CAPSTONE PROJECT

Ameya Shete

Machine Learning Nanodegree

26th June 2021

DEFINITION

Project Overview

The Dog Breed Classifier using Convolutional Neural Networks is a Udacity suggested project in which we were tasked with first classifying if the species in an image was a dog or human and then subsequently classifying the breed of the dog or which breed the human most resembled.

The main challenge of this project was that most dogs have basic features such as four legs, two eyes, tail etc. so we as data scientists needed to come up with an extremely granular method to differentiate between breeds of dogs. Therefore, the main differences we had to use in differentiating between different breeds of dogs was size, color shape.

This project also has many real-world applications. Not only could a project of this nature be applied to classify dogs, we can only apply this to many other species in which there are different breeds such as cats, horses, birds etc. The only requirement for such a project would be that there should be many breeds for each animal and that there are enough images of each breed in the training set so that the model can effectively classify the breed of the animal. Furthermore, this ML classifier would be helpful in the zoology field as it would help in finding out how many animals of a certain breed currently live in the wildlife based purely from satellite images. There is no doubt that such a classifier would help in saving time and resources in zoology and biodiversity fields.

Problem Statement

The problem statement is that given an image of a dog, the algorithm we develop should correctly classify the breed of the dog and thus comes under the realm of fine grained classification. The main challenges of his problem are that at a high-level, most dogs look the same except for their size, shape and color as there are many different dog breeds and there also may be a lot of noise in the images of humans, trees or objects in general in the background.

Convolutional neural networks is the method implemented in this project which develops features to use for classification from the pixels in the image. Furthermore, I also implement a transfer learning approach to converge at a higher performance level, enabling more accurate output since the starting point of a transfer learning approach is better and the learning rate is also faster. An advantage of the transfer learning approach is that the same model can be used for different datasets so this algorithm used to classify dogs can also be used to classify cats or horses. This is due to the fact that the desired performance of the model is reached faster as we use a pre-trained model.

Metrics

Accuracy is a common binary classifier as it takes into account both the true positives and false negatives in equal weight:

$$accuracy = \frac{true\ positives + true\ negatives}{dataset\ size}$$

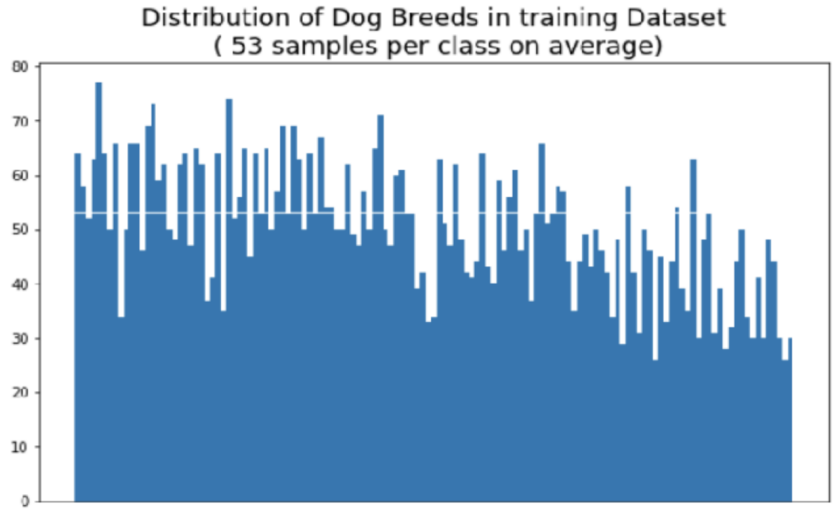
The only evaluation metric that we use is accuracy score. The accuracy score is determined by looking at the predicted label for each image and comparing it with the true label. Finally, the accuracy score will be a measure of how many of the images were correctly labeled as a percent of the total number of images. Accuracy is an easy to understand metric and is quite applicable as well given the simple nature of what we are trying to achieve in this project.

