Machine Learning Engineer Nanodegree

Capstone Project: Dog Bleed Classifier (CNN)

James Lau
September 27th, 2021

I. Definition

Project Overview

Convolutional neural networks (CNN) have broken the mold and ascended the throne to become the state-of-the-art computer vision technique. Among the different types of neural networks (others include recurrent neural networks (RNN), long short-term memory (LSTM), artificial neural networks (ANN), etc.), CNNs are easily the most popular. These convolutional neural network models are ubiquitous in the image data space. They work phenomenally well on computer vision tasks like **image classification**, object detection, image recognition, etc. [1]

A CNN consists of an input layer, output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers, and normalization layers (ReLU). Additional layers can be used for more complex models. This architecture has shown excellent performance in many Computer Vision and Machine Learning problems.

This project will apply the Convolutional Neural Networks (CNN) to predict dog images downloaded from https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip. There are totally 8351 images with 133 dog categories.

We will try various pre-trained models such as VGG16, ResNet50 and InceptionV3 to predict whether an image is a dog first, then develop a model from scratch to classify dog breeds, and finally use the technique of Transfer Learning to create a CNN that can identify dog breed from images.

For developing a model from scratch to classify dog breeds, we will also try different techniques in machine learning such as Batch Normalization (BN) and Global Average Pool (GAP) to see how it helps to improve the performance of the base model.

Problem Statement

In this project, we will apply Convolutional neural networks (CNN) to classify the bleed of a dog based on a given image. If the image is neither a human nor a dog, the output will tell us it's not a human or dog. Given an image of a dog, the algorithm will identify an estimate of the canine's bleed. If supplied an image of a human, it will identify the resembling dog breed. We will also use different models for benchmarking to see their strengths and weaknesses.

After completing this project, I can learn how to build a pipeline to process a real-world problem; from data processing, model design, improvements (use transfer learning) to the final model.

Metrics

Accuracy is the metrics we will use to measure the performance of a model. It is the measure of how accurate the model's prediction is compared to the true data. For our problem, it's simply the rate of correct classification.

The definition of Accuracy is as follows:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

TP – number of true positives

TN – number of true negatives

FP – number of false positives

FN – number of false negatives

Why I choose Accuracy as the metrics:

- Accuracy is a good pick when the dataset is well balanced. From the statistics of the frequencies of dataset classes.
  Max = 77, Min = 26, Mean = 50.23, Median =50

  We can see the dataset is balanced without extreme maximum or minimum, and not skewed a lot as well, so I think accuracy is appropriate to use as a metric in this project.

II. Analysis

(approx. 2-4 pages)

Data Exploration

Exploratory Visualization

We imported the dog datasets from https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip.

There are 3 datasets, the training, validation, and test set with the following statistics.

Training set – 6680 dog images

Validation set – 835 dog images

Test set – 836 dog images

Out of the 8351 images, there are 133 dog bleeds (The distribution is shown in the diagram in the following page).

➢ The dimensions of the input images are consistent, all resized to 224 and normalized using mean= [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225] before fit into the model, and in RGB mode

