Udacity Machine Learning Nanodegree — Chelsea Friday

Capstone Project Report — October 2020

# I. Definition

## Project Overview

Have you ever been on a walk and spotted a dog that you were not quite sure of the breed? Well I have! I live across from a dog park and am always spotting dogs which I can't quite distinguish if they are this breed or that? Think, a Boston Terrier vs. a French Bulldog. Or, an Alaskan Malamute vs. Siberian Husky.

For this classification problem a Convolutional Neural Networks (CNN) will be used. A CNN is a deep learning algorithm which can take in an input, an image, and assign it a class, like a dog breed. CNNs represent a huge breakthrough in image recognition. They are most commonly used to analyze visual imagery because they are fast and efficient. As we can see from the table below taken from the paper "Deep learning and transfer learning approaches for image classification"[4], CNNs can be applied to many different types of datasets with relatively high accuracy.

| No. | Methods used | Data sets used | Training (Images/%) | Testing (Images/%) | Accuracy |
|---|---|---|---|---|---|
| 1 | Transfer learning with AlexNet, GoogleNet,ResNet50 | CIFAR10 | 50000 | 10000 | GoogleNet-68.95%, ResNet50-52.55%, AlexNet-13% |
| | | CIFAR100 | 50000 | 10000 | |
| 2 | Proposed a network contains 5 convolutional and 3 fully connected layers | ImageNet Fall 2011, 15M images, 22K categories | 7.5M | 7.5M | Error rates top-5 : 37.5%, top-1 : 17.0%. |
| 3 | Transfer learning, web data augmentation technique with Alex,vgg16,res net-152 | Flowers102 | 8189 | | 92.5% |
| | | dogs | 20580 | | 79.8% |
| | | Caltech101 | 9146 | | 89.3% |
| | | event8 | 1579 | | 95.1% |
| | | 15scene | 4485 | | 90.6% |
| | | 67 Indore scene | 15620 | | 72.1% |
| 4 | Transfer Learning with Inceptiov3 model | CIFAR10 | 50000 | 10000 | 70.1% |
| | | Caltech Face | 12150 | | 65.7% |
| | | | | | 500 epochs-91% |
| | | | | | 4000 epochs-96.5% |
| 5 | CNN deep learning | Caltech 101 | 9146 | | 96% |
| 6 | Compare NN models | DR | 77% | 23% | LeNet-72%, VGGNet-76% |
| | | CT | 72% | 28% | LeNet-71%, VGGNet-78% |
| 7 | An autonomous learning algorithm automatically generate Genetic DCNN architecture | MNIST, CIF AR10, CIFAR100 | 90% | 10% | 99.72% on MNIST, 89.23% on CIFAR10, 66.70% on CIFAR100 |
| 8 | Google's Inception-v3 | Dogs | | | 96% |
| 9 | CNN + AdaBoost | CIFAR-10 | 80% | 20% | 88.4% |

**Problem Statement**

The main objective of this project is to build a machine learning model that can be used within a web app to process real-world, user-supplied images. Given an image of a dog, the algorithm will return a predicted dog breed. Given an image of a human, the algorithm will identify which dog breed the human face most closely resembles. To achieve this intended solution the following steps must be executed:

Step 1: Import datasets and libraries.

Step 2: Use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

Step 3: Create a dog detector using the pre-trained VGG-16 model along with weights that have been trained on ImageNet.

Step 4: Create a CNN to classify dog breeds from scratch. Train, validate, and test model.

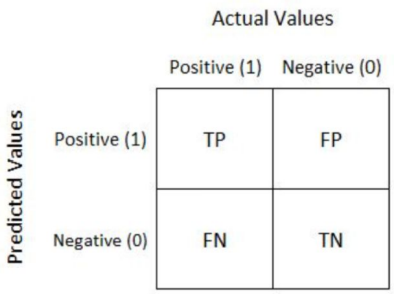
Step 5: Create a CNN to classify dog breeds using Transfer Learning. Train, validate, and test model.

Step 6: Write and test an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then,

- If a dog is detected in the image, return the predicted breed.
- If a human is detected in the image, return the resembling dog breed.
- If neither a dog nor human detected in the image, provide output that indicates an error.

