Capstone Project

Machine Learning Engineer Nanodegree

Report

Osama Maharmeh

Feb 11, 20207

Definition

Project Overview

Fun and enjoyable applications are great since they make our lives better and more pleasant. And since everyone loves dogs, I have decided to build an application to combine these two ideas: enjoyability and human best friend(dogs).

In this project, I have created a web application that is capable of determining the breed of a dog in a given image. The application uses a classifier(CNN) trained on the dog dataset.

Problem Statement

The main objective of this project will be to use Deep Learning techniques (i.e. Convolutional Neural Networks) to classify dog breeds.

When given an image, first we want to determine if the image contains a dog, if so, we need to return the type of this dog breed. But if the image has human in it, we need to return the resembling dog breed for this human. If neither dog nor human was detected, we need to return a message telling so.

So, the application should accept an image as input, processes it and then return the desired output text.

Metrics

Accuracy is a common metric for classification; it takes into account both true positives and true negatives with equal weight.

$$accuracy = \frac{true\ positives + true\ negative}{positives + negatives}$$

And since the train data classes are not highly unbalanced, we can use accuracy as a metric.

Processing delay can also be considered as a metric since it has an impact on the user experience.

$$processing\ delay \simeq image\ preprocessing + classification\ delay$$

Image preprocessing delay is the time it takes to make the image ready for the model to classify it (i.e. Apply cropping and normalization).

Classification delay is the time it takes the model to make a prediction.

Analysis

Data Exploration

The dataset contains 8351 RGB colored dog images of 133 types (classes, breeds) with **different sizes.**

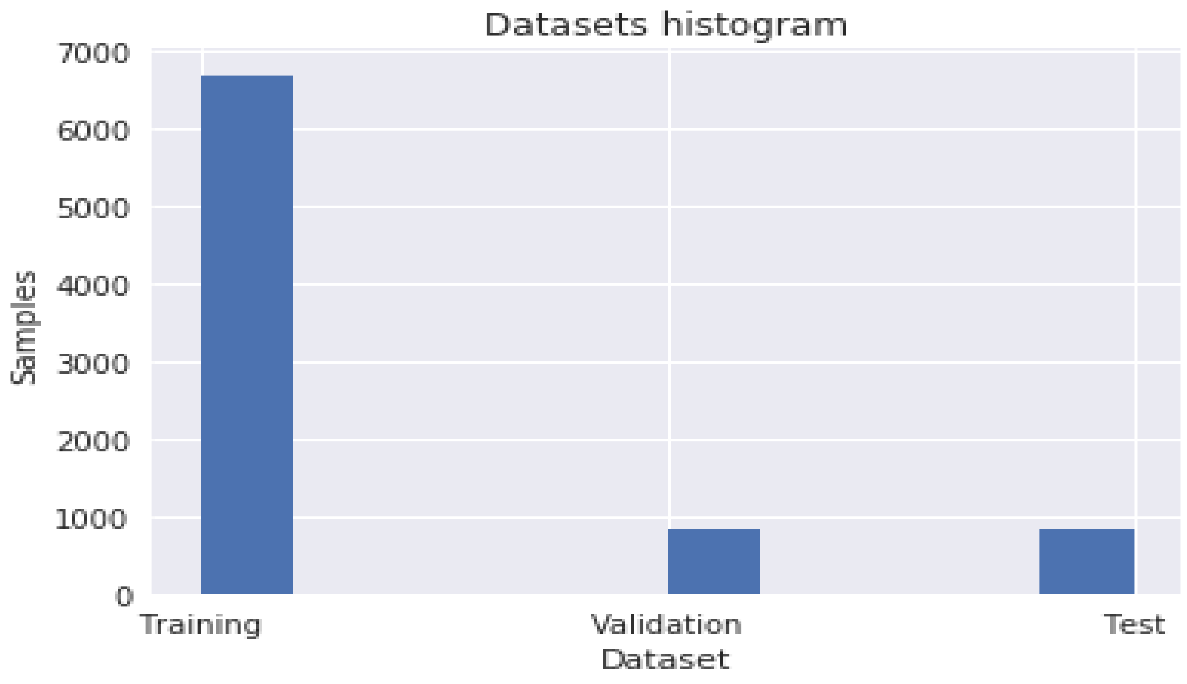
These images are already split into training, validation, and test subsets, where the train and validation folders contain 133 folders each one represents a single dog breed.

