Machine Learning Engineer Capstone Project

CNN Project: Dog Breed Classifier

Tingting Zhou

November 16, 2020

1. Project Background

Nowadays, Machine Learning could be one of the most popular words, people may hear it through various scenarios, such as commercials, finance news and some advanced medical fields. However, we may still think Machine Learning application is still a bit far from real life, although we've already used Alexander to play music for us. Mainly it's because most of people don't have the access to machine learning skills or have the opportunity to build a machine learning application from scratch, and most of the related topics are full of notations and we can not implement the algorithm into real life. Therefore, it interests me that if I can use deep learning algorithm to build applications that close to our life, for example, take a picture on the road when we come across a dog, and by sending the photo to an APP, it will tell us the breed of the dog.

It's easy to tell if an image is picturing human being or dogs for human being. However, it's always a challenging job for computer. Thanks to the access to big data and the development in machine learning algorithm, image classification is one of the areas where deep learning models are very successful and popular, among the various complicated models, the Convolutional neural networks (CNN) model is a very effective class of neural networks that is highly effective at the task of image classifying, object detection and other computer vison problems.[1] With the development of large data age, (CNN) with more hidden layers have more complex network structure and more powerful feature learning and feature expression abilities than traditional machine learning methods. [2] Compared to standard feedforward neural networks with similarly-sized layers, CNN have much fewer connections and parameters and so they are easier to train, while their theoretically-best performance is likely to be only slightly worse. Despite the attractive qualities of CNN, and despite the relative efficiency of their local architecture, they have still been prohibitively expensive to apply in large scale to high-resolution images. Luckily, current GPUs, paired with a highly-optimized implementation of 2D convolution, are powerful enough to facilitate the training of interestingly-large CNN, and recent datasets such as ImageNet contain enough labeled examples to train such models without severe overfitting.[3]

In practice, very few people train an entire CNN from scratch, because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pre-train a ConvNet on a very large dataset(e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the ConvNet either as an initialization or a fixed feature extractor for the task of interest.[4]

Therefore, in this project, I will not only build a simple classifier model from scratch using CNN but also build another transfer learning CNN model from some pre-trained models in Pytorch, which should be able to attain at least 60% accuracy on the test set.

2. Problem Formulation

The application aims to help people identify dog breeds, it will have two main functions, after feeding it with a image, the application should tell users whether it's a photo of human being or dog, and it will return the breed of the dog.

Among the above three functionalities, identifying dog breeds is exceptionally challenging, the same breed may look different due to dog age, color and subspecies, on the contrary, different breeds may look similar , such as young golden and Labrador Retriever. I will build a CNN breeds classifier from scratch, also build a transform learning model in order to attain a higher accuracy.

This project will be implemented in Jupyter Notebook ***dog_app***, provided by Udacity Machine Learning engineer nanodegree team, which already embedded instructions, template codes, referred materials and test functions, I followed the guidelines, implemented additional functionality and improved the model performance.

3. Data Analysis

3.1 Data Exploration

In this project, the CNN model needs to be trained with a large dataset, which is provided by the Udacity Machine Learning engineer nanodegree team. Since the paths of human image and dog image are downloaded and saved as Numpy array, so I defined two data frames 'Human' as shown in **Table 1** and 'Dog' in **Table 2**, and there were three columns in each data frame, representing the path of image, name or breeds and the shape of image, then it became easier for me to explore the statistics of the images.

	Human_images_path	Name	image_shape
0	/data/lfw/Dan_Ackroyd/Dan_Ackroyd_0001.jpg	Dan_Ackroyd	(250, 250, 3)
1	/data/lfw/Alex_Corretja/Alex_Corretja_0001.jpg	Alex_Corretja	(250, 250, 3)
2	/data/lfw/Daniele_Bergamin/Daniele_Bergamin_00...	Daniele_Bergamin	(250, 250, 3)
3	/data/lfw/Donald_Carty/Donald_Carty_0001.jpg	Donald_Carty	(250, 250, 3)
4	/data/lfw/Barry_Switzer/Barry_Switzer_0001.jpg	Barry_Switzer	(250, 250, 3)

Table 1: Head of Human DataFrame

	Dog_images_path	Breeds	image_shape
0	/data/dog_images/train/103.Mastiff/Mastiff_068...	Mastiff	(648, 800, 3)
1	/data/dog_images/train/103.Mastiff/Mastiff_068...	Mastiff	(307, 300, 3)
2	/data/dog_images/train/103.Mastiff/Mastiff_068...	Mastiff	(433, 250, 3)
3	/data/dog_images/train/103.Mastiff/Mastiff_068...	Mastiff	(2304, 3072, 3)
4	/data/dog_images/train/103.Mastiff/Mastiff_068...	Mastiff	(395, 400, 3)