**Metrics**

Since we are dealing with a multi-classification problem that is relatively well balanced we will use accuracy as our evaluation metric. In general, the accuracy metric measures the ratio of correct predictions over the total number of instances evaluated.




$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

## II. Analysis

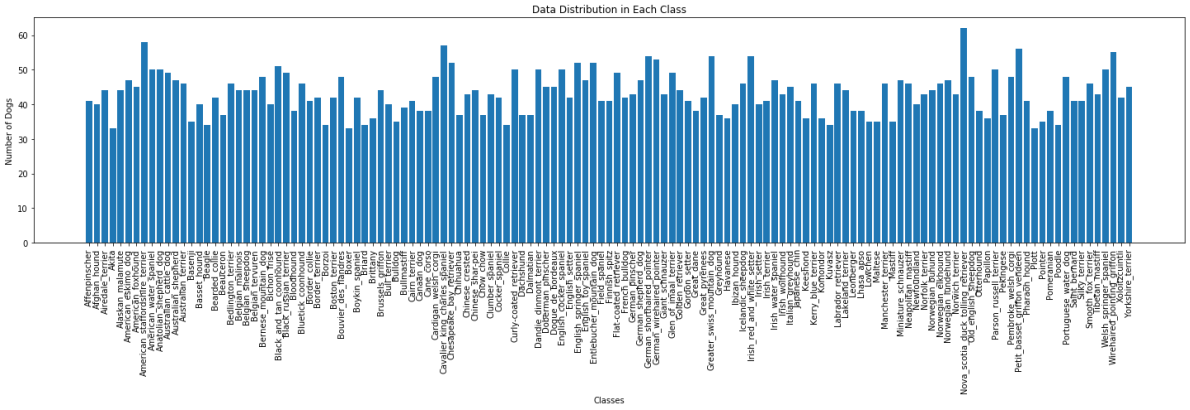
### Data Exploration

There are two datasets for this project provided by Udacity. The dog images dataset has 8,351 total images. This dataset is sorted into 6,680 train images, 836 test images, and 835 validation images. Each has 133 folders corresponding to 133 dog breeds. The human images dataset contains 13,233 total images. The images will be used as input data to create a feature map to feed into the CNN.

*Sample Images*

### Exploratory Visualization

We can take a look at the distribution of data in the dog images dataset across the 133 different classes. We need to take a look at the data distribution to determine whether data is balanced or imbalanced. This will help to determine if we need to add images to the dataset to offset any imbalance and which evaluation metrics will be appropriate. When the distribution is uneven by a small amount (e.g. 4:6) in the training dataset the data is slightly imbalanced. When the distribution is uneven by a large amount (e.g. 1:100 or more) in the training dataset the data is severely imbalanced. Any dataset with an unequal class distribution is technically imbalanced. However, a slight imbalance is often not a concern. In this case, the data in each class is balanced enough to work with without needing to apply any techniques to balance the data. We can observe this by looking at the graph below.




*Plot of data distribution in each class*

**Algorithms and Techniques**

For this problem a CNN is used because it has proven to be an effective algorithm for the task of image processing and classification. A CNN convolves learned features with input data and uses 2D convolutional layers. This means that this type of network is ideal for processing our 2D images of humans and dogs.

A CNN works by extracting features from images. Thus, in order to create CNN models from scratch we first must create the feature detectors. We have the image inputs - the human and dog image datasets. We will use the human dataset and existing Haar feature based Cascade Classifier to create a human face detector. The Cascade Classifier is an effective object detection method where a cascade function is trained from positive and negative images. It is then used to detect objects, in this case the presence of a human face. Before using the face detector, it is standard to convert the images to grayscale. The detectMultiScale function executes the classifier stored in face_cascade and takes the grayscale image as a parameter.

We will use the dog image dataset and a pre-trained VGG16 model to create a dog face detector. Given an image, this pre-trained VGG16 model returns a prediction derived from the 1000 possible categories in ImageNet for the object that is contained in the image. Before we use the pre-trained model, we will need to appropriately pre-process tensors. The dog images will need to be resized to 224x224 and normalized using transform.Resize and transform.Normalize respectively.

