Capstone Project Ankit Kumar Saini

Machine Learning Engineering Nanodegree July 12th, 2021

Dog Breed Classifier Model Using CNN

1. Project Overview

Image classification is a standard computer vision task in the Deep Learning domain. The use of convolutional neural networks (CNNs) have revolutionized the image classification, object detection and segmentation tasks. Dog Breed identification is a well-known multi class image classification task in deep learning. The task was to implement an end-to-end deep learning pipeline that can be used within a web or mobile app to process real-world, user-supplied images. The pipeline accepts any user-supplied image as input and predicts whether a dog or human is present in the image. If a dog is detected in the image, it provides an estimate of the dog's breed. If a human is detected, it provides an estimate of the dog breed that is most resembling. The model can be deployed as a web app using a flask server. I chose this as my capstone project because it allows me to build and deploy a CNN model.

2. Problem Statement

The objective of this capstone project was to build and train a deep learning model that can be deployed as a web or mobile app for real-world use cases. There were two important tasks that needed to be done as part of this project.

- **Dog Detector:** This model was used to identify whether a dog is present in the image or not
- **Human Face Detector:** This model was used to identify whether a human face is present in the image or not

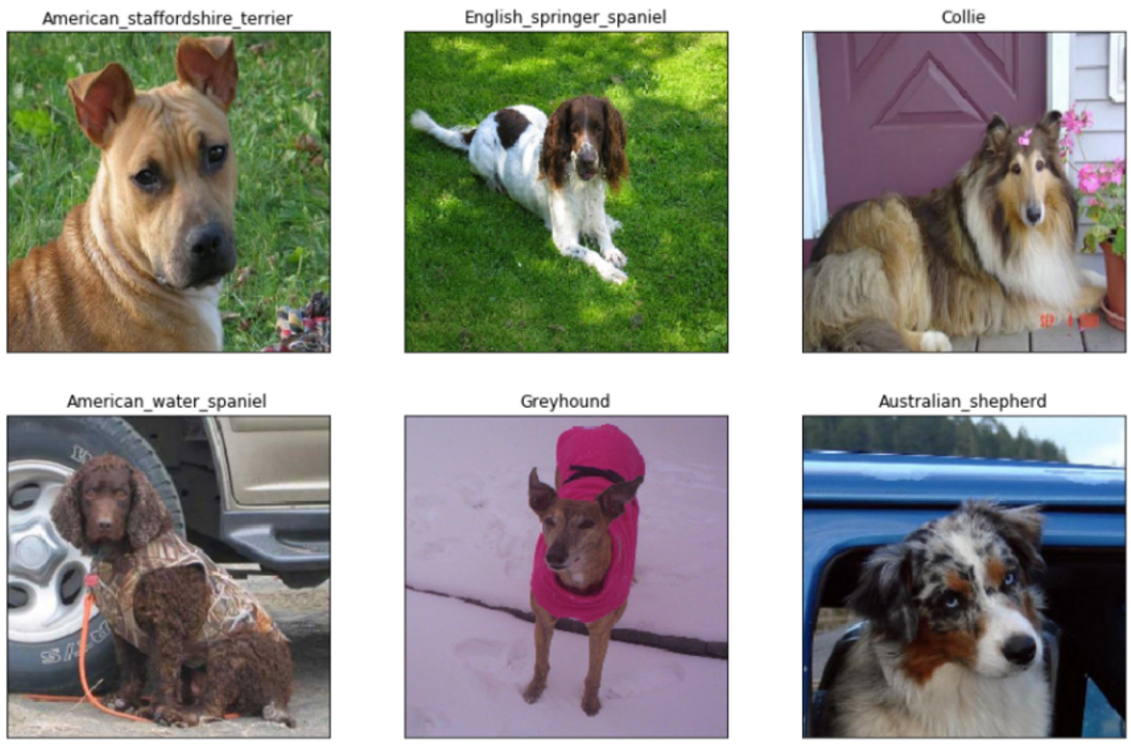
Once the content of the image is identified (dog/human face), the dog breed classifier model was used to predict the dog breed in case of a dog image and the most resembling dog breed in case of a human image. If neither was detected in the image, the algorithm asks the user to input an image containing either dog or human.

3. Data Exploration

For this capstone project, the input data was in the form of RGB images. Two different datasets were provided by Udacity for this project.

- **Human images dataset:** The first dataset was Labeled Faces in the Wild containing 13233 human images. All images were 250x250 pixels in height and width. Most of the images in the dataset contain a single human face. There were several images in the dataset that have half human faces.

- **Dog breed dataset:** The second dataset was the dog breed dataset containing 8351 dog images with 133 dog breeds. It was divided into training (6680 images), validation (835 images) and test (836 images) sets. The images were of different sizes with different backgrounds. The dataset was not balanced. The mean number of images in each class was around 50. There were few classes that have less than 30 images while there were some classes that have more than 70 images. There were few breeds with very similar appearances/features in the dataset that can be confusing for the model to distinguish. Data augmentation methods were used to reduce the impact of data imbalance and confusion among similar breeds.



