successfully used and are considered state of the art [4]. In this project, we will use deep neural networks to identify dog breeds based on input images.

1.1 Problem Statement

The goal of this project is to use machine learning techniques to accept an image as input and if a dog is detected in the image, predict the dog's breed. If the input image is of a human, the program will determine which dog breed the human is most similar to. To achieve this goal, we use different machine learning techniques such as convolutional neural networks and transfer learning.

1.2 Evaluation Metrics

The evaluation metric in this project will be the accuracy of the model. The accuracy is defined as the percentage of correct predictions:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

We chose this metric because in this project we do not differentiate between misclassifications. In other words, all misclassification errors have the same cost.

2. Problem Analysis

2.1 Data exploration and visualization

For this project, we will use the dog data set and human data set provided by Udacity.

- The dog data set contains 8351 dog images and is available in this link: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip
- The human data set contains 13233 human images and is available in this link: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip

The dog data set contains 8351 of dog images from 133 breeds. The data set is not balanced in terms of number of data points in each class. The minimum number of images in a class is 33, maximum is 96, and the median is 62. Figure 1 shows the histogram of number of images in each class.

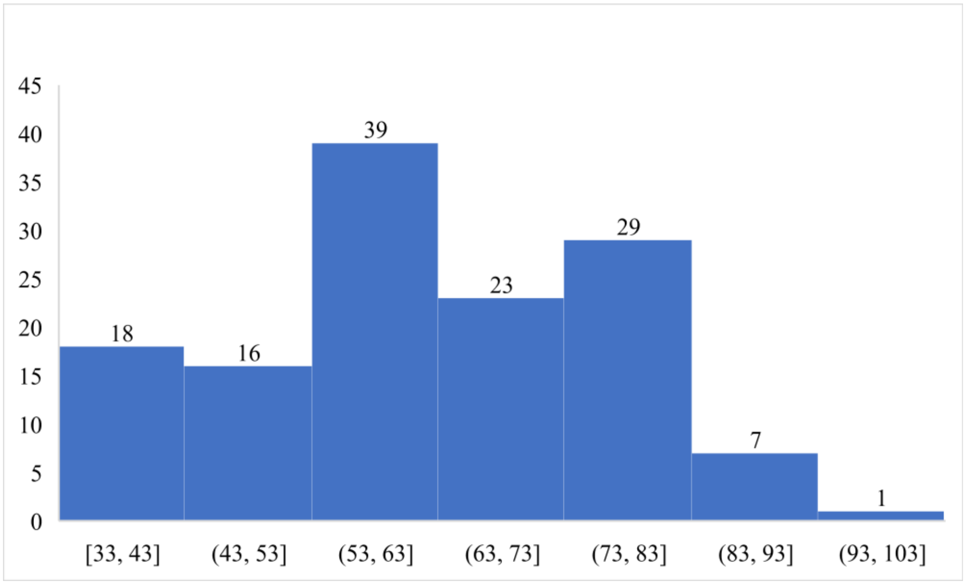


Figure 1 Histogram of number of images in each class

The task of predicting dog breed based in an input image can be challenging. This is true even for a human classifier. Some of the dog breeds look very similar to each other with only small visual differences. For example, Figures 2 and 3 show an English Springer Spaniel and a Welsh Springer Spaniel, respectively. As seen in these figures, it is not very easy to find clear visual features that distinguish the two breeds from each other.