With both the Dog Detector and Human Face Detector created, we can take in an image as input and apply the detector to create a feature map. This feature map will be used with the created CNN model that will be trained through forward and back propagation.

### Benchmark

We will be using the CNN model created from scratch as a benchmark model to compare to the CNN model created using Transfer Learning. The CNN model created from scratch must attain a test 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%.

## III. Methodology

### Data Preprocessing

As discussed in the data exploration, the data in each class is balanced enough to work with without needing to apply any techniques to balance the data. Moving forward, the images will need to be resized to match the input size requirements of the pre-trained model used and converted to grayscale.

The pre-trained VGG16 model used in this project takes an image size of 224x224. All images in the test, train, and validation datasets have been resized to 224x224 to fit this parameter. For the training data, images were transformed by applying a random horizontal flip function and a random rotation function to reduce overfitting. All images were converted into an image tensor and normalized. Normalization helps CNNs perform better by getting images into a selected range and reducing the skewness.

### Implementation

To get to a solution the follow steps must be implemented in said order:

Step 1: Create functions for detecting humans and dog images. Refer to algorithms and techniques under analysis to see how this was achieved.

Step 2: Write three separate data loaders for the training, validation, and test datasets of dog images. Set batch size to 20. Set the number of subprocesses to use for data loading to zero. Specify appropriate transforms.

Step 3: Model Architecture.

We created a CNN model from scratch to classify dog breeds. The model was first created using three convolutional layers with a dropout probability of 0.5. After training, the model yielded a test accuracy of 9% and was overfitting the test data. To increase our test accuracy and reduce overfitting, two more convolutional layers were added and the dropout

probability was dropped to 0.25. The final model design included five convolutional layers with max-pooling in between to extract the most important features from the convolution layers. The max pooling layers function to reduce the x-y size of the input by keeping only the most active pixels from the previous layer. The pooling layer of 2,2 is used to reduce the input size by 2 in this case. Also, ReLu activation function was applied to increase non-linearity. We flattened the pooled feature map (eventual output of our series of convolutional and pooling layers) into one long vector. The vector was put into our fully connected layer. Three fully connected layers were created with the last fully connected layer resulting in 133 nodes.

Step 4: Define loss function and optimizer. Setting the learning rate to 0.01 and the momentum to 0.9

Step 5: Train and validate model, saving the final model parameters at filepath 'model_scratch.pt' and logging the validation/training loss and validation accuracy.

Step 6: Test the model on the test dataset of dog images.

**Refinement**

The initial model with three convolutional layers resulted in a test accuracy of 9%. This did not meet the benchmark, 10%. To address this, two more convolutional layers with pooling were added and the dropout probability was decreased from 0.5 to 0.25, as discussed in the implementation process. This resulted in a test accuracy of 24% (206/836) with a test loss of 3.212. This surpassed the benchmark.

To refine the solution even further another CNN model was created using transfer learning. The ResNet50 architecture was used to create deeper layers, 50 layers, and improve the performance of the scratch CNN model previously created. ResNets, like the VGG16 model used to create the Dog Detector, is pre-trained on the ImageNet dataset and takes in a 224x224 sized image. After training for five epochs the test accuracy increased to 80% (677/836) with a test loss of 0.639.

## IV. Results

**Model Evaluation and Validation**

Our benchmark model, the CNN model created from scratch trained for 30 epochs and produced a test accuracy of 24%. It correctly classified 206 dog images out of the 836 in the testing dataset. This was higher than the 10% accuracy Udacity recommended be achieved. However, it was noticed that after 21 epochs the validation loss stopped decreasing and began to slowly increase while the training loss continually decreased. This suggests overfitting.

