MACHINE LEARNING ENGINEER NANODEGREE

CAPSTONE PROJECT: DOG BREED CLASSIFICATION USING CNN

PROJECT REPORT

REPORT BY:
ROHAN BHAGWATKAR

1. **Overview:**
   Multi-class classification using CNN (Convolutional Neural Network) is an interesting part of the ML domain. In this, we develop a model that is capable of predicting a class to which the input belongs. In our case, the input is an image. The problem at hand is to classify the input image into a dog breed class. If the input image is of a dog then the model should correctly classify it to a breed whereas if the input image is not of a dog, the model should predict the most resembling breed for the input image. We choose CNN as the way to go because this problem statement involves dealing with images, processing them, etc.

2. **Problem Statement:**
   We need to develop a model (deep neural network) to classify the input into different classes. The input may or may not be a dog image. Depending on the input image, the model should predict the class. If the input image is of a dog, the correct breed of that dog must be predicted. If the input image is not of a dog (let's say it is that of a human being) then the model must predict the most resembling breed that the input showcases. Once the model is developed, it

should be tested for a minimum of 6 images supplied by the user. (This is showcased is the last cell output in the dog_app.ipynb)

3. **Exploratory Data Analysis:**
   The complete dataset is provided by Udacity. It can be downloaded from here: Original Udacity Project Link: https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification - Point (2) & (3) in the instructions section of README.
   Total Dog Images: 8351
   Total Human Images: 13233

TYPE	DOG (Images)
Train	6680
Test	836
Valid	835
Total	8351

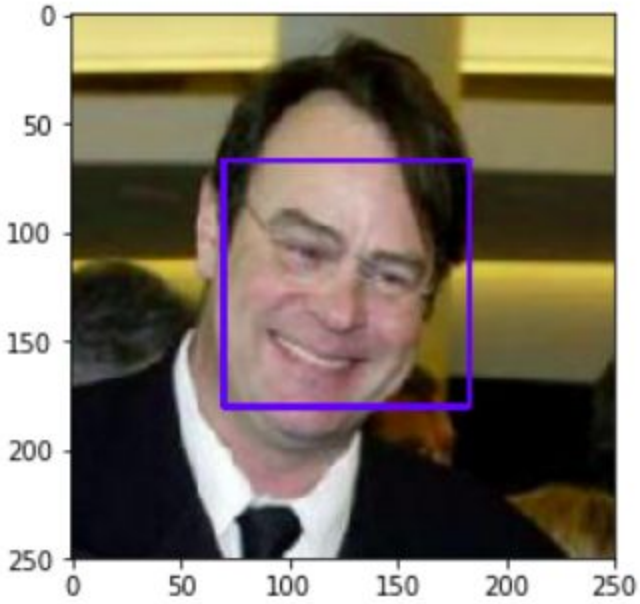
   (The above table is the same as mentioned in the project proposal, review link in the student submission notes)
   From the dataset provided, we can observe that the dataset is not balanced. Some pre-processing is also done which is described in the next section of the report.

4. **Data Preprocessing:**
   Data pre-processing is necessary for every ML problem. The need for data pre-processing arises due to the imbalance dataset.
   Transforms are used to resize the images, ex: RandomResizeCrop (224), RandomHorizontalFlip and RandomRotation (15). The standard input image size for popular models like VGG-16 is of 224X224 generally. Using a similar size would be preferred.
   Normalization is applied to all the 3 datasets.

Train data (train_dataset) - Augmentation is done using the above mentioned transforms to avoid overfitting.
Validation data (val_dataset) - Image Resizing - 224X224
Test data (test_dataset) - Image Resizing - 224X224

5. **Algorithm / Modeling:**
   I have solved the problem statements by using these 4 steps:
   1. **Detecting Human Faces:** The first step is to detect if a human being is present in the input image or not. This is achieved by using OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

```
Number of faces detected: 1
```



The human face detector runs very efficiently and the results are as follows:
Percentage Classified as Human Faces: 98 - From Human folder
Percentage Misclassified as Human Faces: 17 - From dog folder
These results are also available in the dog_app.ipynb

2. **Detecting Dogs:** The first step is to detect if the image input is that of a dog or not. This is accomplished using a pre-trained VGG-16 model which was imported into the code using the torchvision.models.
   Percentage Classified as Dog Faces: 100 - from dog folder
   Percentage Misclassified as Dog Faces: 1 - from human folder
   These results are also available in the dog_app.ipynb
3. **Creating a CNN from scratch without transfer learning:**
   The architecture that is built from scratch for this step looks like this:
   Net(
   (conv1): Conv2d(3, 36, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (conv2): Conv2d(36, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
   (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
   (fc1): Linear(in_features=100352, out_features=512, bias=True)
   (fc2): Linear(in_features=512, out_features=133, bias=True)
   (dropout): Dropout(p=0.25)
   (batch_norm): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
   Activation Function = ReLU - This is decided based on our application at hand. In order to avoid overfitting of the model, a dropout of p=0.25 is added. Finally, we have an output tensor of 113 which equals the breed categories - total classes available for this multi-class classification problem. I have used the cross-entropy loss as the loss function and SGD optimizer and learning rate alpha=0.02. This model after running for 20 epochs has produced a 16% test accuracy (141/836) which is better than the expected 10%.

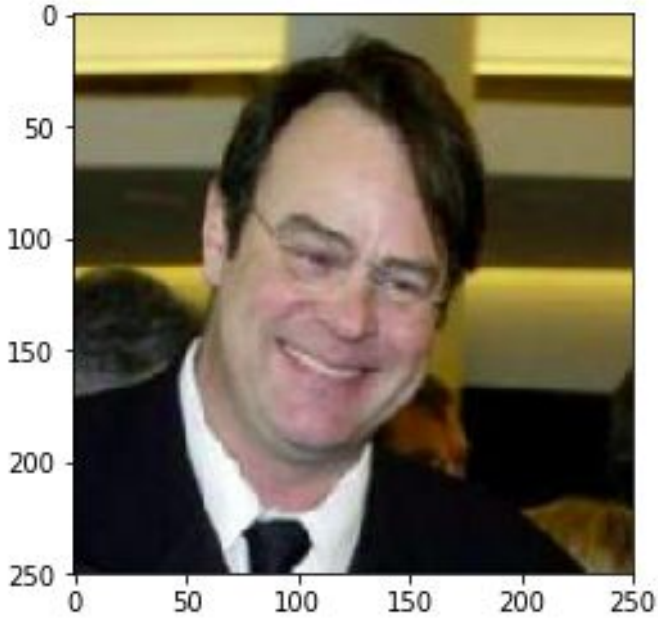
4. **Creating a CNN from scratch using Transfer Learning:**
   I have used a deep neural net to achieve this task.
   Since we need to use transfer learning, I decided to consider a pre-trained model that is already trained on a large image dataset like the ImageNet dataset. So I choose the Resnet101 which is already present in the torchvision.models. I imported the model from there. I managed to achieve a test accuracy of 83% (701/836) which is evident from the cell outputs after training the model for 20 epochs. This accuracy is much higher than the required 60%. The fully connected layer is changed to 133 for the problem at hand since we have 133 breed classes. CrossEntropyLoss is chosen as the loss function with a learning rate of 0.02. These are standard settings for multi-class classification problems.
   This completes the algorithm and modeling section.

6. **<u>Benchmark & Metrics:</u>**
   The expected accuracies are as follows:
   - Scratch CNN without transfer learning = 10%, Result = 16%
   - Scratch CNN with transfer learning = 60%, Result = 83% using Resnet101 architecture

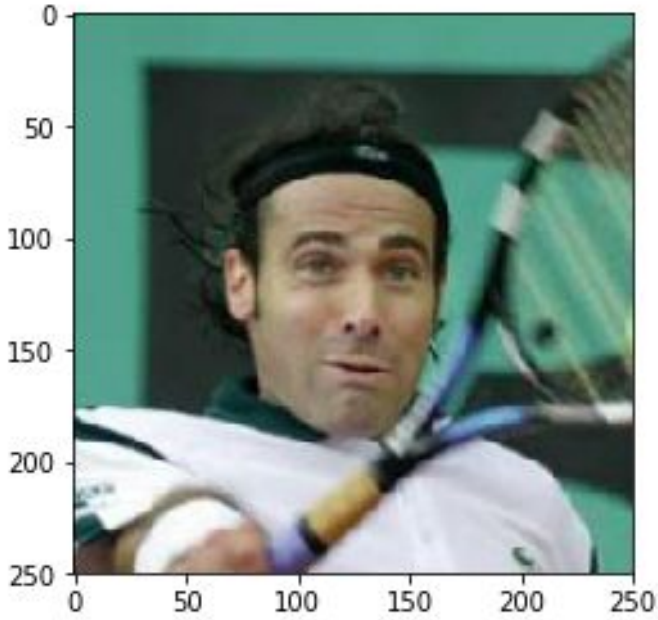
   The loss function selected is the cross entropy loss.
   Accuracy = Correctly classified items / All classified items

7. **<u>Results:</u>**
   The model is performing nicely as seen in the output section of the dog_app.ipynb bottom section. The following are the results/predictions from the trained model. These are the output once someone provides an image input to the app: (Next page)

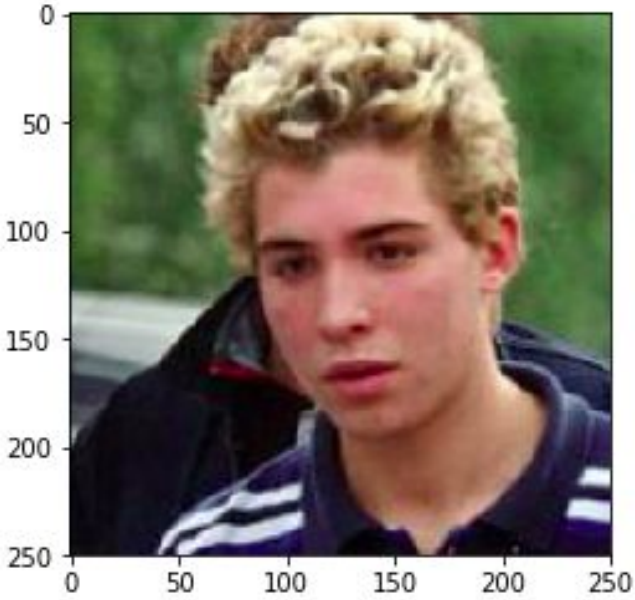
<----------Hello Human!---------->
Your Predicted Breed: Irish wolfhound



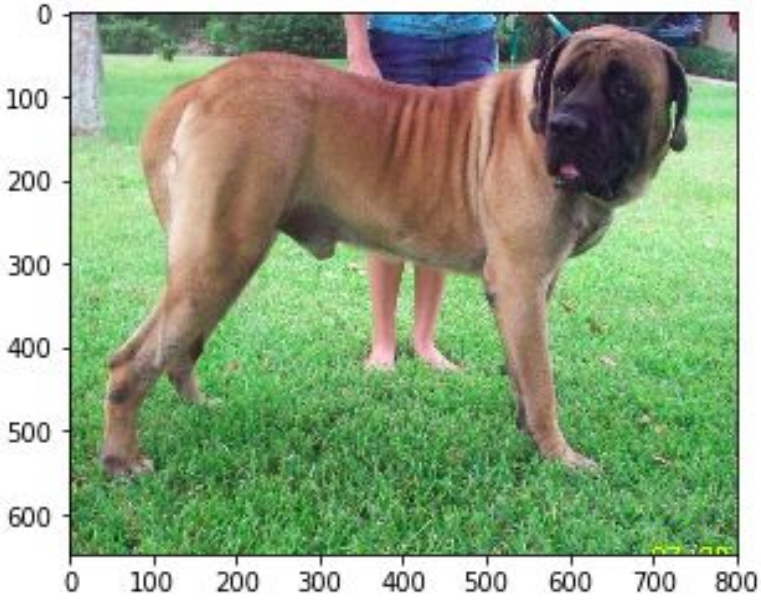
<----------Hello Human!---------->
Your Predicted Breed: Smooth fox terrier



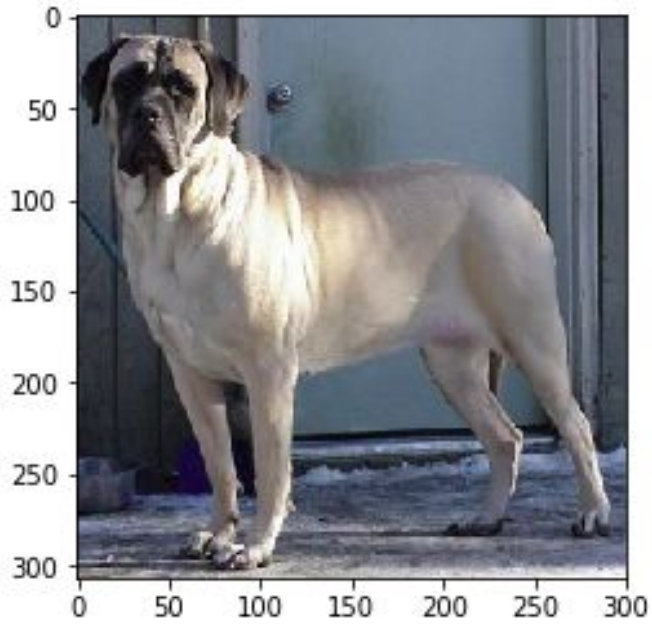
<----------Hello Human!---------->
Your Predicted Breed: Alaskan malamute



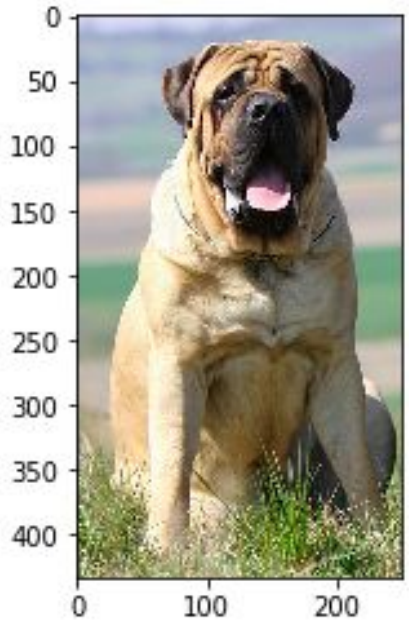
<----------Hello Dog!---------->
Your Predicted Breed: Bullmastiff



<----------Hello Dog!---------->
Your Predicted Breed: Bullmastiff



<----------Hello Dog!---------->
Your Predicted Breed: Mastiff



8. **Conclusion**:

The model that I have developed as mentioned in section (5) works well and surpasses the expected accuracy. This solves our problem statement of correctly classifying an input image to a dog breed class. The achieved accuracy for the model is 83% test accuracy that far exceeds the expected 60%. There are a few things that might help improve the model performance that I have also enlisted in the dog_app.ipynb. Some of them include:

- The model could be trained for higher epochs (>20) that might improve the current metrics.
- Fine-tuning of parameters could be done on the basis of train, valid and test loss curves or metrics.
- Since this can be deployed on a mobile or web app, A/B testing could be done with different sets of parameters at once and then decide on what parameter values to be chosen. This is important as we need to stay relevant with input data and the data available for training.
- Last but not the least, more image augmentation techniques could be used to virtually increase the dataset size, this might be helpful for better model performance.

9. **References**:

1. Pytorch Documentation
2. CNN
3. OpenCV
4. Image Classification