Capstone Project Hemang Goyal

Machine Learning Engineer Nanodegree 30/5/2020

Dog Breed Classifier using CNN

Project Overview

The Dog breed classifier is a well-known problem in Machine Learning domain. Given an image of a dog, the classifier has to identify an estimate of the canine's breed. If supplied an image of a human, the classifier will identify the resembling dog breed. The goal is to build a pipeline which can process real world user supplied images and identify an estimate of the canine's breed. We can use supervised machine learning algorithms to solve this problem as this is a multi-class classification problem.

Problem Statement

The goal of this project is to build a machine learning model which will act as a classifier to classify images which can be used within web app to process real-world, user-supplied images.

The algorithm has to perform two tasks:

Dog face Detection: Given an image of a dog, the algorithm will have to detect that it is a dog's face and predict an estimate of the dog's breed.

Human face Detection: If supplied an image of a human, the algorithm will identify that it is a human face and then will try to find the dog breed from the model which will resemble the most or closely to that human face.

Metrics

The data is split into train, test and valid dataset and the model is trained using the train dataset. We use the testing data to predict the performance of the model on unseen data. We will use accuracy as a metric to evaluate our model on test data.

Accuracy=Number of items correctly classified/ All classified items

We are using Accuracy as the metric here because the metric given in the benchmark is accuracy :

- The CNN model created from scratch must have accuracy of at least 10%.
- The CNN model created using transfer learning must have accuracy of 60% and above.

So it will easy to compare the performance of the model with the benchmark when using accuracy as the evaluation metric.

We can also think of using other evaluation metrics like F1-score or ROC AUC but that is more effective to use in binary classification than multi-class classification. Also F1-score is used when data is very much imbalanced but here the data is not that much imbalanced and so I decided to use accuracy as the evaluation metric.

During model training, we compare the test data prediction with validation dataset and calculate Multi class log loss to find the best performing model. Log loss calculates the amount of uncertainty of prediction based on how much it varies from actual label and this will help in evaluating the model.

Data Exploration :

For this project, the input format must be of an image type, because we want to input an image and identify the breed of the dog. The dataset for this project is provided by Udacity itself. It contains pictures of dogs and humans.

Here are the details about the dataset :

Dog images dataset*:*

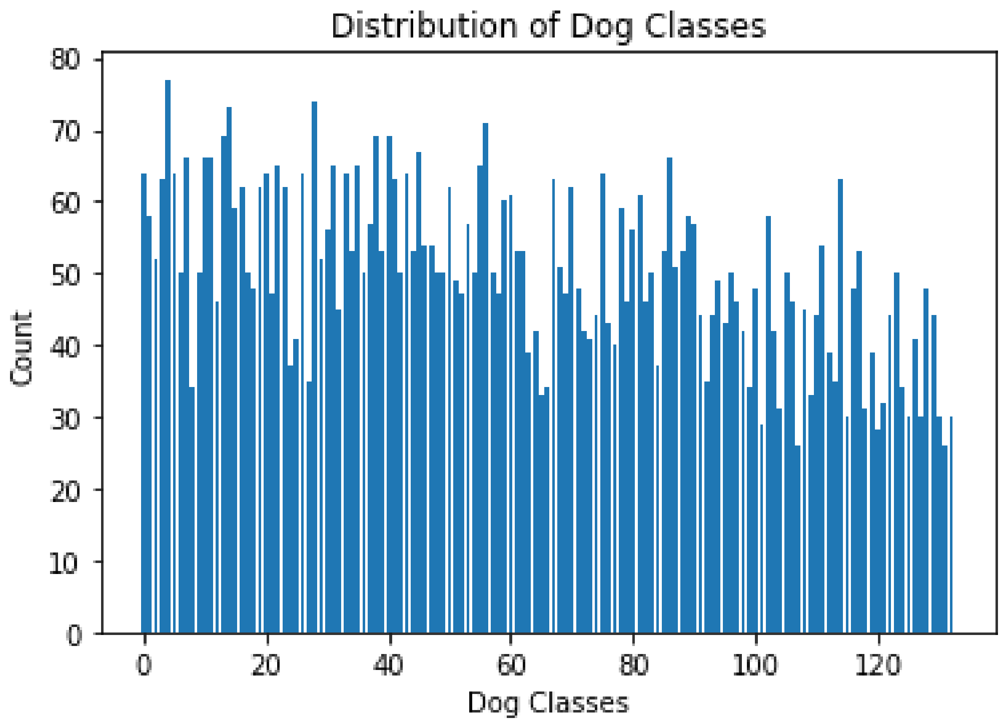
Total images : 8351

Train images : 6680

Valid images : 835

Test images : 836

Each of this directory (train, test, valid) have 133 folders corresponding to dog breeds. The images are of different sizes and different backgrounds, some images are not full-sized. Here is the distribution of various dog classes in the training dataset :