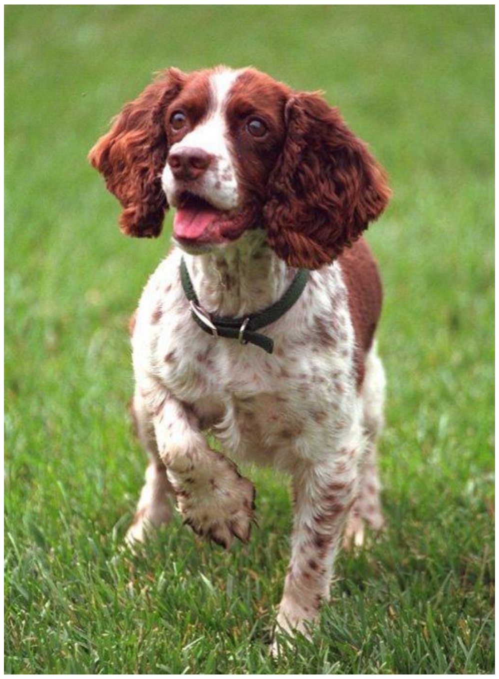


Figure 2 English Springer Spaniel

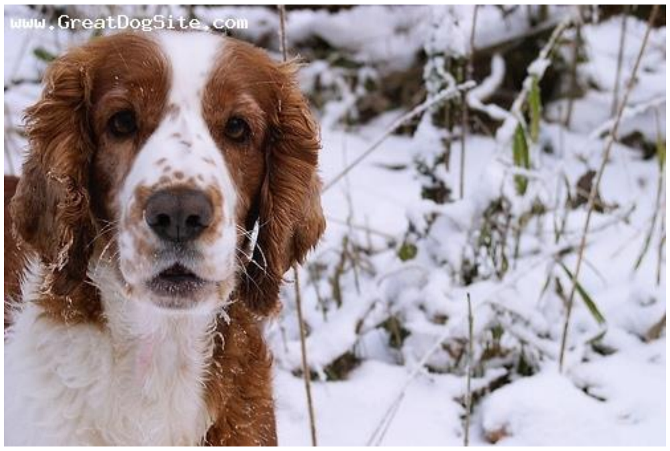


Figure 3 Welsh Springer Spaniel

Figures 4 and 5 show a less challenging example of differentiating Bichon Frise from white poodles. As seen in these figures, the only distinguishing feature seems to be the slightly longer nose of poodles.

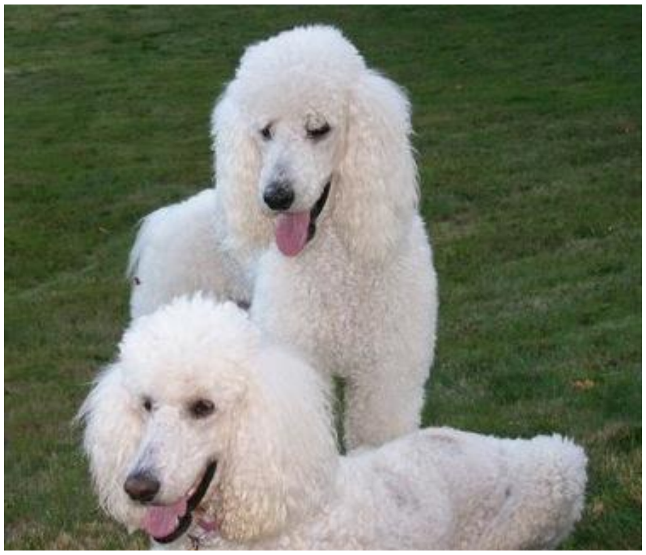


Figure 4 Poodle

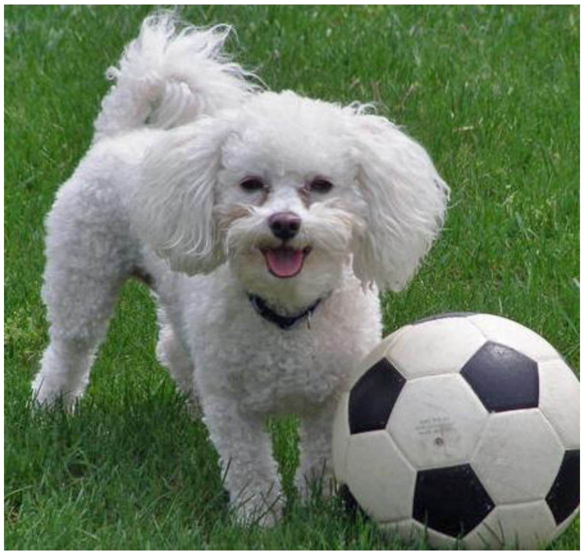


Figure 5 Bichon Frise

Another challenge is that the dogs from the same breed seem to look quite different from each other. For example, both of the images shown in Figures 6 and 7 belong to the same breed of Affenpinscher.