➢ Statistics on the frequencies of each dataset classes

Max: 77

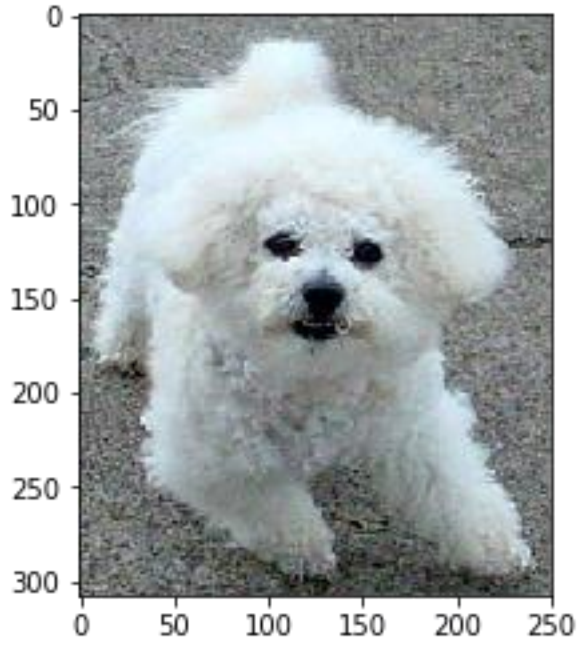
Min: 26

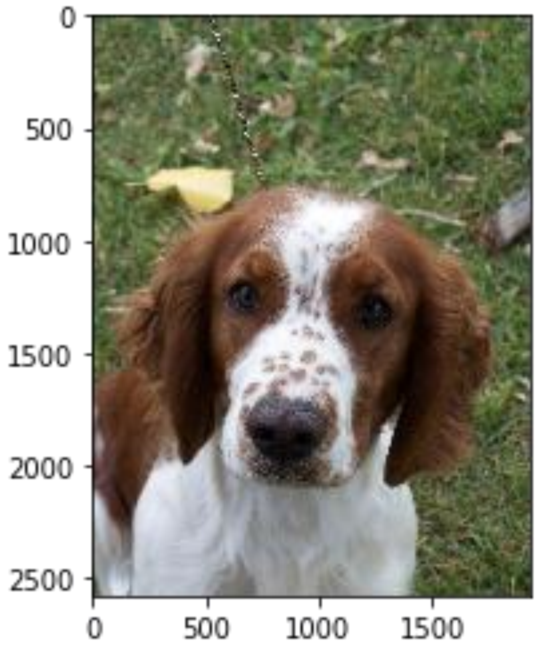
Mean: 50.23

Median: 50

From the above statistics, we can see that the class distribution is quite balanced, and no extremely low or extremely high value in the distribution.

- The followings are some sample images of the dogs

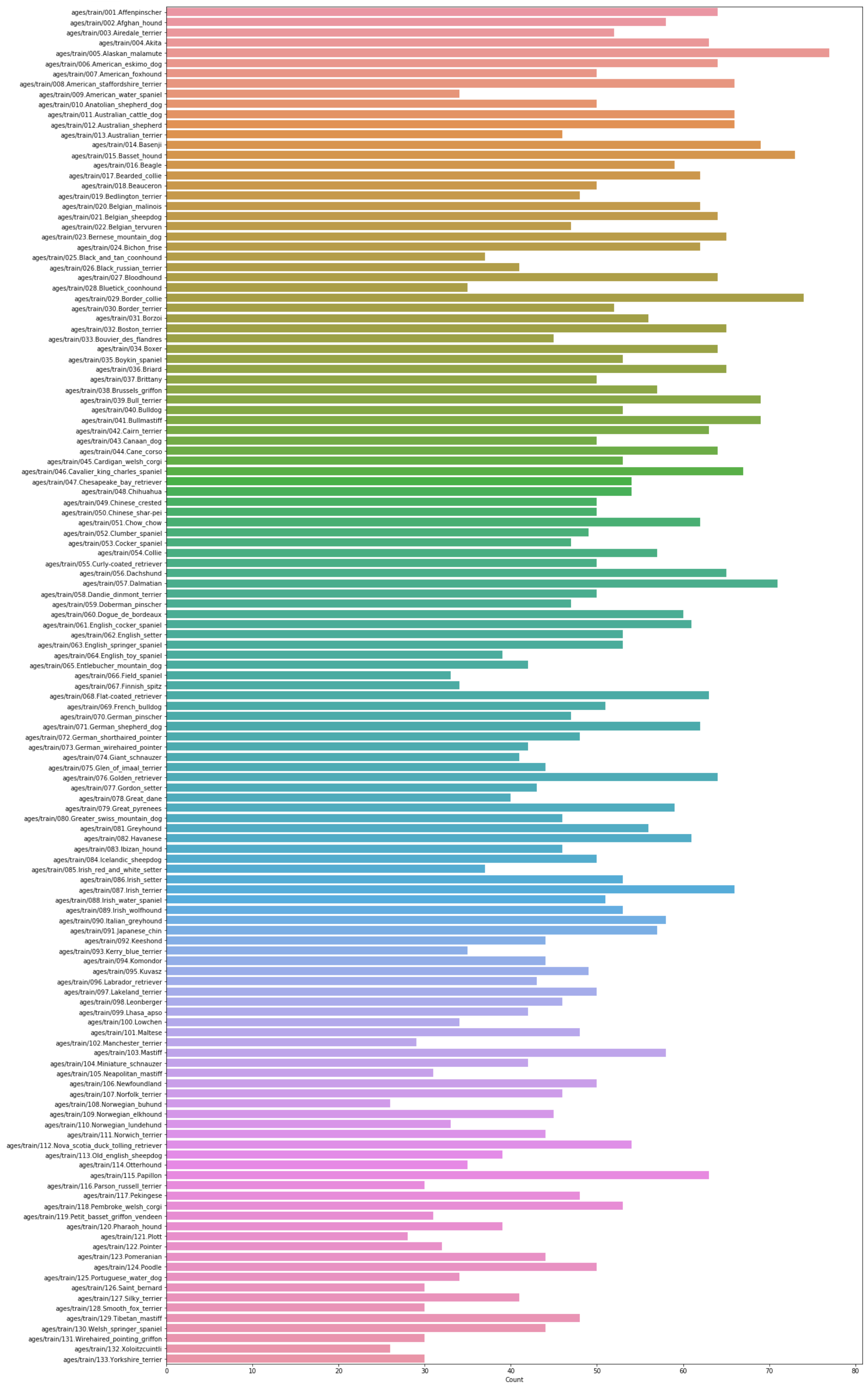




The number of dog images in training set (6680 out of 8351) may not be enough for training model in the later part of the project (create a model from scratch). Since the size of training set affected the accuracy of the prediction of our model, we will use another other technique called augmentation to increase the training set size. We will explain more about data augmentation in later section.

Dog Bleed

Dog Bleed vs. Count



Algorithms and Techniques

Algorithms:

1. Import datasets
- ➢ The data is downloaded from the site provided by Udacity.
- ➢ I resize the data images to 224 and normalized using mean= [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225] before fit into the pre-trained model

2. Detect Humans
   - ➢ We use OpenCV's implementation of Haar feature-based cascade classifiers to detect faces in images

3. Detect Dogs
   - ➢ Use a Pre-trained VGG-16 Model to detect dogs.

4. Create a CNN to Classify Dog Breeds (from Scratch)
   - ➢ Since the size of data samples is not big, I use the technique of Data Augmentation to create new data samples for the training set.
   - ➢ Create a model from scratch
     - Use a simple VGG structure as base
     - Add batch normalization (BN) layer for refinement/benchmark
     - Add Global Average Pool (GAP) Layer for refinement/benchmark
   - ➢ Train, Validate and test model

5. Create a CNN using Transfer Learning to Classify dogs
   - ➢ Use ResNet50 for model transfer
   - ➢ Use VGG16 for model transfer as Benchmark
   - ➢ Use InceptionV3 as Benchmark
   - ➢ Train, Validate and test model for the 3 models

Techniques

I use the technique of **data Augmentation** to improve the performance of creating a CNN model to classify dog breeds (from Scratch).

Why Data Augmentation?
To train a machine learning model, you need plenty of data examples. Altogether, these examples form a training dataset, that enables your machine learning model to learn. The collected dataset should be representative enough and should contain all possible cases and objectives you want the AI model to understand. These requirements often represent a key challenge**. Data Augmentation** generate different versions of a real dataset artificially to increase its size, it helps to improve the performance and outcomes of machine learning models, especially when the original size of the dataset is small.

Since the size of our training set is not big(6680 dog images), by manipulating the original data to create new data samples, I hope overfitting can be avoided, diversity in data can be increased, and the performance of the model will get improved. I transformed the original dataset by resizing, center cropping, random horizontal flip, random rotation and set RandomGrayScale(p=0.1).

Another technique I used is transfer learning.

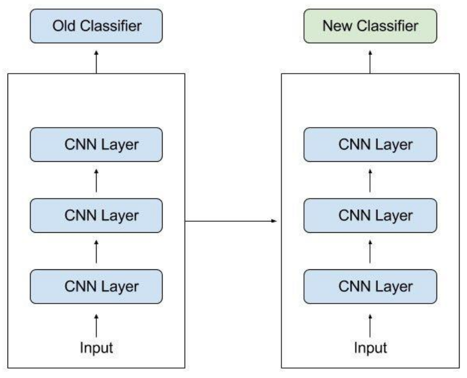
In transfer learning, the knowledge of an already trained model is applied to a different but related problem. For example, if you trained a simple classifier to predict whether an image contains a bag, you could use the knowledge that the model gained during its training to recognize other objects like glasses.

With transfer learning, we basically try to exploit what has been learned in one task to improve generalization in another. We transfer the weights that a network has learned at "task A" to a new "task B." The general idea is to use the knowledge a model has learned from a task with a lot of available labeled training data in a new task that doesn't have much data. Instead of starting the learning process from scratch, we start with patterns learned from solving a related task. Transfer learning is mostly used in computer vision and natural language processing tasks like sentiment analysis due to the huge amount of computational power required.

<u>How Transfer Learning works?</u>

In computer vision, for example, neural networks usually try to detect edges in the earlier layers, shapes in the middle layer and some task-specific features in the later layers. In transfer learning, the early and middle layers are used, and we only retrain the latter layers. It helps leverage the labeled data of the task it was initially trained on.

Let’s take ResNet50 model as an example, which will be used to identify dogs. In the earlier layers, the model has learned to recognize objects, because of that we will only retrain the latter layers so it will learn what separates dogs from other objects.



In transfer learning, we try to transfer as much knowledge as possible from the previous task the model was trained on to the new task at hand. This knowledge can be in various forms depending on the problem and the data. For example, it could be how models are composed, which allows us to identify novel objects more easily.

Weakness of transfer learning:

One of the biggest weakness of transfer learning is the problem of negative transfer. Transfer learning only works if the initial and target problems are similar enough for the first round of training to be relevant. We can draw reasonable conclusions about what type of training counts as similar enough to the target, but the reality doesn't have to agree. If the first round of training is too far off the mark, the model may perform worse than if it had never been trained at all.

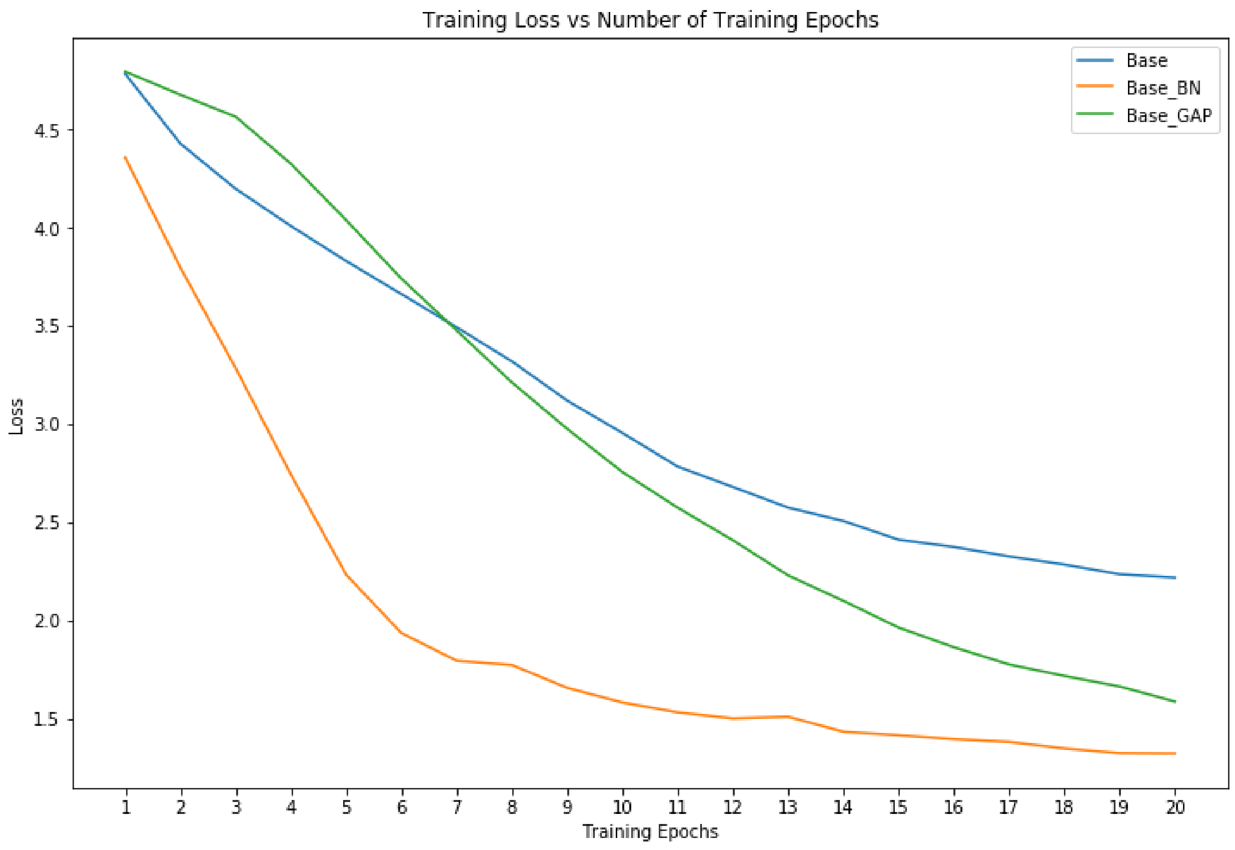
Benchmark

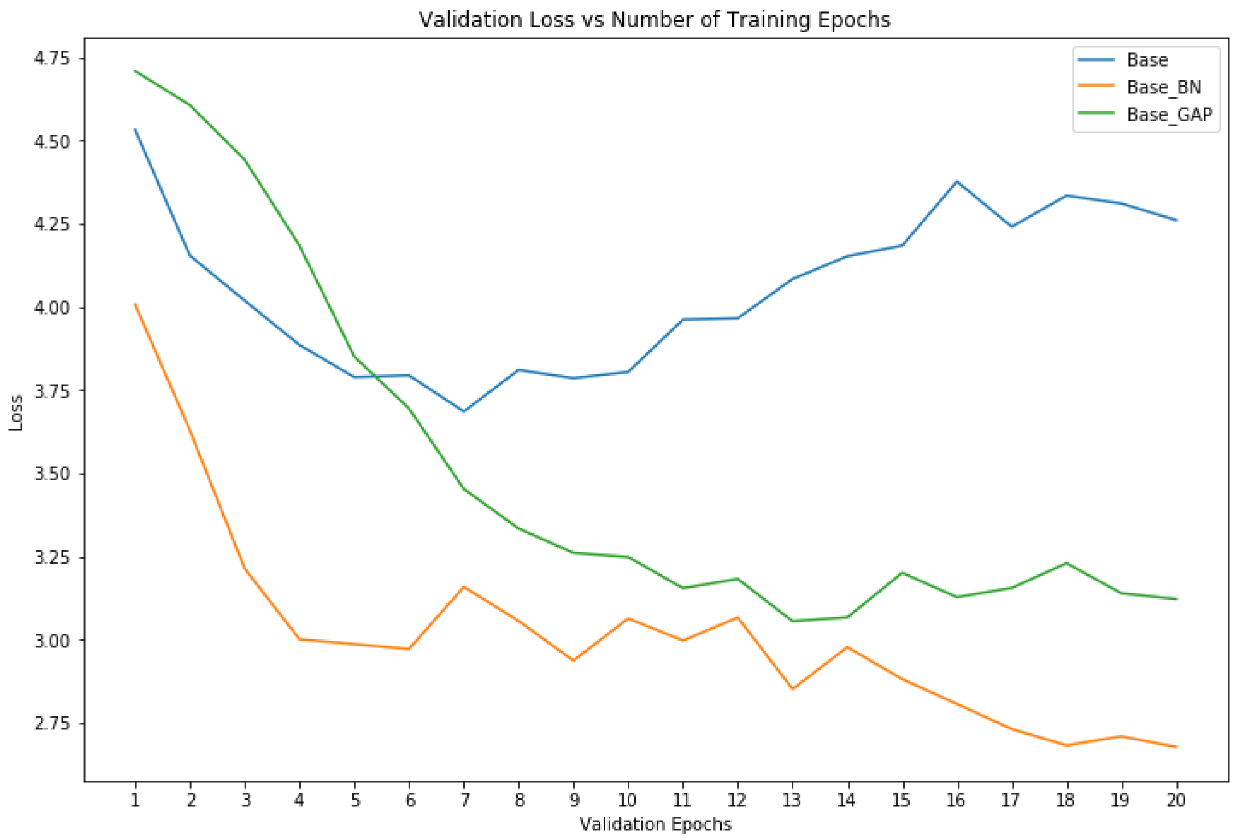
Benchmark is used in two sections of the project:

1. Create a CNN to Classify Dog Breeds (from Scratch)
   - I am trying to achieve 10% accuracy with this "from scratch" model
   - I use a simple VGG structure as a BASE, and then compare it with another 2 models, they are
     - BASE + Batch Normalization (BN) layer
     - BASE + Global Average Pooling (GAP) layer

Results:

Model	# of Epoch	Accuracy
Base(Simple VGG architecture)	20	15.55%
Base + BN(Batch Normalization)	**20**	**37.32%**
Base + GAP(Global Average Pool)	20	29.43%


Training Loss vs Number of Training Epochs
Base
Base_BN
Base_GAP
Loss
Training Epochs
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20


Validation Loss vs Number of Training Epochs
Base
Base_BN
Base_GAP
Loss
Validation Epochs
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

2. Create a CNN to Classify Dog Breeds (using Transfer Learning)
    - I am trying to achieve 60% accuracy with this transfer learning model
    - 3 Model Architecture are used for Comparison
        - ResNet50
        - VGG16
        - InceptionV3

Results:

Model	# of Epoch	Accuracy
ResNet50	10	82.78%
VGG16	**10**	**88.28%**
Inception V3	10	63.28%

III. Methodology

(approx. 3-5 pages)

1. Create a CNN to Classify Dog Breeds (from Scratch)

A simple VGG-block has one or more convolutional layers with the same number of filters and a filter size of 3×3, a stride of 1, same padding so the output size is the same as the input size for each filter, and the use of a rectified linear activation function. These layers are then followed by a max pooling layer with a size of 2×2 and a stride of the same dimensions.

I tried to adopt a simple VGG architecture as a start and serve as a Base.

Base

1. It has alternated conv, pool layers (conv1 - conv5, pool1 - pool5)
2. fully connected layers (fc1, fc2) are added at the end
3. A dropout layer is also added at the very end.
4. I trained it with 20 epochs at a learning rate of 0.001
5. test the performance of the model

Base_BN (use the technique of Batch Normalization)

Batch normalization is used because it standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

1. Add a Batch Normalization layer between the conv and pool layer (bn1 - bn5)
2. add another Batch Normalization layer(bn6) between the fully connected layers(fc1 and fc2) as well
3. drop the dropout layer
4. I trained it with 20 epochs at a learning rate of 0.001
5. test the performance of the model

Base_GAP(use the technique of adding Global Average Pool Layer)

1. Add a Global Average pool layer before the fully connected layer and keep only one fc layer
2. remove the dropout layer
3. I trained it with 20 epochs at a learning rate of 0.001
4. test the performance of the model

Global Average Pool Layer calculates the average output for each feature map from previous layer, that reduces the data significantly and prepares the model for the final classification layer, thus speeding up the training of the model.

From the test accuracy of the 3 models, the architecture with the Batch normalization(Base_BN) got the best results(37%). Base got 15%, the architecture with Global Average Pool(Base_GAP) got 28%.

So, the final architecture that won is the architecture with Batch Normalization.

2. Create a CNN to Classify Dog Breeds (using Transfer Learning)

   Transfer Learning is a method where a model built for a task is reused as a starting point for a model on another task.