As we can see, there is not much imbalance in the dataset. Here are some statistics about this dataset :

Minimum count value for a class : 25

Maximum count value for a class : 77

Median value of the above distribution : 50

Mean value of the above distribution : 50.22

Standard deviation of the above distribution : 11.81

The above figures suggest that there is not huge imbalance in the training dataset and so is that in validation and test dataset and since the dataset is small in size, removing some imbalanced data classes will make our model inefficient and hence we will use methods like image augmentation, image normalization, image rotation, etc. to balance the datasets in our model.

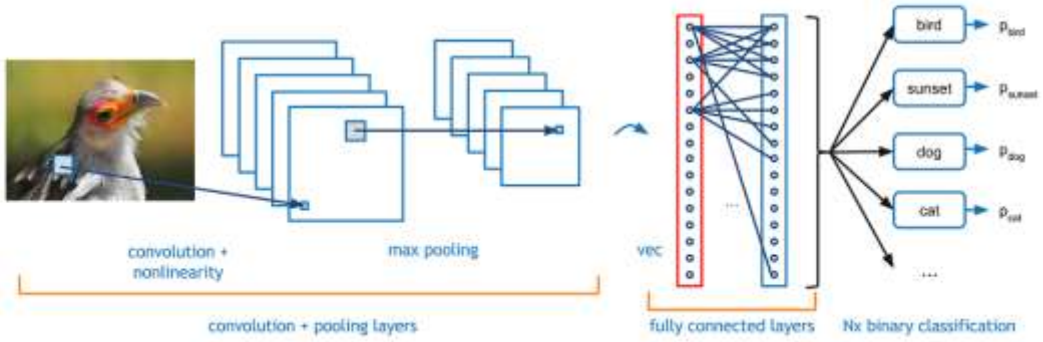
Human images dataset :

The human dataset contains 13233 total human images which are sorted by names of human (5750 folders). All images are of size 250x250. Images have different background and different angles.

Sample images from the dataset

Algorithms and Techniques

The main algorithm we are going use to solve this multi-class classification problem is Convolutional Neural Network. A **Convolutional Neural Network (CNN)** is a Deep Learning algorithm which can take in an input image, can learn various features in the image (using learnable weights and biases) and using this features, it can be able to differentiate between different objects/images and can help classify them.



A CNN has

- **Convolutional layers** : Convolutional layers apply a convolution operation to the input. This passes the information on to the next layer.
- **Activation layers** : It applies an activation function like ReLU (Rectified Linear Unit) function to the output of Convolutional layers to increase non-linearity
- **Pooling layers** : It combines the outputs of clusters of neurons into a single neuron in the next layer.
- **Fully connected layers** : They connect every neuron in one layer to every neuron in the next layer.

A CNN works by extracting features from images. This eliminates the need for manual feature extraction. The features are not trained beforehand but they are learned while the network trains on a set of images. This makes deep learning models extremely accurate for computer vision tasks. CNNs learn feature detection through tens or hundreds of hidden layers. Each layer increases the complexity of the learned features.

This is how the CNN works on an image :

- starts with an input image
- applies many different filters to it to create a feature map
- applies a ReLU or any other activation function to increase non-linearity
- applies a pooling layer to each feature map
- flattens the pooled images into one long vector.

- inputs the vector into a fully connected artificial neural network.
- processes the features through the network. The final fully connected layer provides the “voting” of the classes that we’re after.
- trains through forward propagation and backpropagation for many, many epochs. This repeats until we have a well-defined neural network with trained weights and feature detectors.