Table 2: Head of Dog DataFrame

Now by using **describe ()** function in **Pandas** to view the statistics of human and dog images, I can tell the number of images contained in each file and their sizes range. Here, the dataset includes 13,233 human images and 8,351 labeled dog images from 133 breeds, as shown in **Table 3** and **Table 4**, the human being images will be only used for human detector, and they're uniformly sized with 256 by 256 pixels, while the dog images are RBG with size ranging from hundreds by hundreds to thousands by thousands of pixels, which will need to be transformed to uniform size to feed the model.

	Human_images_path	Name	image_shape
count	13233	13233	13233
unique	13233	5749	1
top	/data/lfw/Vincent_Brooks/Vincent_Brooks_0004.jpg	George_W_Bush	(250, 250, 3)
freq	1	530	13233

Table 3: Descriptive Statistics of Human Images

	Dog_images_path	Breeds	image_shape
count	8351	8351	8351
unique	8351	133	4217
top	/data/dog_images/train/051.Chow_chow/Chow_chow...	Alaskan_malamute	(480, 640, 3)
freq	1	96	476

Table 4: Descriptive Statistics of Dog Images

3.2 Data Visualization

In the second column of each DataFrame, it's the name of human or breed of dog, so I used **value_counts ()** function in **Pandas** to count the frequency of all dog breeds. Among the 8,351 dog images, the most common one is Alaskan_malamute with 96 images in the dataset, accounting for 1.15%, and the least common breed is Norwegian_buhund as there is only 33

images with a percentage of 0.40%, the dataset is fairly uniformly distributed, there is no obvious skew, so there is no need to balance the data or define customed evaluation metrics, then I plotted the histogram for the top 30 common dog breeds in **Figure 1**.

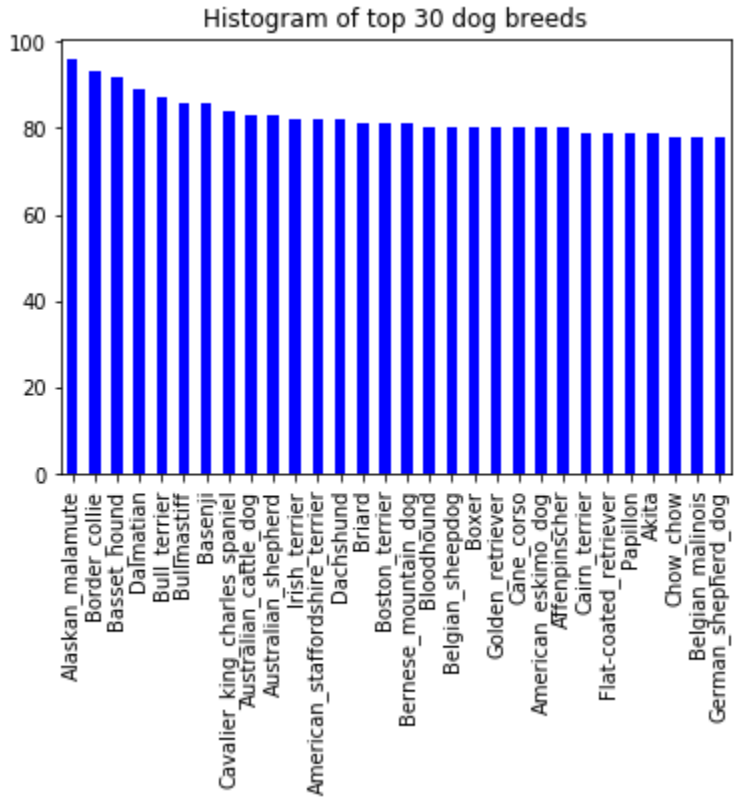

Figure 1: Histogram of Top common 30 Dog Breeds

4. Evaluation Metrics

Since the evaluation will be depend on accuracy score and test loss, I need to address the definition of the evaluation metrics here:

- Accuracy: our model is a classifier, the accuracy will depend on the correct predictions that the model determines, the accuracy score will be calculated as:

$$\text{Accuracy}=\frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

- Loss function: For this image classification problem, I will use Cross-entropy loss function, which measures the performance of classification model whose output is a probability value between 0 and 1, and the loss penalizes both types of errors, but especially those predictions that are confident and wrong. The formula is described by the Pytorch documentation as below:

$$loss(x, class) = -\log\left(\frac{exp(x[class])}{\sum_j exp(x[j])}\right) = -x[class] + log\left(\sum_j exp(x[j])\right)$$

or in the case of weight argument being specified:

$$loss(x, class) = weight[class]\ (-x[class] + log(\sum_{j} exp(x[j])))$$

The loss value will need to be calculated for training set, validation set and test set, by comparing the loss value between each dataset, I can tell whether the model under-fits or over-fits the training dataset, if the training loss is much lower than the validation loss, the model might overfit, on the contrary, if both the validation loss and training loss are high, the model may be under-fitting.

5. Algorithm and Implementation:

5.1 Face Detector:

As mentioned in the problem formulation, in the project, I need to build a human face detector, dog face detector and a dog classifier to make those functions work. The first two detectors will be used for identifying the existence of human faces and dog faces, Udacity team has provided some sample code:

- **Human detector**: Use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.
- **Dog detector**: I will use a pre-trained VGG-16 model, this model will take the path to an image as input and return the index corresponding to the ImageNet different categories, since there are 1000 categories in the ImageNet dataset, the output of this model is expected to be a integer between 0 and 999, inclusive. However, only an uninterrupted sequence includes all categories for dog breeds, so for a dog face detector, we only need to check if the output is in 151 and 268(inclusive).

5.1.1 Algorithm and Data

A pre-trained VGG-16 model will be used for dog detector, which includes 13 convolutional layers and 3 linear layers with weights, and it's considered to be one of the excellent vision model architecture till date. All the pre_trained models from Pytorch expect inputs images normalized in the same way: mini-batches of 3-channel RGB images of shape (3 x H x W), where H and W are expected to be at least 224. The images have to be loaded in to a range of [0, 1] and then normalized using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225], so I will define a data processor function to resize the input images, and the function is expected to return an index through 151 to 269 indicating the dog categories classification.

5.1.2 Test Results

The sample of the output results for Human face detector and dog detector is shown in **Figure 2** and **Figure 3,** by assembling the detectors in the Boolen functions, I can use some human images and dog images to test the accuracy of these two algorithm, the test data set includes 100 images from ***human_files*** and 100 images from ***dog_files***, and the test results are:

For the human face detector:

- The percentage of human face detected in 100 human images is 98.0%.
- The percentage of dog face detected in 100 human images is 17.0%

For the dog detector:

- The percentage of human face detected in 100 dog images is 0.0%
- The percentage of dog detected in 100 images is 100.0%

The accuracy rate for both detectors attained are high enough, but the dog detector has relatively higher prediction accuracy, so in the final application, I will double check the results of these two detectors, if both the human face one and dog one reported False, the final application will raise an error, if both the human detector and dog one predicted True, I will assume the photo is from a dog.

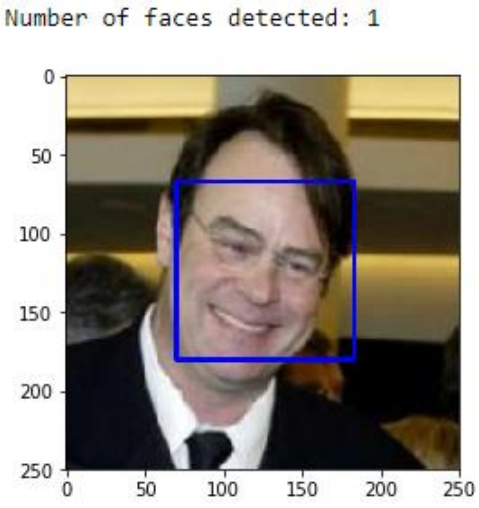

Figure 2: Sample from Human Face Detector

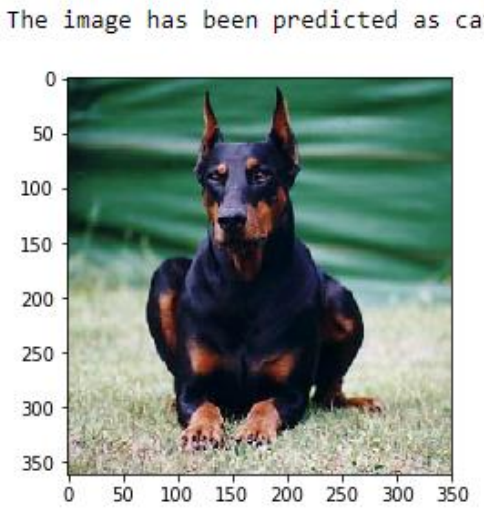

Figure 3: Sample from Dog Detector

Once the detector confirms there is a dog face presented in the input image, the image path will be passed to the breed classifier to identify its breed. Otherwise, the application will return that a human face is detected and then reported a dog breed that mostly resembles the human face or require another image if neither human face nor dog face has been determined from the input image.

Dog breed classifier is the essential function of this project, I will firstly build a CNN model with relatively simple architecture. However, in order to attain a higher prediction accuracy score, I will build another classifier that transformed from pre-trained models, such as AlexNet, ResNet50, ResNet101 and VGG16, the final model will be determined by the performance of each candidate model, the evaluation will include test accuracy score and test loss value.

5.2 Breeds Classifier

As mentioned above, there are two classification models in this project, the simple-structure CNN model will be developed from scratch, and the accuracy score is expected be over 10% as it's very difficult to make enough right predictions with a simple architecture model, and the random guess will provide a correct answer with a accuracy of less than 1%.

For the transfer learning model, the accuracy is expected to be over 60% as the model will be built from much more complicated pre-trained model.

5.2.1 Data preparation

The dog dataset will be split into training set with 6,680 images, validation set with 835 images and 836 images in the test set. As I planned to use the same data loader for the simple CNN model and transfer learning model, as I mentioned in **Section 5.1.1**, for all pre-trained models in PyTorch, the dataset should follow special standard format, so I firstly resized all the image to 256 squares, then I cropped the image to 224 squares, to augment the dataset, random horizontal flip and random rotation were used here, then I then normalized using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225]. Afterwards, training set, validation set and test set will be saved in a whole data set, and the breeds name comes from the class of train set. Here I set batch size equal to 32, then by using **DataLoader ()** function, the data was prepared.

5.2.2 Model architecture

- **Simple CNN**: I will start to build the model with a simple 3 convolution layers and a linear layer that sample code provides, then I will try different architecture and hyper-parameters depending on the validation and test performance that evaluated by the metrics.
- **Transfer Learning CNN**: As I mentioned in **Section 5.1**, the candidate transfer learning models will include AlexNet, ResNet50, ResNet18, ResNet34, VGG16 and ResNet101, [5] the benchmark accuracy for a not bad classier is at least 60%. Since the pre-trained models have extracted lots of the low level features, I will simply train the model with different parameters in their last layer, so we can reduce lots of training efforts and still attain a good performance. Because we will have already a very huge amount of parameters for all the candidate pre-trained models, I chose to 'freeze' those layers except for the last block so those weights won't change. And it save time and computation cost.

5.2.3 Refinement

This process mainly refers to model selection and tuning the model, I tried different combinations of hyperparameters and pre-trained models from PyTorch, and I will choose the best performer according to the evaluation metrics. I started from ResNet50, because I've read a blog about the comparisons of different image classifier models, and I remembered ResNet50 not only has a low prediction error but also costs a reasonable training time, with a learning rate of 0.01, 15 epochs, the loss didn't converge as quickly as expected, and the value was never lower than 3, I got a test accuracy as 72%.However, both the training loss and validation loss were still decreasing, and the validation loss was still a bit higher than the training loss, indicating that I

could still improve the model performance by increasing the epochs or changing the learning rate.

In order to find the most efficient models, I decided to try other candidate model with the same epochs and learning rate first, so I implemented transfer learning on ResNet101,AlexNet and VGG16, for ResNet101, both the performance was worse, the loss function converges with a lower speed. As to the AlexNet, the training loss went down at a much higher speed than the validation one, indicating the model might be overfitting. Last but not the Least, VGG16 generated a test accuracy 73%, just 1% higher than ResNet50.

To attain a higher accuracy score, I finally chose ResNet50, and increase the epochs to 30 (it's 10 in the Jupyer notebook, as I trained the model twice, the initial weights had been adjusted), meanwhile, to avoid the overfitting issue, I set the learning rate equal to 0.001. This time, the training and validation loss declined relatively with the same speed, and the trained model yielded a much lower validation loss of 1.12 after 20 epochs. Therefore, I got my final CNN architecture.

6. Results

6.1 Model Evaluation and Validation

By running the test functions, I got a final 78% (654/836) prediction accuracy rate and test loss 1.005, I felt satisfied with the model performance as I mentioned classifying dog breed is exceptionally challenging task.

Now I have two detectors and one classifier, so I assemble these three functions in the final application ***run_app***, the function will take the image path as inputs, and in the body, it will firstly identify the image is from human or dog, then returns the mostly possible breed. Two prediction sample are shown below in Figure 4 and Figure 5, both images come from local directory so they are out of sample tests, the ***run_app*** application performs well.

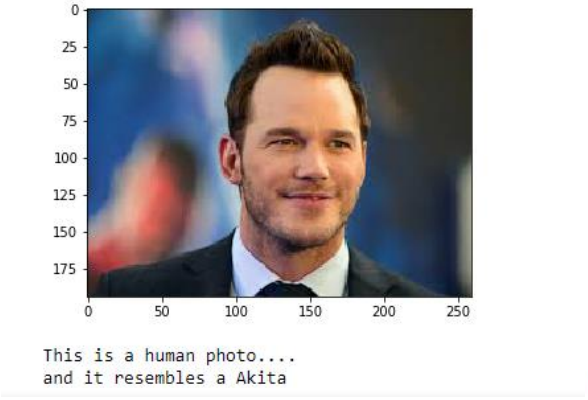

Figure 4: Test Result for Human Image

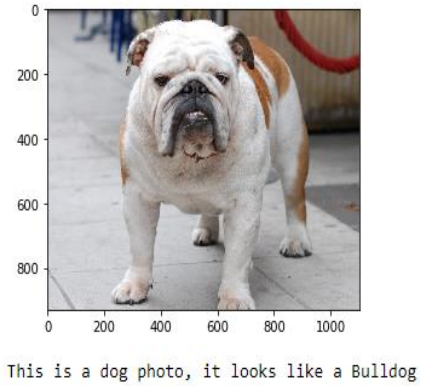

Figure 5: Test Result for Dog Image

6.2 Justification

My simple architecture CNN model gets a test loss of 3.84 after 15 epochs, and test accuracy equals to 13% (117/836), while the transfer learning ResNet50 model has a test loss as 1.00 and test accuracy 78% (654/836). Obviously, the transfer learning ResNet50 model outperformed the simple model. Moreover, from the aspect of training time that each model took, we can save lot of training time as we start using the weights of a previously trained model.

7. Conclusions

7.1 Reflection

This project begins with problem formulation, where I stated that two detectors and one classifier should be well built. Following the instruction from Udacity team, I built a human face and a dog detector basing on pre-trained OpenCV's and VGG-16 Pytorch model. Once I got a good result from these two detectors, the next step would be building a classifier. Since a classification CNN model needs a large dataset and proper data processing before training, so I built separated data loaders for training set, validation set and test set, I resized all images into a standard format. Moreover, in order to augment the dataset, I also randomly flipped and rotated the images. Then, the next step is to build, train and test a simple architecture CNN model for the dog breed classification. Once the benchmark met the 10% accuracy benchmark, I tried several pre-trained PyTorch models to build, train and validate a transfer learning model. Finally, I picked the best validation performer ResNet50 as my final algorithm and assembled two detectors and this best classifier into ***run_app*** function, and tested its performance using local images.

7.2 Possible Improvements

- It’s always better to have a larger training dataset, especially when the final algorithm has difficulty in identifying similar breeds, by increasing the training and validation set, the algorithm could be considerably improved simply because it can detect more subtle differences**.**
- Instead of only changing the last block of ResNet50, we can add layers to achieve a higher accuracy.
- To find a balance between bias, overfit and computation cost, we can try different optimizer (SGD, Adam, RMSprop), lower learning rate and epochs to get a successfully trained model.

References

1. M. T. Islam, B. M. N. Karim Siddique, S. Rahman and T. Jabid, "Image Recognition with Deep Learning," *2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, Bangkok, 2018, pp. 106-110, doi: 10.1109/ICIIBMS.2018.8550021.

2. A. M. Al-Saffar, H. Tao and M. A. Talab, "Review of deep convolution neural network in image classification," *2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)*, Jakarta, 2017, pp. 26-31, doi: 10.1109/ICRAMET.2017.8253139.

3. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM, 60*(6), 84-90. doi: 10.1145 / 3065386
4. Chilamkurthy, S. (2017). Transfer Learning Tutorial. In PyTorch. Retrieved from https:// pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

5. Talo, Muhammed & yıldırım, Özal & Baloglu, Ulas & Aydin, Galip & Acharya, U Rajendra. (2019). Convolutional neural networks for multi-class brain disease detection using MRI images. Computerized Medical Imaging and Graphics. 78. 101673. 10.1016/j.compmedimag.2019.101673.