The CNN model created using transfer learning with ResNet50 was trained for five epochs. During a single training step it viewed 20 toal images and learned at a rate of 0.01 with a momentum of 0.9. The model produced a test accuracy of 80%. It correctly classified 677 dog images out of the 836 in the testing dataset. This surpassed the benchmark model. However, upon testing the model with images outside of those supplied in the training and testing datasets, the model did not correctly classify three dog breeds out of the ten images it was supplied. Which is only a 70% accuracy. More images will need to be supplied to test the model even further because ten images is too small to clearly look at the effectiveness of the model.

**Justification**

After the implementation and training of the transfer learning model for a total of five epochs, we evaluated the performance of the model using the test function:

test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)

We saw that the model achieved an accuracy of 80% which is larger than the benchmark models 24% and the 60% Udacity requested we achieve. The final solution, as will be discussed next, fits expectations and solves the problem of dog breed classification.

## V. Conclusion

**Reflection**

As discussed previously, the steps taken to solve this classification problem include:

Step 1: Identify a problem, import relevant datasets and libraries.

Step 2: Use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

Step 3: Create a dog detector using the pre-trained VGG-16 model along with weights that have been trained on ImageNet.

Step 4: Create a CNN to classify dog breeds from scratch. Train, validate, and test model.

Step 5: Create a CNN to classify dog breeds using Transfer Learning and ResNet50 model architecture. Train, validate, and test model.

Step 6: Write and test an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then,

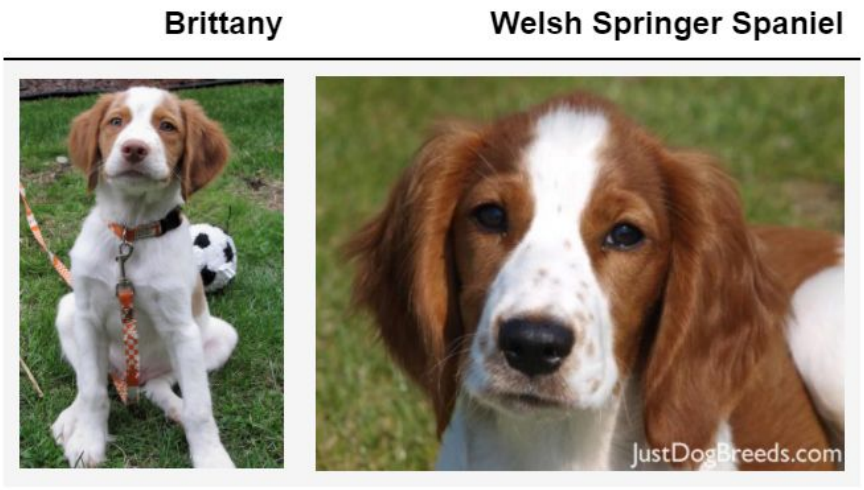
- If a dog is detected in the image, return the predicted breed.
- If a human is detected in the image, return the resembling dog breed.
- If neither a dog nor human detected in the image, provide output that indicates an error.

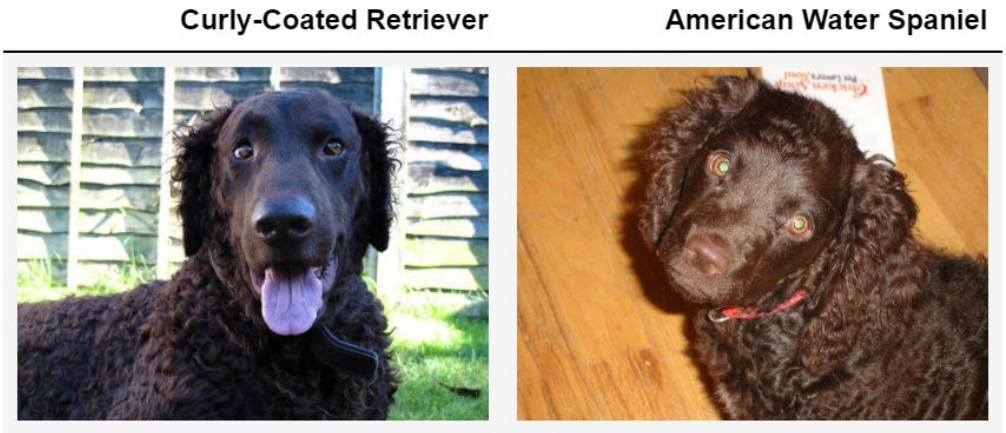
The most difficult step of this process was creating a CNN model from scratch. It required a lot of trial and error. CNNs have an input layer, output layer and hidden layers. The hidden layers usually consist of convolutional layers, ReLU layers, pooling layers and fully connected layers. The classic CNN architecture usually looks like:

Input -> Convolution -> ReLU -> Convolution -> ReLU -> Pooling -> ReLU -> Convolution -> Pooling -> Fully Connected

Determining how many layers to include, the stride, kernel size, and pooling was the longest process of this project. At first the model produced a EOF error because of how the code was written. Second, the model did not reach the 10% accuracy benchmark and layers had to be added and adjusted. After adjustments, the model produced better accuracy but had started to overfit the data just after ten epochs. We can see where the model began to overfit by comparing the validation loss and test loss. If test loss continues to decrease while validation goes up, data is overfitting and training should be stopped. To fix this, the dropout probability was adjusted from 0.5 to 0.25. The model trained for 30 epochs and we observed overfitting after 21 epochs. Since this is our benchmark model and we planned to use transfer learning to create a second CNN model for our solution, it was determined that for this case the scratch model would suffice.

In general, the task of classifying dog breeds from images is exceptionally difficult because even a human has trouble distinguishing between cousin breeds, like the Brittany and Welsh Springer Spaniel, or, like the Curly-Coated Retriever and American Water Spaniel.






At its core, a CNN model mimics the organization of the visual cortex and works similar to how neurons respond to visual stimuli in the receptive fields making it ideal for image recognition tasks. The final CNN model fit expectations for the problem because it produced 80% test accuracy, higher than the 60% Udacity requested the model achieve. And, quite possibly, higher than an average human could achieve given the difficulties of the task. This model can, and is, frequently used to solve image classification problems. The ResNet 50 architecture is widely used for transfer learning because the model has useful learned features, is easily accessible and is known to be an effective solution for the image recognition task.

In all, this means we have created an effective model that can determine between a dog and human face, classify a dog breed 80% of time when a dog is detected and assign a dog breed which a human face most closely resembles when a human face is detected.

**Improvement**

Though the final solution resulted in 80% test accuracy, the model could be improved in a number of ways:

- Add more images to the training and test data. The current solution was trained on 6,680 images and tested on 836 images.
- Perform more image augmentation to avoid overfitting and improve accuracy.
- Train the ResNet50-CNN model on more epochs.
- Use a different model architecture like VGG19, ResNet101, GoogleNet, and etc.
- Choose a different optimizer function, adjust learning rate and/or adjust momentum.

## VI. References

[1] Brownlee, J. (2020) How to Choose Loss Functions When Training Deep Learning Neural Networks. *Machine Learning Mastery* [Online]. Available: https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

[2] Kaur, T., Gandhi, T.K. (2020) Deep convolutional neural networks with transfer learning for automated brain image classification. *Machine Vision and Applications* 31, 20 [Online]. Available: https://doi.org/10.1007/s00138-020-01069-2

[3] Krishna, S., Kalluri, H. (2019) Deep learning and transfer learning approaches for image classification. International Journal of Recent Technology and Engineering Vol. 7, Issue. 5S4 [Online]. Available: https://www.ijrte.org/wp-content/uploads/papers/v7i5s4/E10900275S419.pdf

[4] Han, D., Liu, Q., Fan, W. (2017) A New Image Classification Method Using CNN transfer learning and Web Data Augmentation. *Expert Systems with Applications. Elsevier* Vol. 95, pp. 43-56 [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0957417417307844

[5] OpenCV (2013) Tutorial Cascade Classifier [Online]. Available: https://docs.opencv.org/master/db/d28/tutorial_cascade_classifier.html

[6] PyTorch (2020) Torchvision Models. *torchvision.models - PyTorch 1.6.0 documentation* [Online]. Available: https://pytorch.org/docs/stable/torchvision/models.html

[7] Udacity (2019) project-dog-classification [Source code]. Available: https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification