Udacity Machine Learning Engineer Nanodegree
Capstone Project Report

Dog Breed Classification
Using Convolutional Neural Network

Ersin Genel
3 April 2021

I. Definition

Project Overview

According to the World Canine Organisation (Fédération Cynologique Internationale), there are over 300 recognised dog breeds in the world. In this project, we aim to determine the canine's breed using an image. More interestingly, if an image of a human is provided instead, our algorithm identifies the resembling dog breed.

CNN (Convolutional Neural Network) can be used to solve these kinds of classification problems. In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery which perfectly fits our project.

Problem Statement

We want to build a machine learning model that uses images as input. It should return an estimation for the breed of the dog if there is a dog in the image. If a human is detected in the image the algorithm should find the dog breed that most closely resembles. Our model can be consumed by a web application, web service or mobile application when ready. It can be helpful if the model can handle some cases such as user-supplied images without a dog or human.

As a first step, we need to understand what is provided by the user in an image. So we need to find a way to detect the object. There can be a dog or a human, or neither. Then we aim to estimate the canine's breed if it’s a dog, or estimate the resembling dog breed if it’s a human in the picture.

Metrics

As a common approach to test the accuracy of a model, we compare total images count to successfully classified images count. In order to evaluate our machine learning algorithm, Multi Class Log Loss used to handle class imbalances.

As I see the same evaluation metrics used for a similar Kaggle competition which can be very helpful to compare: https://www.kaggle.com/c/dog-breed-identification/overview/evaluation

II. Analysis

Data Exploration

Input data will be images in this project as we can easily guess. There should be images of dogs and humans. Fortunately, all resources we need are provided by Udacity. We have two datasets:

Human Dataset: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip

There are 13233 human images in this dataset that will be used to detect human faces. We have 5749 folders that contain different counts and one dimension (250x250 px) of human faces. There are different colours and backgrounds in the images. Human images are imbalanced.

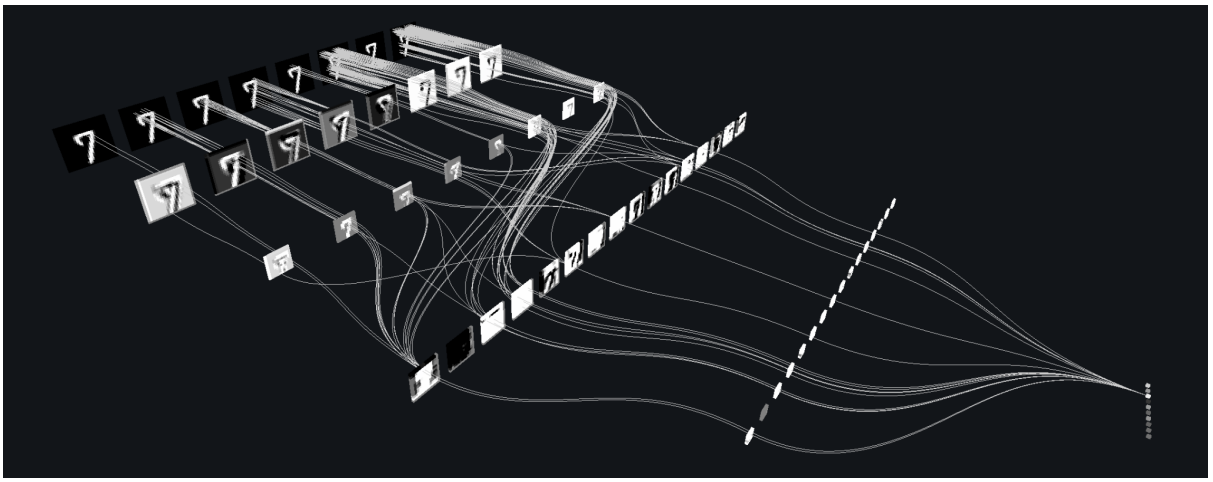
Dog Dataset: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip

There are 8351 dog images in this dataset that will be used to detect dog's breeds. We have 133 folders that contain different counts and dimensions of a dog breed. There are different colours and backgrounds in the images, and also two or more dogs in an image. Dog images are imbalanced.

You can find some human and dog images in below:

Algorithms and Techniques

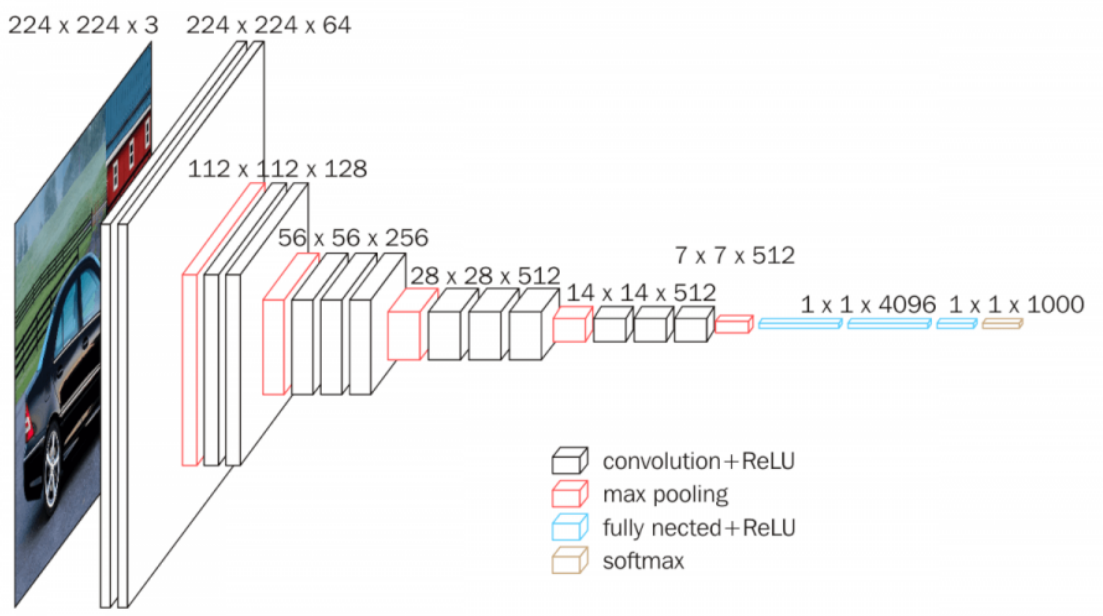
We aim to use CNN (Convolutional Neural Network) to solve this problem. A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.
The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.



We use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images. Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, “Rapid Object Detection using a Boosted Cascade of Simple Features” in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

We use a pre-trained model, VGG-16 to detect dogs in images. VGG16 is a convolution neural net (CNN) architecture which was used to win ILSVR (Imagenet) competition in 2014. It is considered to be one of the excellent vision model architecture till date. Most unique thing about VGG16 is that instead of having a large number of hyper-parameter they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC(fully connected layers) followed by a softmax for output. The 16 in VGG16 refers to it has 16 layers that have weights. This network is a pretty large network and it has about 138 million (approx) parameters.

You can see the architecture of VGG16 in below:



Benchmark

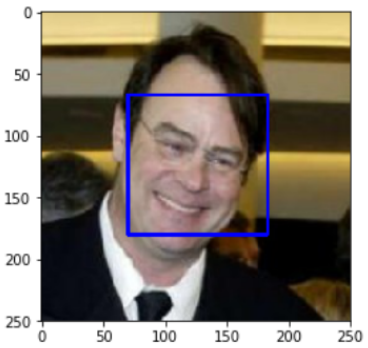
The CNN model was built from scratch, used as a benchmark model in the project. That model must obtain a test accuracy of at least 10%, the CNN model using transfer learning must obtain at least 60% accuracy.

III. Methodology

Detect Humans

After importing the human dataset from the provided sources, we detect the human faces in images using Haar feature-based cascade classifiers. OpenCV provides many pre-trained face detectors. We have downloaded one of these detectors and stored it in the haarcascades directory. The images were converted to greyscale as a standard process firstly. Then we get an array that contains the faces results.