Sample images from the dataset

4. Data Preprocessing

All CNN models in TensorFlow/Keras require a 4D array/tenor as input with shape (batch_size, image_width, num_channels). The shape of each image needs to be the same for training the CNN model in batches. Therefore the input data for the dog detector model and dog breed classifier model was reshaped so that all the images have the same shape.

Some additional processing steps were required to prepare the 4D tensor for pre-trained ImageNet models. Each model has its own preprocessing function that can be used to prepare images for training. All pre-trained models have the additional normalization step that the mean pixel (expressed in RGB as [103.939, 116.779, 123.68] and calculated from all pixels in all images in ImageNet) must be subtracted from every pixel in each image. Alternatively all pixels in an image could be normalized in the range [0, 1] or [-1, 1] (dividing by 255 or dividing by 127.5 and subtracting 1).

Data augmentation was performed on the training images using ImageDataGenerator. Operations such as rotation, horizontal flip, change in brightness, height shift, width shift, shear, and zoom were performed randomly on the training images to reduce the impact of overfitting.

5. Algorithms and techniques

Convolutional Neural Networks (CNNs) were used in this project for multi class classification problems. CNNs use kernels/filters to automate the task of feature extraction from the images in an incremental manner. Earlier layers in a deep neural network get specialized in detecting lower level features such as edges, texture, corners, high frequencies, etc. and deeper layers detect object level features such as faces and other complex geometrical shapes by combining the lower level features to classify, detect and segment objects in images.

Before jumping directly to transfer learning, a CNN model was built and trained from scratch. The model was simple and it was neither too deep nor too shallow. It has five blocks of Conv2d layer followed by MaxPool2d layer. Two dropout layers were added to the network architecture to avoid overfitting. First dropout layer was added after the third block of Conv2d and MaxPool2d layers and the second was added between two fully connected layers. This model didn’t perform well.

Transfer learning: It is a machine learning method where a model trained for one task can be used as the starting point for another model on a similar task. It is a very popular technique in deep learning to get really good performance with less computation even on very small datasets.

Five different models with pre-trained ImageNet weights were used to classify dog breeds. These models include VGG16, VGG19, ResNet50, DenseNet121 and EfficientNetB4. All these models were trained on the ImageNet dataset. The images in the dog breed dataset were similar to the images in the ImageNet dataset and hence transfer learning would be suitable for this task. These pre-trained models act as very good feature extractors from images and hence used in this breed classification task by transferring what the models have already learned. VGG16, VGG19, ResNet50 and DenseNet121 models were trained in Pytorch while the EfficientNetB4 model was trained in TensorFlow on Google Colab. The details of EfficientNetB4 model training are listed below.

Three important tasks that were performed in this project and the models used for these tasks were:

- **Human Face Detector**: A pre-trained Haar cascade face detector model from the OpenCV library was used to determine if a human face was present in the image or not.

- **Dog Detector:** A pre-trained VGG16 model was used for dog detection. This model has been trained on ImageNet, a very large and popular dataset for image classification and other vision tasks.

- **Dog Breed Classifier Model (EfficientNetB4):** This model was used for feature extraction by removing the classification head (setting the argument include_top = False in the class constructor). A Global Average Pooling (GAP) layer followed by a Dropout layer with 0.3 probability was added on top of the pre-trained model.

  The GAP layer computes the average output for each channel. This reduces the number of trainable parameters and tendency of overfitting. It also speeds up the training process. Averaging operation over the channels of feature maps makes the model more robust to spatial translations in the data. Most importantly, this layer has no trainable parameters. The Dropout layer was added to further reduce the tendency of overfitting. Finally, a Dense classification layer having four output neurons with softmax activation was added. The pre-trained ImageNet weights were freezed so as to prevent them from degradation during training of the model. The only trainable layer in the model was the final dense classification layer.

- **EfficientNetB4 Training**: The model was compiled and trained using Adam optimizer (default parameters values) with sparse categorical crossentropy loss. The metrics to evaluate the model performance during training was the loss on the validation dataset. The number of training epochs were set to 50 and EarlyStopping callback was used to prevent the model from overfitting on the training data. The model training was stopped if the loss of the model on the validation data didn’t decrease by 0.03 for three consecutive epochs.

- **EfficientNetB4 Fine Tuning**: After training the last dense classification layer, the model was trained again but this time by setting the last 20 layers of the model trainable as opposed to during the previous training (all pre-trained layers were set to non-trainable). There were two possible ways that can degrade the pre-trained weights of the model in the process of fine-tuning. First, making the BatchNormalization layers trainable and second, use of a large learning rate. The former will disturb the batch statistics learned by the BatchNormalization layers through training on the imagenet dataset. If these layers were set to trainable then they will try to learn the statistics of the new data. Since the amount of new data is not very large, these layers will not be able to capture the true statistics of the new dataset. This will degrade the pre-trained weights of the BatchNormalization and Convolutional layers. When the learning rate used is large, the updates made to the model weights will also be large. This can degrade the pre-trained weights and can cause instability in training. To avoid both the problems mentioned above, we set the BatchNormalization layers non-trainable and used a small learning rate (1e-4). This helped in achieving better model performance with stable training.

6. Benchmark

- The pre-trained CNN model with ImageNet weights trained using Transfer Learning should achieve an accuracy of 70% or above.
- The CNN model built and trained from scratch should have an accuracy of 10% or above. This will ensure that the model is learning from the data and is not guessing randomly.

7. Metrics

The dog breed dataset was split into training, validation and test sets. The loss (log loss) on validation set during training was used to select the best model out of six different models trained in this project. The best model was evaluated on the test dataset using accuracy, f1-score, precision, recall and confusion matrix. The log loss on the test dataset was also calculated to ascertain the generalization performance of the model.

Accuracy is a good indicator of model performance in case of a balanced dataset. But it is not sufficient to tell in which cases the model is performing good and where it needs improvement. The precision of a class defines how trustable the result is when the model classifies that an image belongs to a particular class. The recall of a class is defined as how well the model is able to detect that class. The F1-score of a class is given by the harmonic mean of precision and recall. It combines precision and recall of a class in one metric. Confusion matrix displays the correct and incorrect classifications predicted by the model.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

TP = True positive

TN = True negative

FP = False positive

FN = False negative

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

8. Model Evaluation and Validation

The human face detector model was evaluated on 200 images (100 images of dogs and 100 images of human faces). It has classified 98 human faces correctly from the first 100 images of human faces and 17 images of dogs were flagged as human faces from the first 100 dog images.
The dog detector model was also evaluated on 200 images (100 images of dogs and 100 images of human faces). It has flagged 2 human faces as dogs from the first 100 images of human faces. It has classified all the 100 images of dogs correctly.

The CNN model built and trained from scratch achieved an accuracy of 13% on the test dataset. The test set accuracy achieved by pre-trained ImageNet models are listed below in the table.

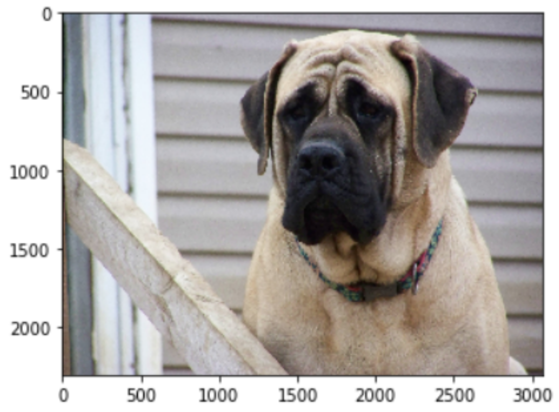
Model Name	Test Accuracy	Framework
DenseNet121	80%	PyTorch
VGG16	81%	PyTorch
ResNet50	81%	PyTorch
VGG19	83%	PyTorch
EfficientNetB4	91%	TensorFlow

The EfficientNetB4 model achieved an average F1-score of 89.3%, precision score of 91.3% and recall score of 89.4%.

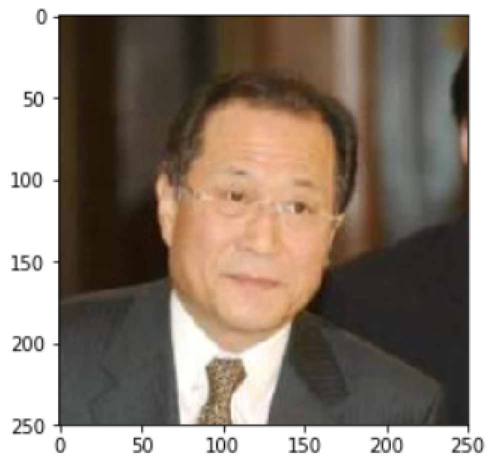
Justification: The EfficientNetB4 model trained in TensorFlow achieved really good results given the dataset was not balanced and it was a small dataset. This performance was achievable because of transfer learning. Incremental improvements in performance were obtained using data augmentation and model fine-tuning.

Results: Results generated by the algorithm

```
This dog breed is ...
Predicted breed: Mastiff
```



```
Hello! You look like a ...
Predicted breed: Basenji
```



Improvements: The performance of model can be improved further by taking the actions mentioned below

- Get more number of images per class
- Make the dataset balanced
- Use image augmentation methods such as CutOut, MixUp and CutMix
- Use VAEs/GANs to generate artificial data
- Use more efficient algorithms such as EfficientNet
- Use activation maps to interpret the model

9. References

1. EfficientNet paper: arXiv:1905.11946
2. OpenCV cascade classifier: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
3. ResNet50: https://github.com/keras-team/keras-applications/releases/download/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5'
4. TensorFlow documentation: https://www.tensorflow.org/api_docs
5. PyTorch documentation: https://pytorch.org/docs/stable/index.html