There are various hyper-parameters used in a CNN model and one can do hyper-parameter tuning in the model in various ways like :

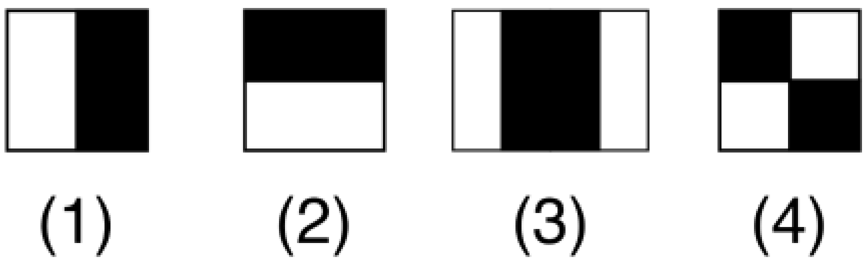
- Adding and removing dropouts in convolutional layers
- Batch Normalization (BN)
- L2 regularisation
- Increasing the number of convolution layers
- Increasing the number of filters in certain layers

In our solution, we are doing the following:

1) Detect Human Images using face detector:

To do this, we can use existing algorithm like OpenCV’s implementation of Haar feature based cascade classifiers.

A Haar-Feature is just like a kernel/filter in CNN, except that in a CNN, the values of the kernel are determined by training, while a Haar-Feature is manually determined. Haar-Features are good at detecting edges and lines. This makes it effective in face detection.



Common Haar Features used to detect edges and lines in images

The OpenCV's Haar feature based cascade classifier detects human faces using this haar features but they are effective only in detecting human faces in the image.

2) **Detect Dog Images** :
We have done dog image detection using a pretrained VGG16 model. we tried to detect

3) **Use CNN to classify Dog Breed :**
After the image is identified as dog/human, we can pass this image to an CNN which will process the image and predict the breed that matches the best out of 133 breeds.

For this we have used 2 models :

- CNN model created from scratch : A CNN was created from scratch to implement the classifier to classify the dog breed. Details about the model can be found in the implementation part. The accuracy of the model was more than the benchmark accuracy required for CNN model from scratch but in general it was quite less and so I decided to run a model using transfer learning

- CNN model using Transfer learning : I have implemented a CNN model through transfer learning using ResNet101 model. I have selected the Resnet101 architecture because it is pre-trained on ImageNet dataset using million images and it can classify images into 1000 different object categories. The architecture is 101 layers deep. The fully connected layer at the end of the model was modified to make the output layer classify image into 133 classes.

Benchmark

- ✓ The CNN model created from scratch must have accuracy of at least 10%. This can confirm that the model is working because a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%.
- ✓ The CNN model created using transfer learning must have accuracy of 60% and above.

Data Preprocessing

All the images are resized to 224*224, then normalization is applied to all images (train, valid and test datasets). For the training data, Image augmentation is done to reduce overfitting and imbalance in the dataset. The train data images are randomly rotated and random horizontal flip is applied. Finally, all the images are converted into tensor before passing into the model.

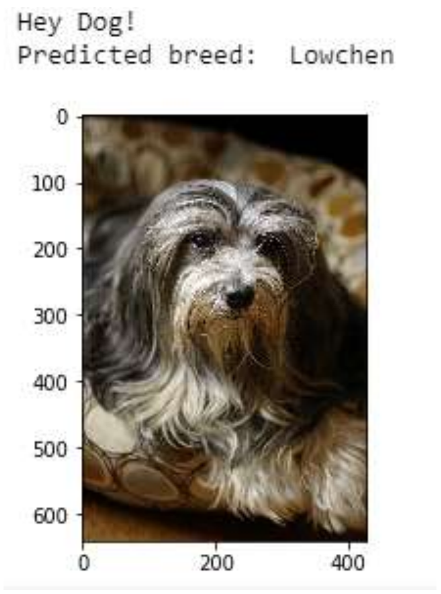
Implementation

To solve this problem, I built a CNN model from scratch. The model has 3 convolutional layers. All convolutional layers have kernel size of 3 and stride 1. The first conv layer (conv1) takes the 224*224 input image and the final conv layer (conv3) produces an output size of 128. ReLU activation function is used here. The pooling layer of (2,2) is used which will reduce the input size by 2. We have two fully connected layers that finally produces 133-dimensional output. A dropout of 0.25 is added to avoid over overfitting.