And for the test folder, it contains 133 folders too but each one with 5 to 10 images for testing purposes.

Exploratory Visualization

The dataset is fairly simple, there isn't much exploration to do. But let's underly some basic facts about the dataset.

The following figure shows how the data is split.



Dataset splits Figure

Since accuracy is my measurement metric, I needed to ensure that the data classes are somehow balanced. Classes frequency Figure (for better view check this link) verifies that the dataset is fairly balanced with minor classes having a bit fewer images than other classes.

Algorithms and Techniques

I have used Convolutional Neural Network as my classifier which is the most powerful algorithm for most image processing tasks, including classification. It needs a huge amount of data compared to other machine learning approaches and techniques.

And for human detection, I have used HaarCascades (frontal) in order to detect human faces. This feature detection algorithm has been proving its great capability of detecting objects (faces in our case).

Our dataset is not considered to be huge since we only have about 63 images per class.

The algorithm (CNN) outputs an assigned probability/score for each class, so we can examine how much is our model confident with a particular decision it has made.

Here I have a list of some parameters that can be tuned to optimize the network:

- The number of epochs.
- Batch size (Number of images to process in a single training step).
- Optimizer (SGD, Adam, Adamax, see optimizers).
- Activation function (ReLU, ELU, Swish, see activation).
- Weight initializer (glorot normal, glorot uniform, He normal, see initializers) with/without momentum.
- Learning rate.
- Network architecture parameters:
    - Number of layers.
    - Number of neurons in each layer.
    - Different layer parameters (i.e. Kernel size, use bias, use pooling, stride size, ...).
    - Dropout probability.

If GPU is available, then load the model to the GPU so it trains faster.

During the training phase, both the training and validation datasets are loaded to the GPU Memory in the form of batches which will increase the speed of retrieving these data samples. Mini-batch gradient descent is used to update the model's weights (the optimizer).

Benchmark

For the benchmark model, I have used two different models. The first one is the VGG16 model which can detect dogs, and the other model is the ResNet101 model which is trained on the ImageNet dataset. So here we have used transfer learning and I have made some changes to the ResNet101 model so it suits the dataset we have.

Both of these two models have shown that they are very powerful and capable of doing a great job in classification tasks.

Choosing such a powerful benchmark model will lead to use it instead of the built model simply because it's the result of huge research teams in different big companies and it took them a long time to accomplish such great models.

The Benchmark model (ResNet101) got 81.1% accuracy on the test dataset which is a great result since it was trained for 60 epochs only and the dataset is small(63 images per class).

Methodology

Data Preprocessing

The data preprocessing is done using PyTorch's data Loaders via applying the following transformations:

All datasets:

- Resize to 244 by 244.
- Normalization.

Training data:

- Random crop size 224 by 224.
- Random rotation by a range of (-30, 30) degrees.
- Random translation by a fraction of 0.2.
- Random scaling by a range of (0.5, 1.5) times.
- Random shear by 0.2 px.

All these random transformations will add variety to the dataset, reduce overfitting, and make the model regularized. It's worth to mention that these random transformations are applied to the training data only and the other two datasets only need to be resized and normalized.

When getting an image from a user, the image needs only to be resized and normalized then passed to the model and get predictions. Figure 1 shows some transformed images.

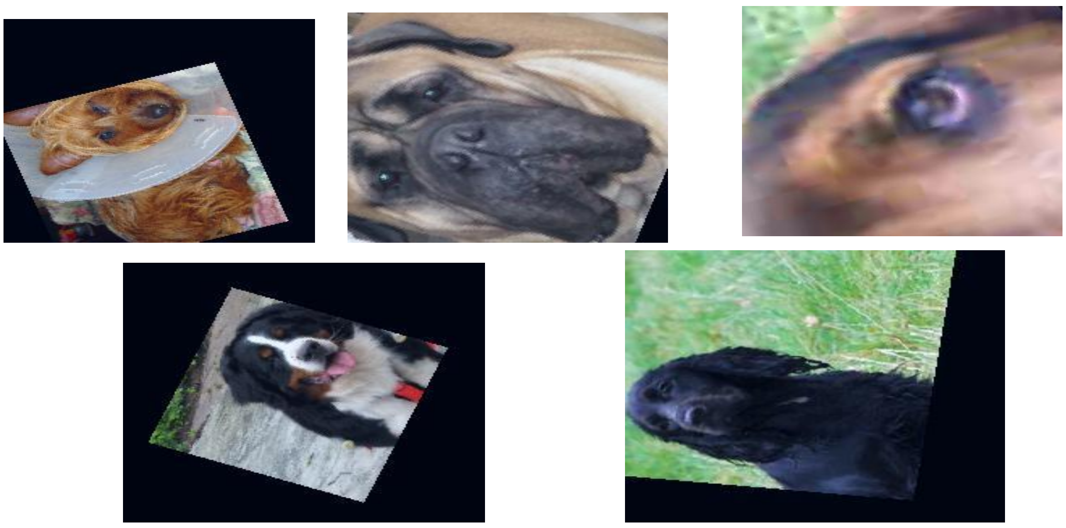


Figure 1: Sample Images (transformed) from the dataset.

Implementation

The implementation process can be divided into two main phases:

1. The CNN training phase.
2. The web application development phase.

In the first phase, the model was trained on the preprocessed training data, and this phase can be divided into the following steps:

1. Use PyTorch's data Loaders to load the datasets and apply the previously mentioned transformations.
2. Define the CNN architecture and instantiate the model.
3. Select CrossEntropy loss function.
4. Select Stochastic Gradient Descent (SGD) optimizer.
5. Specify a batch size and learning rate.
6. Train the network and log the validation/training loss.

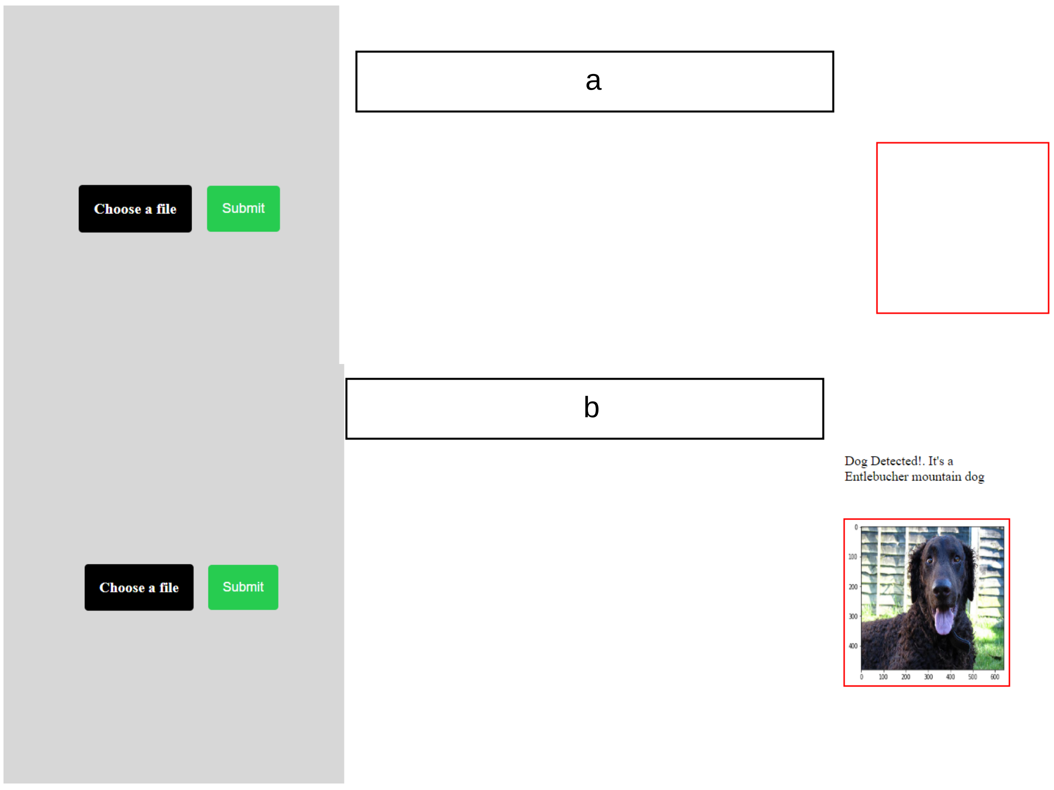
7. While the accuracy is less than indicated in the instructions, return to step 2 with different choices.
8. Save the model.