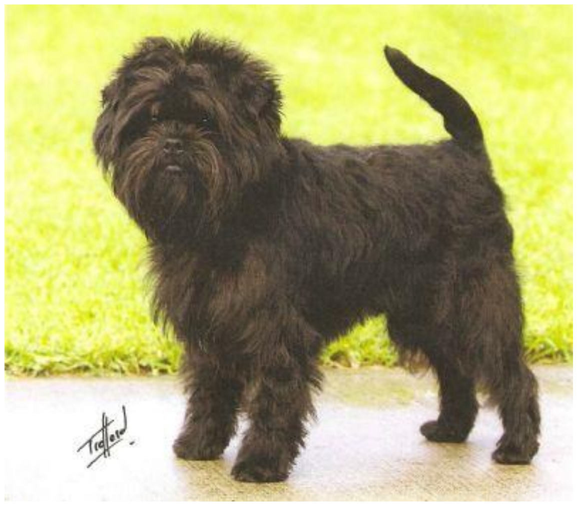


Figure 6 Affenpinscher

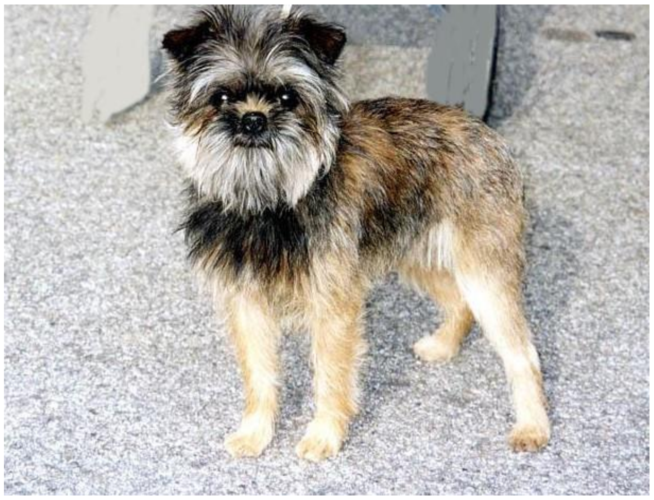


Figure 7 Affenpinscher

Another challenge for a human classifier is the number of classes. It is hard to identify the exact breed of a dog out of 133 possible breeds.

2.2 Algorithms and Techniques

The first step in this project is to detect human faces. To achieve this, we use an implementation of the Haar feature-based cascade classifiers in OpenCV. The information about this implementation is available in this link: https://docs.opencv.org/trunk/db/d28/tutorial_cascade_classifier.html

We use a pretrained face detector provided by OpenCV as an XML file to detect human faces in a given image.

The next step is to detect dogs in the image. To do this, we use a pretrained model called VGG-16. The VGG-16 model gives a prediction out of 1000 possible categories for the object in the image.

After being able to determine if a given image contains a human or dog, the next step is to identify the breed of the dog, if the image contains a dog. To achieve this, two approaches are considered. The first approach is to train a deep neural network from scratch. The second approach is to use the transfer learning technique to get a better performance. Transfer learning technique uses the knowledge gained from solving one problem and applies it to a different but related problem. In this project, we use the pretrained VGG-16 model to perform transfer learning. We expect this to enhance the performance of the model because the VGG16 network was trained on dogs, among other things, and contains useful information about high-level feature of dogs.

2.3 Benchmark Model

We consider the regular training from scratch approach as the benchmark model because it is the traditional approach to solve image classification problems. In other words, the benchmark model is a convolutional neural network and no transfer learning technique will be used for this model.

3. Implementation

3.1 Data Preprocessing and Augmentation

We will conduct various data preprocessing and augmentation approaches such image resizing, random rotations, and flipping.

All of the images undergo a resizing to 256x256 pixel and a cropping from the center to obtain a 224x224 pixel image:

```
>> import torchvision.transforms as transforms

>> transforms.Resize(256)

>> transforms.CenterCrop(224)
```

This is done because most of the pretrained model in PyTorch take a 224x224 pixel image as input. This will be useful when using pretrained models to detect dogs and humans. This preprocessing will also be useful in applying transfer learning using pretrained models.

For training procedures, additional data augmentations are applied. Specifically, random rotations, and flipping are applied:

>> transforms.RandomHorizontalFlip()