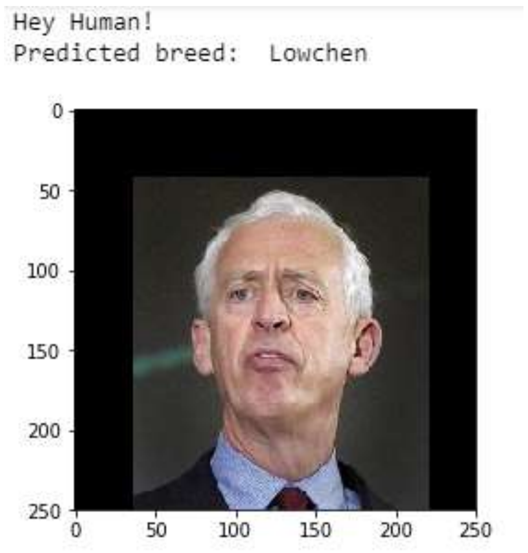
Refinement

The CNN created from scratch have accuracy of 23%, Though it meets the benchmarking, the model can be significantly improved by using transfer learning. To create CNN with transfer learning, I have selected the Resnet101 architecture which is pre-trained on ImageNet dataset, the architecture is 101 layers deep. The last convolutional output of Resnet101 is fed as input to our model.

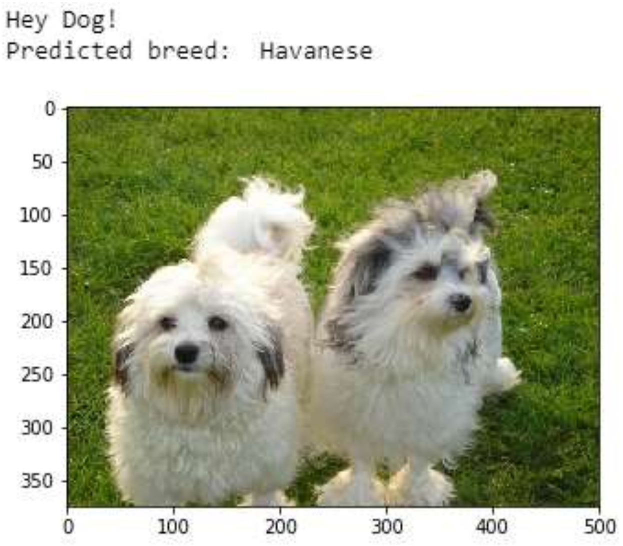
We only need to add a fully connected layer to produce 133-dimensional output (one for each dog category). The model performed extremely well when compared to CNN from scratch. With just 5 epochs, the model got 80% accuracy.



```
Hey Dog!
Predicted breed:  Lowchen
```



```
Hey Human!
Predicted breed:  Lowchen
```



```
Hey Dog!
Predicted breed:  Havanese
```

Sample output predicted from the model

Model Evaluation and Validation

Human Face detector: The function used in this was created using OpenCV's implementation of Haar feature based cascade classifiers. 98% of human faces were detected in first 100 images of human face dataset and 17% of human faces detected in first 100 images of dog dataset.

Dog Face detector: The dog detector function was created using pre-trained VGG16 model. 100% of dog faces were detected in first 100 images of dog dataset and 0% of dog faces detected in first 100 images of human dataset.

CNN model created from scratch : The CNN model created from scratch was trained for 25 epochs and the model produces an accuracy of 23% on test data. The model correctly predicted breeds for 195 images out of 836 total images.

CNN using transfer learning: The CNN model created using transfer learning with ResNet101 architecture was trained for 5 epochs, and the final model produces an accuracy of 80% on test data. The model correctly predicted breeds for 675 images out of 836 total images.

Accuracy on test data: 80% (675/836)

Justification

Overall the CNN model created using transfer learning from ResNet101 model performed better than expected. The accuracy of the model was 80% which is better compared to the accuracy of 23% from the CNN model created from scratch.

Improvements

The model can be improved by adding more training and test data. Right now, the model is created using only 133 breeds of dog. Also, by performing more image augmentation, we can avoid overfitting and improve the accuracy.

The accuracy of the CNN model built from scratch can be improved more as it is only 23% right now. One can try to add more convolutional layers and make a deep architecture for better feature detection and accuracy. Also for feature extraction, I have tried only with ResNet 101 architecture, so may be the model can be improved using different architecture.

References

1. Original repo for Project - GitHub*:*

   https://github.com/udacity/deep-learning-v2- pytorch/blob/master/project-dog-classification/

2. About Resnet101 architecture*:*

   https://pytorch.org/docs/stable/_modules/torchvision/models/resnet.html#resnet101

3. Imagenet training in Pytorch*:*

   https://github.com/pytorch/examples/blob/97304e232807082c2e7b54c597615dc0ad8f

   6173/imagenet/main.py#L197-L198

4. CNN:

   https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural- networks-the-eli5-way-3bd2b1164a53

5. About Log loss function *: http://wiki.fast.ai/index.php/Log_Loss*