ANALYSIS

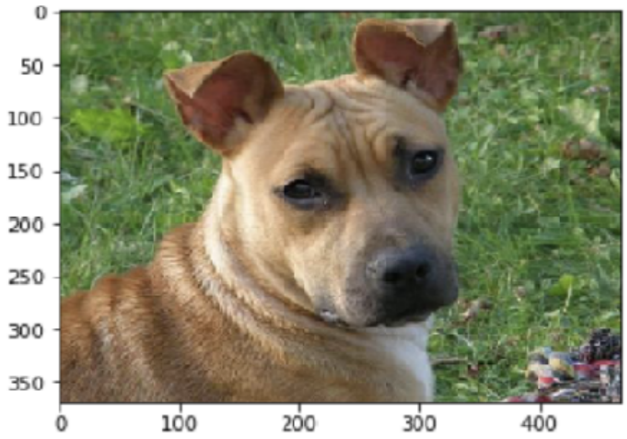
Data Exploration

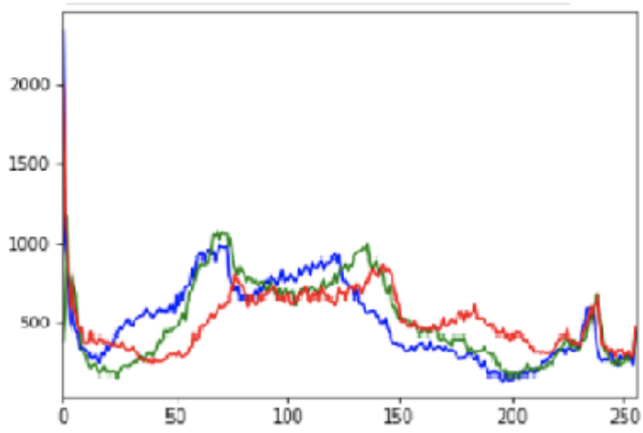
The datasets I have used in this project are from the Jupyter notebook that Udacity provided, dog_app.ipynb. The datasets have 8351 dog images and 13233 human images. The number of images in the dataset is also interesting because there seems to be a bit of class imbalance within the dog breeds. Class imbalance negatively affects the sampling of the data and also will affect the metric as there are better metrics that can be employed when we have an imbalance class. In this case, we find that there are, on average, 53 samples per class with slight variations within the classes.

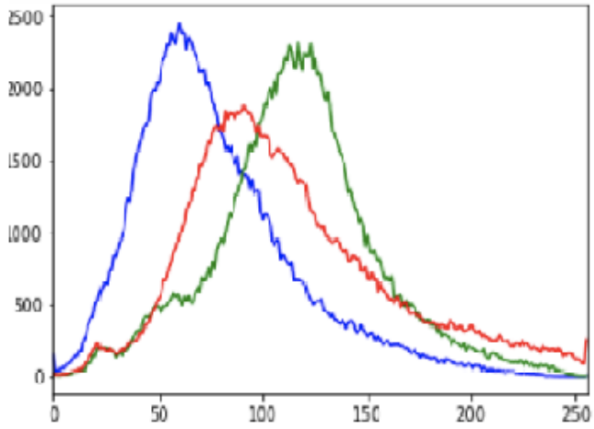
Data Visualization



We can see from this graph that there are about 53 images per breed and the variation in each sample highlights the fact that there is class imbalance in this dataset.







I also compared the pixel intensity distribution for the images in the American Staffordshire Terrier and found that they are varying in contrast, brightness and size. Furthermore, looking at the RGB graphs for the two photos shows that distribution for the same is quite varied making the task of classification of dog breeds all the more challenging.

Algorithms and Techniques

The classifier is a Convolutional Neural Network which is a state-of-the-art algorithm for most image processing tasks such as classification. For CNN to work most effectively, we need to use a large dataset which we do have with the human and dog datasets. The main hyperparameter that I used to train the model was the training length, i.e- number of epochs, which I used as 50.

For the initial CNN from scratch model, I created a loaders_scratch variable in which I stored the training, test and validation data. Then, I also used the variable criterion_scratch to define the criteria, Cross Entropy Loss which would be the loss function when training the model. Cross entropy Loss is an effective method since we are trying to solve a multi-classification problem and this loss function is one of the most common ones used. Cross Entropy loss allows us to iteratively adjust the model weights to reduce loss to an optimal amount and deduce the probability of how close the prediction is to the true label.

Finally, I also used the variable optimizer_scratch to define the optimization model used in the algorithm. In this algorithm, I used the Stochastic Gradient Descent. We use SGD in this case since it is a commonly used optimizing function to find the optimum point of a loss function, the cross entropy loss. Finally, these variables were all stored in the model_scratch variable.

In [17]:
```
# ---Defining Param-----
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

param_epochs = 50

# train the model
model_scratch = train(param_epochs, loaders_scratch, model_scratch, optimizer_scratch,
                      criterion_scratch, use_cuda, 'model_scratch.pt')
```

Benchmark Model

Typically in the past, image classification problems have relied on extracting features by hand such as edges and texture descriptors combined with ML methods such as PCA or linear discriminant analysis. This has now been replaced by CNNs and is far more accurate due to the fact that the model now learns from actual differences in the images.

Therefore, one benchmark model that we can use would be a CNN model developed from scratch in which the model guesses randomly which breed the dog belongs to and so will have a test accuracy of about 10%. We will use transfer learning later in the model we actually want to implement and thus can compare the accuracy of this new model with the benchmark model of a CNN model built from scratch.

METHODOLOGY

Pre-processing data

I loaded the training, testing and validation datasets and then created data loaders for the same. After this, I resized all images to center cropped, 224 pixels. I also augmented the dataset by randomly adding rotation, horizontal and vertical flips also helps with overfitting the data.

Furthermore, most of the pre-trained models require the image to be in 224x224 format so I resized them accordingly. Furthermore, I also needed to have the exact normalization as when the models were trained and so that the red, blue green channels must be normalized separately. The means are [0.485, 0.456, 0.406] and the standard deviations are [0.229, 0.224, 0.225].

Implementation

The implementation stage can be split up into two different stages, one stage was for the CNN model built from scratch and the other stage can be defined as the CNN using transfer learning. As the CNN model being built from scratch was simply a benchmark model with an accuracy of over 10% since the classification was random, I will be mainly discussing the CNN model using transfer learning.

Starting with the model architecture:

In [60]:
```
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

        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)

        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)

        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)

        self.pool = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(28 * 28 * 64, 500)
        self.fc2 = nn.Linear(500, len(train_data.classes))
        self.dropout = nn.Dropout(0.25)
        self.batch_norm = nn.BatchNorm1d(num_features=500)

    def forward(self, x):
        ## Define forward behavior
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))

        x = x.view(x.size(0), -1)
        x = self.dropout(x)
        x = F.relu(self.batch_norm( self.fc1(x)) )
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

This is the neural network we implemented previously for the CNN model built from scratch. As we can see, the conv1, conv2 and conv3 layers are the convolutional layer while fc1 and fc2 layers are fully connected layers. pool indicates that it is the max pooling hyper parameter. The forward function defines the forward behavior of the neural network. There are no cycles or loops in this neural network. The dropout layer is discussed in more detail in the 'Refinement' section of this report below.
We can see that we have implemented resnet50 as it is a very effective method. In resnet50, the previous layers are feeded into the next layers so forward layers learn from the previous ones. This allows for the features that are extracted to be more powerful and accuracy of the model is increased.

In [27]:
```
criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.SGD(model_transfer.fc.parameters(), lr=0.001)
```

In [25]:
```
import torchvision.models as models
import torch.nn as nn

## TODO: Specify model architecture
model_transfer = models.resnet50(pretrained=True)

for parameter in model_transfer.parameters():
    parameter.requires_grad = False

model_transfer.fc = nn.Linear(2048, 133, bias=True)

fc_parameters = model_transfer.fc.parameters()

for parameter in fc_parameters:
    parameter.requires_grad = True

if use_cuda:
    model_transfer = model_transfer.cuda()
print(model_transfer)
```

In this snippet of code, we have chosen the loss function criteria and the optimizer function which are cross entropy loss and stochastic gradient descent respectively. For SGD, the network calculates the derivative of the error function with respect to the network weights, and changes the weights such that the error decreases.

In [28]:
```
# train the model
model_transfer = train(param_epochs, loaders_transfer, model_transfer, optimizer_transfer,
                       criterion_transfer, use_cuda, 'model_transfer.pt')

# load the model that got the best validation accuracy (uncomment the line below)
model_transfer.load_state_dict(torch.load('model_transfer.pt'))
```

Here, we are training the model with the various variables we had defined earlier such as loaders transfers, model transfers, optimizer transfers and criterion transfers. We have also used a training length of 50 epochs.

In [29]:
```
test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 0.549265


Test Accuracy: 84% (707/836)
```

Here, we have tested the transfer learning CNN model we developed earlier and have found that the test accuracy to be 84%.

(IMPLEMENTATION) Predict Dog Breed with the Model

Write a function that takes an image path as input and returns the dog breed (`Affenpinscher` , `Afghan hound` , etc) that is predicted by your model.

In [55]:
```
### TODO: Write a function that takes a path to an image as input
### and returns the dog breed that is predicted by the model.

# list of class names by index, i.e. a name can be accessed like class_names[0]
class_names = [item[4:].replace("_", " ") for item in train_data.classes]

def predict_breed_transfer(img_path):
    # load the image and return the predicted breed
    image_tensor = image_convert_tensor(img_path)

    if use_cuda:
        image_tensor = image_tensor.cuda()

    y = model_transfer(image_tensor)

    _, preds_tensor = torch.max(y, 1)
    if not use_cuda:
        prediction = np.squeeze(preds_tensor.numpy())
    else:
        prediction = np.squeeze(preds_tensor.cpu().numpy())

    return class_names[prediction]

def print_image(img_path, title="Title"):
    image = Image.open(img_path)
    plt.title(title)
    plt.imshow(image)
    plt.show()
```

Finally, we write the algorithm which allows us to predict the breed of the dog and print the image of the dog after predicting the breed of the dog in the image. Here, we have used the model_transfer variable that we had defined earlier which contains the trained model.

In [17]:
```
# ---Defining Param-----
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True
```

One main complication faced while implementing the algorithm was that during the training process, the training process couldn't begin since we kept getting an error "image file truncated (150 bytes not processed)". Therefore, to solve this issue, we imported the load truncated images method from PIL which helped resolve this issue.

Refinement

There was a lot of refinement done with this model to find the most effective model. One of the areas we continually refined the model was when writing the Net() function itself as it was important to choose the number of input and output paths and the number of kernels. This process was done iteratively to choose the optimal values for each hyperparameter.

Furthermore, another aspect of the Net() function that was refined to be added was the dropout layer. The dropout layer was implemented to add a bit of randomness to the model since the dropout layer will randomly drop weights into the layer and causes the outputs to be scaled by a factor of p/(1-p) where in this case, we have used a p = 0.25. The dropout method has proven to be an effective technique for regularization and preventing the co-adaptation of neurons.

RESULTS

Model Evaluation and Validation

While developing the model, we used a validation set to test the model after training it.

The final architecture and hyperparameters were chosen since they were the most effective. The final parameters used were:

```
self.conv1 = nn.Conv2d(3, 16, 3, padding=1)

self.conv2 = nn.Conv2d(16, 32, 3, padding=1)

self.conv3 = nn.Conv2d(32, 64, 3, padding=1)

self.pool = nn.MaxPool2d(2, 2)

self.fc1 = nn.Linear(28 * 28 * 64, 500)
self.fc2 = nn.Linear(500, len(train_data.classes))
self.dropout = nn.Dropout(0.25)
self.batch_norm = nn.BatchNorm1d(num_features=500)
```

For the three different convolutional layers, we used a different number of input and output paths at (3, 16), (16, 32) and (32, 64) respectively. Initially, we kept the in and out channels the same but found that the model was most effective with an increase in the number of in and out channels. The out channels are the number of filters that the model will use so as we go deeper into the neural network, more filters will mean the feature extraction in the training process is more refined.

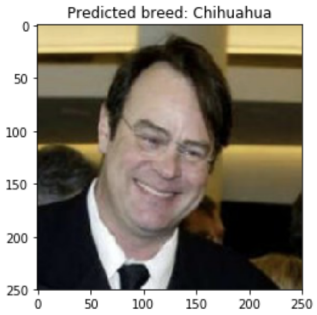
We have used max pooling with kernel size = 2 and this helps the model because max pooling is a sample-based discretization process. This helps in dimensionality reduction and allows for assumptions to be made about features. It also helps in reducing overfitting and reduces computational cost by decreasing the number of parameters to learn.

We have used the fully connected layers to be a linear model with (28 * 28 * 64, 500) parameters and these layers are simply used in the forward feeding process. We see that we have used 500 out channels and this is advantageous to the model since there is a lot of information being feeded forward.

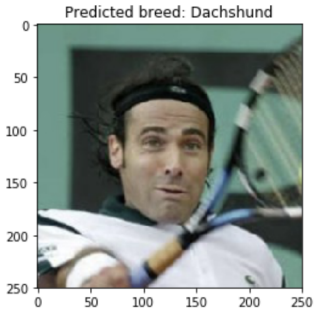
Finally, we also have the batchnorm 1D method which is an integral part of neural networks and in this case we have taken the number of features as 500. Batch norm is important to a neural network model since it takes the data feed from a previous layer and then normalizes it before sending it to the next layer.

To verify the robustness of the model, we tested the final model on some images of humans and dogs to see the prediction.

```
Hello, human!
```

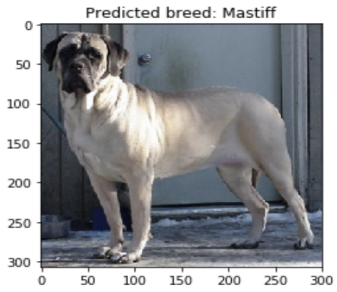



```
You most closely resemble-
Chihuahua
Hello, human!
```

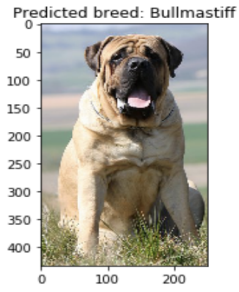



```
You most closely resemble-
Dachshund
```

```
Your breed is-
Bullmastiff
Hi, dog!
```




```
Your breed is-
Mastiff
Hi, dog!
```




```
Your breed is-
Bullmastiff
```

Justification

We can see from above that the model works quite well. The accuracy score is 84% compared to the CNN model built from scratch which had an accuracy score of 16%. It is able to detect between dogs and humans and what's more interesting is that it is even able to tell the difference between bullmastiff and mastiff! Seeing as these two breeds look very similar, the fact that the model is able to correctly tell the difference between such nuanced breeds is quite remarkable.

While 84% is an acceptable prediction level (above the 60% that was required), this model is certainly not complete. There certainly is room for improvement and there are strategies and methods to implement that will aid in doing so. These are discussed below.

Improvements

The model accuracy could certainly be improved by implementing augmentation and more training data as the dataset was a little imbalanced. If more data were to be added, the neural network would be able to understand the key differences between different breeds of dogs more thoroughly, however the downside would be that training time would increase a lot and so would the amount of memory being used.

Another approach to improve accuracy could be through ensemble techniques wherein multiple networks' predictions are averaged together to see if model accuracy is improved. Furthermore, another improvement could be to fine-tune hyperparameters by applying more hyper-parameter tuning strategies.

References

[1]https://towardsdatascience.com/deep-learning-build-a-dog-detector-and-breed-classifier-using-cnn-f6ea2e5d954a