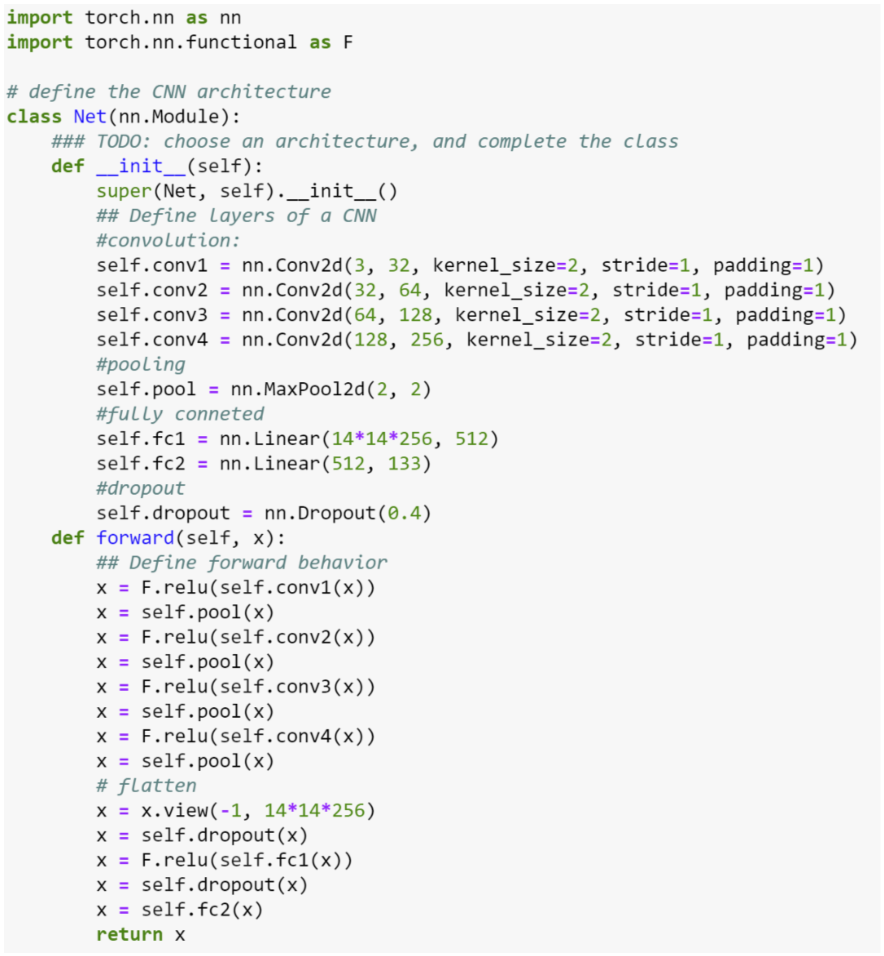
>> transforms.RandomRotation(15)

3.2 Data Splitting

We split the data set of 8351 dog images into training set (80%), validation set (10%), and test set (10%).

3.3 Model Architecture

We consider two approaches in developing neural network models to predict dog breeds. The first approach, which is considered the benchmark approach, is to train the network from scratch. The neural network architecture for this model consists of two parts: feature generation part and classification part. The feature generation part of the model consists of four convolutional layers and the classification part consists of two fully connected layers. The code shown here characterizes this structure:



```
import torch.nn as nn
import torch.nn.functional as F

# define the CNN architecture
class Net(nn.Module):
    ### TODO: choose an architecture, and complete the class
    def __init__(self):
        super(Net, self).__init__()
        ## Define layers of a CNN
        #convolution:
        self.conv1 = nn.Conv2d(3, 32, kernel_size=2, stride=1, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=2, stride=1, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=2, stride=1, padding=1)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=2, stride=1, padding=1)
        #pooling
        self.pool = nn.MaxPool2d(2, 2)
        #fully conneted
        self.fc1 = nn.Linear(14*14*256, 512)
        self.fc2 = nn.Linear(512, 133)
        #dropout
        self.dropout = nn.Dropout(0.4)
    def forward(self, x):
        ## Define forward behavior
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = self.pool(x)
        x = F.relu(self.conv4(x))
        x = self.pool(x)
        # flatten
        x = x.view(-1, 14*14*256)
        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

The second approach is to use transfer learning. In this approach a pretrained model, VGG16, is used for the feature extraction part of the network and only the classification portion of the network is trained.

The VGG16 network consists of two parts: feature extraction part and classification part. Because VGG16 was trained on dog pictures, it is very useful for our case and we use the feature extraction part as is and fix the parameters of the model for this part. We then add a classifier on top of the feature extraction part and the training is only done on the parameters of the classification part. The classification part contains three fully connected layers with two dropout layers in between to prevent overfitting. The size of the last layer corresponds to the number of classes. This architecture is shown in the following code:

```
import torchvision.models as models
import torch.nn as nn

## TODO: Specify model architecture

model_transfer = models.vgg16(pretrained=True)