You can see an output below:



Before testing we create a face detector method to process an image and return the count of faces in the image. You can see the face detector test results for first 100 human and dog images below:

- `Face detected in 98% of the first 100 images in human_files.`
- `Face detected in 17% of the first 100 images in dog_files.`

Detect Dogs

After importing the dog dataset from the provided sources, we detect the dogs in images using a pre-trained model, VGG-16, along with weights that have been trained on ImageNet.

While looking at the dictionary, we find that the categories corresponding to dogs appear in an uninterrupted sequence and correspond to dictionary keys 151-268, inclusive, to include all categories from 'Chihuahua' to 'Mexican hairless'. Thus, in order to check to see if an image is predicted to contain a dog by the pre-trained VGG-16 model, we need only check if the pre-trained model predicts an index between 151 and 268.

Before testing we create a dog detector method that uses the information above to process an image and return whether a dog detected in the image. You can see the dog detector test results for first 100 human and dog images below:

- `Face detected in 1% of the first 100 images in human_files.`
- `Face detected in 100% of the first 100 images in dog_files.`

Create a CNN from Scratch to Classify Dog Breeds

After detecting humans or dogs in the images, we need to create a CNN from scratch to classify dog breeds that must obtain a test accuracy of at least 10%. Firstly, we need to specify data loaders for the dog dataset, we also preprocess the data as below.

- We crop the training images to random size and aspect ratio, and resize to 224x224. We also horizontally flip the training images with the default probability of 0.5.
- We resize the validation and test images to 224x224 and crop the image at the center to 224x224, if image size is smaller than 224x224 along any edge, image is padded with 0 and then center cropped.
- We resize the images to 224x224 since we are dealing with different image sizes.
- We augment the dataset using the RandomResizedCrop method to crop the given image randomly, and the RandomHorizontalFlip method to horizontally flip the given image randomly, and the RandomRotation method to rotate the given image by angle.

We implement the CNN architecture as below:

- We add a 2D convolutional layer, taking in 3 input channels outputting 16 convolutional features, with a square kernel size of 3, stride of 1 and padding of 1.
- Then add a 2D convolutional layer, taking in 16 input channels outputting 32 convolutional features, with a square kernel size of 3, stride of 1 and padding of 1.

- Then add a 2D convolutional layer, taking in 32 input channels outputting 64 convolutional features, with a square kernel size of 3, stride of 2 and padding of 1.
- After adding three convolutional layers, we apply a 2D max pooling over convolutional layers to reduce dimensions.
- We apply the RELU function to make the network non-linear.
- We add two fully connected layers that outputs 512 and 133 labels.
- We apply a 2D dropout to promote independence between feature maps.
- We add 2D batch normalization to convolutional layers and 1D batch normalization to the first fully connected layer to normalize the activations of the network between layers.

We specify the loss function as CrossEntropyLoss and the optimizer as SGD. Then we train the model (30 Epochs). You can see the output below:

- `Epoch: 1  Training Loss: 4.806272  Validation Loss: 4.598965`
- `Epoch: 2  Training Loss: 4.620492  Validation Loss: 4.442835`
- `Epoch: 3  Training Loss: 4.564020  Validation Loss: 4.360752`
- `Epoch: 4  Training Loss: 4.523397  Validation Loss: 4.359308`
- `Epoch: 5  Training Loss: 4.498525  Validation Loss: 4.225847`
- `Epoch: 6  Training Loss: 4.468140  Validation Loss: 4.239058`
- `Epoch: 7  Training Loss: 4.435859  Validation Loss: 4.227190`
- `Epoch: 8  Training Loss: 4.429532  Validation Loss: 4.175518`
- `Epoch: 9  Training Loss: 4.400808  Validation Loss: 4.135453`
- `Epoch: 10  Training Loss: 4.394260  Validation Loss: 4.210156`
- `Epoch: 11  Training Loss: 4.349110  Validation Loss: 4.074286`
- `Epoch: 12  Training Loss: 4.321904  Validation Loss: 4.060047`
- `Epoch: 14  Training Loss: 4.294847  Validation Loss: 4.129696`
- `Epoch: 15  Training Loss: 4.290298  Validation Loss: 3.995526`
- `Epoch: 16  Training Loss: 4.264571  Validation Loss: 3.994423`
- `Epoch: 17  Training Loss: 4.231234  Validation Loss: 4.032627`
- `Epoch: 18  Training Loss: 4.232930  Validation Loss: 3.901232`
- `Epoch: 19  Training Loss: 4.202550  Validation Loss: 3.905613`
- `Epoch: 20  Training Loss: 4.225564  Validation Loss: 3.844921`
- `Epoch: 21  Training Loss: 4.172461  Validation Loss: 3.951441`
- `Epoch: 22  Training Loss: 4.187317  Validation Loss: 3.877223`
- `Epoch: 23  Training Loss: 4.161955  Validation Loss: 3.760353`
- `Epoch: 24  Training Loss: 4.147604  Validation Loss: 3.824188`
- `Epoch: 25  Training Loss: 4.127481  Validation Loss: 3.768083`
- `Epoch: 26  Training Loss: 4.121225  Validation Loss: 3.719263`
- `Epoch: 27  Training Loss: 4.128139  Validation Loss: 3.754927`
- `Epoch: 28  Training Loss: 4.094692  Validation Loss: 3.682965`
- `Epoch: 29  Training Loss: 4.115124  Validation Loss: 3.714743`
- `Epoch: 30  Training Loss: 4.093284  Validation Loss: 3.677129`

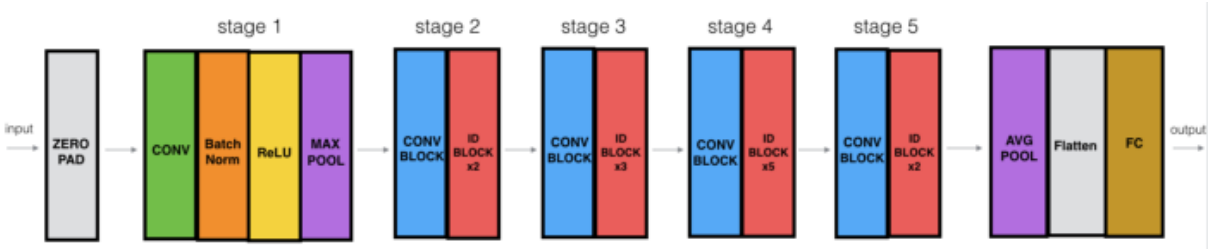
The test accuracy is 13% (112/836) for the model.

Create a CNN to Classify Dog Breeds Using Transfer Learning

We now use transfer learning to create a CNN that can identify dog breed from images. Our CNN must attain at least 60% accuracy on the test set.

We prefer to use Residual Networks which is a popular neural network used as a backbone for many computer vision tasks. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150+ layers successfully. We use ResNet50 which was mentioned before as an alternative pre-trained network to VGG-16. The ResNet-50 model consists of 5 stages each with a convolution and identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers. The ResNet-50 has over 23 million trainable parameters.

You can see the architecture of ResNet-50 below:



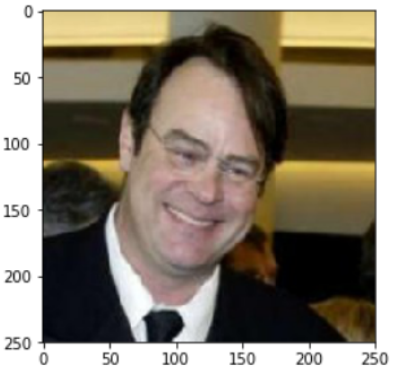
We specify the loss function as CrossEntropyLoss and the optimizer as SGD as before. Then we train the model (10 Epochs). You can see the output below:

- `Epoch: 1  Training Loss: 8.218162  Validation Loss: 4.107317`
- `Epoch: 2  Training Loss: 7.613813  Validation Loss: 4.166151`
- `Epoch: 3  Training Loss: 6.832040  Validation Loss: 3.577013`
- `Epoch: 4  Training Loss: 6.979835  Validation Loss: 3.820831`
- `Epoch: 5  Training Loss: 7.007840  Validation Loss: 3.319883`
- `Epoch: 6  Training Loss: 6.788256  Validation Loss: 3.490227`
- `Epoch: 7  Training Loss: 6.746976  Validation Loss: 3.529666`
- `Epoch: 8  Training Loss: 6.589819  Validation Loss: 3.486921`
- `Epoch: 9  Training Loss: 6.615377  Validation Loss: 3.327050`
- `Epoch: 10  Training Loss: 6.207893  Validation Loss: 3.790337`

The test accuracy is 75% (629/836) for the model. Then we create a simple code to evaluate our models.

You can see the output below:

```
Image Path: /data/lfw/Dan_Ackroyd/Dan_Ackroyd_0001.jpg
Result: Beagle
```

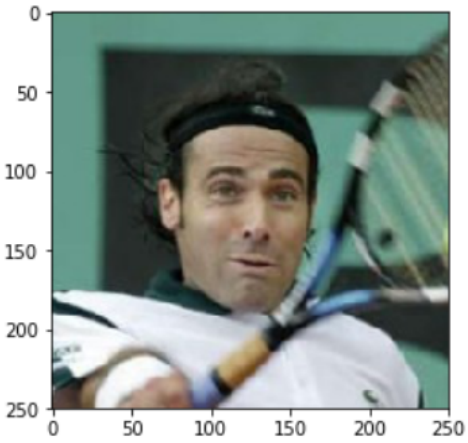


Lastly, we create a method to test our algorithm as below:

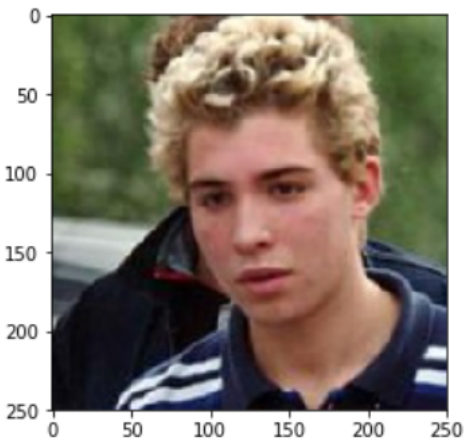
```python
def run_app(img_path):
    result = 'It seems the image does not contain a human or dog face.'
    if dog_detector(img_path):
        breed_transfer_result = predict_breed_transfer(img_path)
        result = f'Dog prediction: {breed_transfer_result}'
    elif face_detector(img_path):
        breed_transfer_result = predict_breed_transfer(img_path)
        result = f'Human prediction: {breed_transfer_result}'

    print(result)
    image = mpimg.imread(img_path)
    imgplot = plt.imshow(image)
    plt.show()
```

You can see some predictions below:

```
Human prediction: Bull terrier
```



```
Human prediction: American water spaniel
```



IV. Results

In conclusion, the accuracy for the CNN from Scratch (without transfer learning) is 13% (112/836), and the accuracy for the CNN using transfer learning is 75% (629/836) even with 10 Epochs, the output is better than I expected.

V. Conclusion

In this project, we detected human and dog faces in an image, then we created a method that returns an estimation of the canine's breed if it’s a dog, or an estimation of resembling dog breed if it’s a human.

We can still make some improvements in our model for a better performance:

- We can tune parameters such as number of epochs, learning rate, optimizer parameters etc.
- We can try to identify overfitting or underfitting issues. Training with more data, cross validation, early stopping and adding more layers can help us to find a good balance.
- We can apply more preprocessing steps like data augmentation.

References

https://en.wikipedia.org/wiki/Dog_breed
http://www.fci.be/en/Presentation-of-our-organisation-4.html
https://en.wikipedia.org/wiki/Convolutional_neural_network
https://docs.opencv.org/master/db/d28/tutorial_cascade_classifier.html
https://developers.google.com/machine-learning/crash-course/classification/accuracy
https://neurohive.io/en/popular-networks/vgg16/
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
https://neurohive.io/en/popular-networks/vgg16/