All these steps are implemented in the **dog_app.ipynb** notebook.

Also, there are some explanations on the model architecture answered in the notebook.

For the second phase of implementation, building the web application is pretty basic and simple.

It consists of one page, where the user can upload his image and submit it. The image will be supplied to the model in order to get the prediction, then the image will be displayed with the prediction message that describes what inside this image (dog, human, neither). The following figure shows a sample test of the web application's page.



Dog Detected!. It's a Entlebucher mountain dog

Figure 2: a: upload and submit the image page. b: results page.

The web application is built using the Flask framework, and the page was designed using pure HTML and CSS.

You can find the web application implementation code inside the **WebAPP** folder.

Refinement

The initial model had the following properties:

1. Convolution, fully connected, activation layers only.
2. Used 0.001 as a learning rate.
3. Batch size of 32 images per batch.

After training this model for 50 epochs I got 6% accuracy, and both training and validation loss were high. This is an indication of underfitting, so I knew that I needed to add complexity to the model.

I also noticed that the training loss was decaying slowly and I need to increase it.

In order to overcome these problems, I have modified the model as follows:

1. Increased the number of Neurons in each layer, and also introduced Dropout layers in order to regularize the model.
2. Increased the learning rate so the training becomes faster (made it 0.006 -> six times the previous one).
3. Used a higher batch size of 64, in order to improve the Stochastic Gradient Descent (SGD) optimizer.

After these modifications, I have trained the model for another 25 epochs until I got 20% accuracy which is higher than the required 10%.

After doing the Transfer Learning section and trained the ResNet101 model and got 81% accuracy out of it, I decided to use it as my classifier for the web application. Figure 3 presents and compares training and validation loss.