# fix the parameters of vgg16 model:
for param in model_transfer.parameters():
    param.requires_grad = False

classifier = nn.Sequential(
    nn.Linear(25088, 8192),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(8192, 1024),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(1024, 133)
)

model_transfer.classifier = classifier
```

3.4 Model Training and Evaluation

Both models are trained with the objective of minimizing binary cross entropy. The Adaptive Moment Estimation (Adam) optimizer has been used for both models.

Here is a screen shot of training results for the first approach (learning from scratch):

```
Epoch: 1        Training Loss: 4.854072         Validation Loss: 4.695496
Validation loss has decreased from inf to 4.69550, Saving the model ...
Epoch: 2        Training Loss: 4.549758         Validation Loss: 4.445070
Validation loss has decreased from 4.69550 to 4.44507, Saving the model ...
Epoch: 3        Training Loss: 4.337314         Validation Loss: 4.267756
Validation loss has decreased from 4.44507 to 4.26776, Saving the model ...
Epoch: 4        Training Loss: 4.187451         Validation Loss: 4.174553
Validation loss has decreased from 4.26776 to 4.17455, Saving the model ...
Epoch: 5        Training Loss: 4.030789         Validation Loss: 4.064877
Validation loss has decreased from 4.17455 to 4.06488, Saving the model ...
Epoch: 6        Training Loss: 3.908982         Validation Loss: 3.945243
Validation loss has decreased from 4.06488 to 3.94524, Saving the model ...
Epoch: 7        Training Loss: 3.795979         Validation Loss: 3.878996
Validation loss has decreased from 3.94524 to 3.87900, Saving the model ...
Epoch: 8        Training Loss: 3.689729         Validation Loss: 3.859506
Validation loss has decreased from 3.87900 to 3.85951, Saving the model ...
Epoch: 9        Training Loss: 3.553748         Validation Loss: 3.873094
Epoch: 10       Training Loss: 3.463057         Validation Loss: 3.815440
Validation loss has decreased from 3.85951 to 3.81544, Saving the model ...
Epoch: 11       Training Loss: 3.364032         Validation Loss: 3.923247
Epoch: 12       Training Loss: 3.264459         Validation Loss: 3.804316
Validation loss has decreased from 3.81544 to 3.80432, Saving the model ...
Epoch: 13       Training Loss: 3.132136         Validation Loss: 3.850276
Epoch: 14       Training Loss: 3.044513         Validation Loss: 3.809372
Epoch: 15       Training Loss: 2.952956         Validation Loss: 3.779408
Validation loss has decreased from 3.80432 to 3.77941, Saving the model ...
Epoch: 16       Training Loss: 2.849008         Validation Loss: 3.831933
Epoch: 17       Training Loss: 2.766817         Validation Loss: 3.916162
Epoch: 18       Training Loss: 2.667743         Validation Loss: 3.826327
Epoch: 19       Training Loss: 2.548138         Validation Loss: 3.937912
Epoch: 20       Training Loss: 2.475390         Validation Loss: 3.931740
```

Figure 8 shows training and validation loss for different epochs. As seen in this figure, after around 9 epochs, validation loss does not improve anymore.

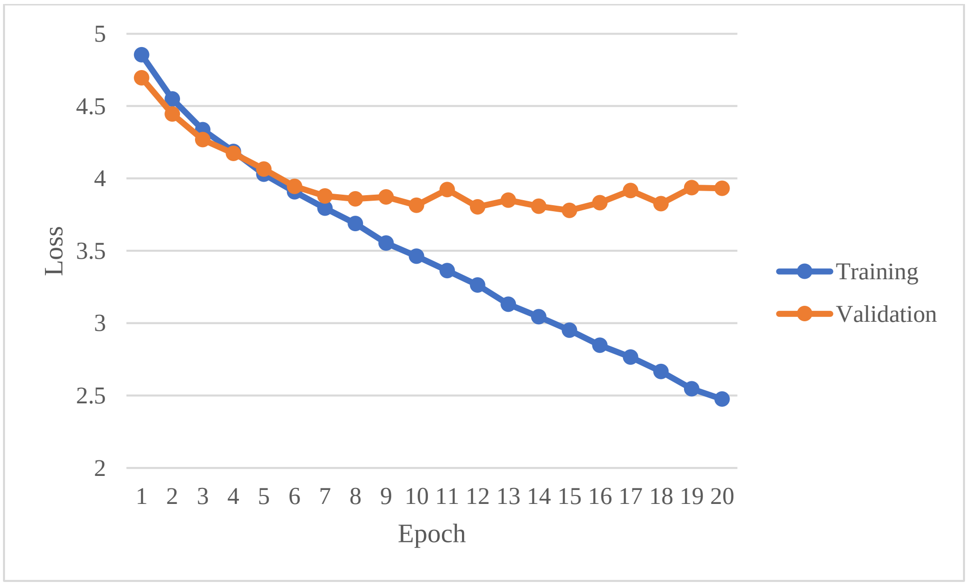
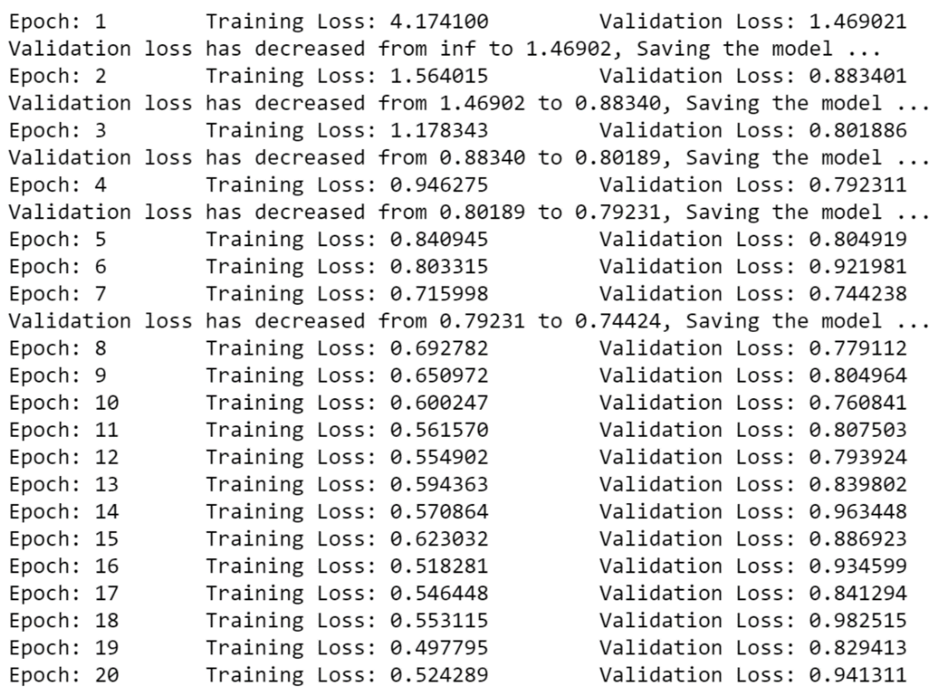


Figure 8 Training and validation loss over epochs

Here is a screen shot of training results for the first approach (learning from scratch):



```
Epoch: 1        Training Loss: 4.174100         Validation Loss: 1.469021
Validation loss has decreased from inf to 1.46902, Saving the model ...
Epoch: 2        Training Loss: 1.564015         Validation Loss: 0.883401
Validation loss has decreased from 1.46902 to 0.88340, Saving the model ...
Epoch: 3        Training Loss: 1.178343         Validation Loss: 0.801886
Validation loss has decreased from 0.88340 to 0.80189, Saving the model ...
Epoch: 4        Training Loss: 0.946275         Validation Loss: 0.792311
Validation loss has decreased from 0.80189 to 0.79231, Saving the model ...
Epoch: 5        Training Loss: 0.840945         Validation Loss: 0.804919
Epoch: 6        Training Loss: 0.803315         Validation Loss: 0.921981
Epoch: 7        Training Loss: 0.715998         Validation Loss: 0.744238
Validation loss has decreased from 0.79231 to 0.74424, Saving the model ...
Epoch: 8        Training Loss: 0.692782         Validation Loss: 0.779112
Epoch: 9        Training Loss: 0.650972         Validation Loss: 0.804964
Epoch: 10       Training Loss: 0.600247         Validation Loss: 0.760841
Epoch: 11       Training Loss: 0.561570         Validation Loss: 0.807503
Epoch: 12       Training Loss: 0.554902         Validation Loss: 0.793924
Epoch: 13       Training Loss: 0.594363         Validation Loss: 0.839802
Epoch: 14       Training Loss: 0.570864         Validation Loss: 0.963448
Epoch: 15       Training Loss: 0.623032         Validation Loss: 0.886923
Epoch: 16       Training Loss: 0.518281         Validation Loss: 0.934599
Epoch: 17       Training Loss: 0.546448         Validation Loss: 0.841294
Epoch: 18       Training Loss: 0.553115         Validation Loss: 0.982515
Epoch: 19       Training Loss: 0.497795         Validation Loss: 0.829413
Epoch: 20       Training Loss: 0.524289         Validation Loss: 0.941311
```

Figure 9 shows training and validation loss for different epochs. As seen in this figure, after around 9 epochs, validation loss does not improve anymore.

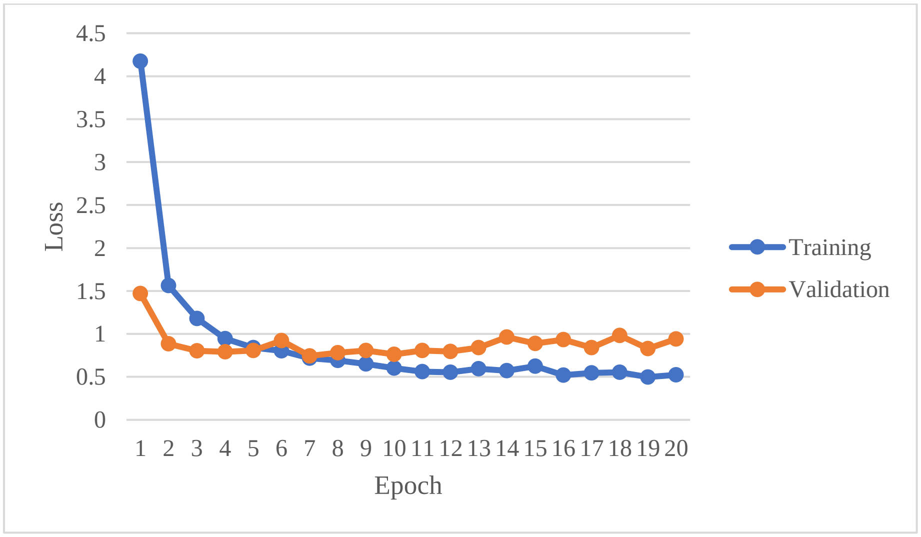


Figure 9 Training and validation loss over epochs

4. Results

Table 1 shows the classification accuracy on the test data set for convolutional networks with and without transfer learning.

Table 1 Classification accuracy on the test data set

Model	Classification Accuracy
CNN learning from scratch (benchmark)	13%
CNN with transfer learning	**77%**

As seen in this table, the network with transfer learning performs significantly better than the network without transfer learning. We need to mention that both networks were trained for 20 epochs on the same device and the training times are roughly the same for both methods. This highlights the benefits of using transfer learning.

5. Conclusions

In this project, we develop a program to take an image as input and determine if there is a dog or a human in the image. In the first case, the program predicts the dog breed. In the second case, the program determines which dog breed the human is most similar to. We used pretrained models to determine if there is a dog or human in the image. To determine the dog breed, we used two approaches. In the first approach, we used a convolutional neural network with four convolutional layers. We trained this network from scratch with no transfer learning. In the second approach, we used the transfer learning technique to get a better performance. Our results show that the second approach performs significantly better than learning from scratch. This highlights the usefulness of using transfer learning whenever it is possible.

The current project leads to 77% percent accuracy on the test data set for the task of identifying dog breeds. We believe this can be improved by employing techniques to prevent overfitting of the model. These include increasing the dropout probability in the dropout layers, and using a simpler model. Other potential improvement to this project is addressing the case when there is more than one entity in the image. For example, when there are two dogs or a dog and a human. Showing the second most probable breed is also useful in some situations.

References

[1] Iwata, A., H. Kawajiri, and N. Suzumura. "Classification of hand-written digits by a large scale neural network'CombNET-II'." [Proceedings] 1991 IEEE International Joint Conference on Neural Networks. IEEE, 1991.

[2] Lawrence, Steve, et al. "Face recognition: A convolutional neural-network approach." IEEE transactions on neural networks 8.1 (1997): 98-113.

[3] Li, Qing, et al. "Medical image classification with convolutional neural network." 2014 13th International Conference on Control Automation Robotics & Vision (ICARCV). IEEE, 2014.

[4] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.