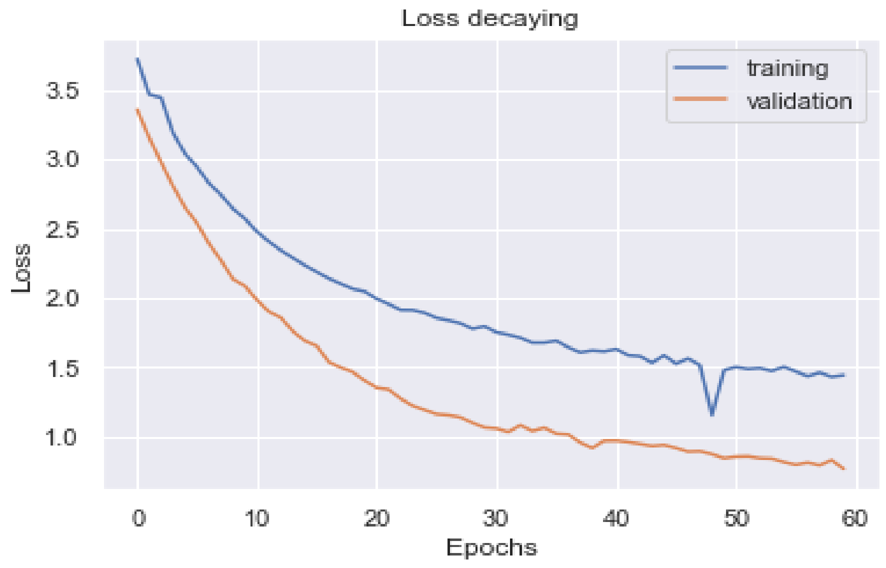


Figure 3: training and validation loss decaying over epochs

After the training phase finished the model had 81% accuracy which is higher than the required 60%.

Results

Model Evaluation and Validation

During development and training, a validation dataset was used to evaluate the model. The final architecture and hyperparameters were chosen because they performed the best among the tried combinations.

For a complete description of the final model and the training process, refer to Figure 4 along with the following list:

- Kernel size of the convolution layers is 3 by 3, and a stride of 2(which helps with dimensionality reduction throughout the forward pass) with 1px padding.

- The first feature extractor (convolution layer) uses 32 learners, the second uses 64, and the final one uses 128.
- Pooling (2D Max Pooling) layers with strides of 2 and pooling of 2 in order to reduce the size of the input.
- Add regularization by using Dropout layers.
- 25 epochs.
- 64 images per batch.
- 0.006 learning rate.

During the training, I have used the provided required accuracy to bound my training time and epochs.

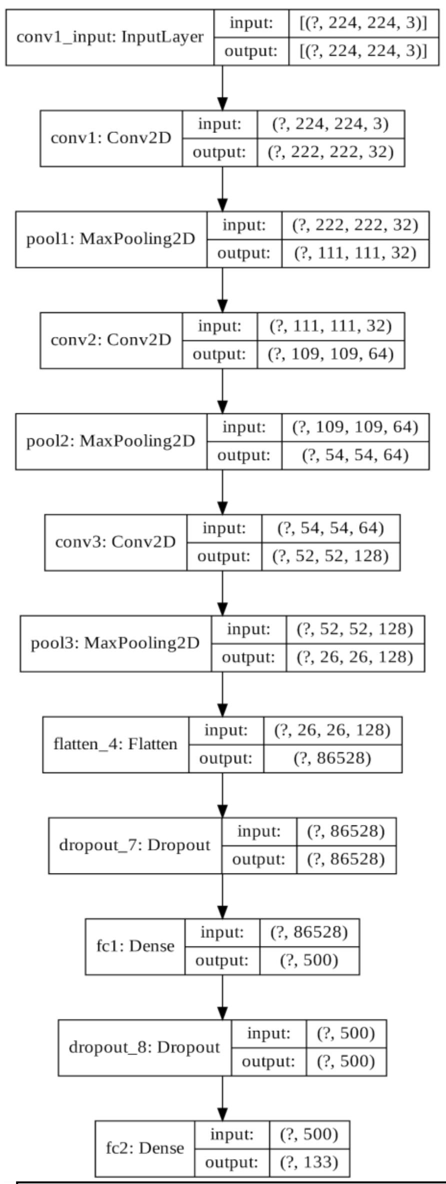


Figure 4: The model architecture

Justification

The used benchmark model clearly provided better results than the one I have built; it spends more time making a prediction since it's much more complex which will affect the user experience. The time difference is not significant though (1.4 seconds on average).

To measure the model's experience, I let some of my friends use the benchmark model and others use the one I built, I have noticed that the enjoyed the one I built because "the mistakes it did were somehow funny and enjoyable".

In summary, I have used the benchmark model because I liked its results the most, I think that the applications' benefits are to increase people's familiarity with dog breeds on one side and add joy and humour on the other side.