   1. Here I use the model ResNet50 as a starting point to solve my problem.
   2. I will keep the common inner layers and customize the final layer(the classifier part).
   3. The classifier part is a single fully connected layer. I replace the classifier with 133 classes that accomodated to my current problem. model_transfer.fc = nn.Linear(2048, 133, bias=True)

   After that, I use VGG16 and InceptionV3 as the model for transfer learning.

   The accuracy of the models are as follows:
   ResNet50 – 83%
   VGG16 – 88%
   InceptionV3 – 63%

Data Preprocessing

1. For Pre-trained model

   Before an image feed into a pre-trained model (e.g. VGG16, ResNet50, InceptionV3), it is converted to a normalized tensor.

   The way it converts follows the documentation from Pytorch as follows:

   "All pre-trained models expect input images normalized in the same way, i.e. 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"

   torchvision.models — Torchvision 0.10.0 documentation (pytorch.org)

2. For creating a CNN from Scratch

   I transformed the original dataset by resizing(224), center cropping, random horizontal flip(p=0.5), random rotation(10), RandomGrayScale(p=0.1) and normalized it using mean=[0.485, 0.456, 0.406]. It was then added to the original training dataset for data augmentation to create the final training set.

Implementation

Metrics:

Training loss → calculated on training, measure how well the model is doing on the training set

Validation loss → calculated on validation, measure how well the model is doing on the validation set

If Validation Loss > Training Loss → it's a overfitting

If Validation Loss < Training Loss → it's a underfitting

Accuracy = Number of Correct predictions / Total number of predictions made

Algorithm:

1. Create a CNN model
2. Train the model with 5 epochs as an increment
3. Test the performance of the model
4. Repeat Step 2 to Step 4 to see if the performance is good enough.

   Accuracy > 10 % for creating a CNN from Scratch and Accuracy > 60% for creating a CNN using Transfer Training.

5. If Step 4 fail, Go to Step 1 to create another CNN model
   For CNN from Scratch, I tried a BASE VGG model, then VGG with Batch Normalization (BN) layer and lastly VGG with Global Average Pool Layer.

   For CNN using Transfer Learning, I tried, RedNet50, VGG16 and InceptionV3

   Techniques:
   Augmentation
   Data Augmentation generate different versions of a real dataset artificially to increase its size, it helps to improve the performance and outcomes of machine learning models, especially when

the original size of the dataset is small. Since out input dataset is small, Augmentation is a good technique to use to create a larger training dataset.

Refinement

In the Transfer learning section, I tried to change the parameter to see if it could make some improvements in the accuracy of the model. I picked the learning rate as the parameter to adjust. The initial value of learning rate is 0.001. I tried 0.0005 and 0.004 so that the initial value is in between.

Results:

Learning rate	Accuracy
0.001	82.78%
0.004	79.9%
0.0005	**86.48%**

From the results, I can see decreasing the learning rate will help to improve the accuracy of the model and increasing it will make the performance worse.

The result does make sense, a learning rate that's too high will result in the network weights being changed by far too much each update step, and so they end up jumping past their optimal value, with small learning rate the network will start to slowly converge which results in loss values getting lower and lower.

By lowering the learning rate to 0.0005, we get a higher accuracy of 86.46%, 3.7% more.

IV. Results

(approx. 2-3 pages)

Model Evaluation, Validation and Justification

I) Creating a CNN Model (from scratch)

Accuracy of different Architecture:

Model	# of Epoch	Accuracy
Base(Simple VGG architecture)	20	15.55%

Base + BN(Batch Normalization)	20	37.32%
Base + GAP(Global Average Pool)	20	29.43%

From the accuracy shown in above table, it is very clear that the model of **Base+BN** outperforms the **model Base** and **Base+GAP** Model.

Batch normalization(BN) is a layer that allows every layer of the network to do learning more independently. It is used to normalize the output of the previous layers. Using batch normalization learning becomes efficient also it can be used as regularization to avoid overfitting of the model.

Global Average Pooling(GAP) layers are used to reduce the spatial dimensions of a three-dimensional tensor. However, GAP layers perform a more extreme type of dimensionality reduction, where a tensor with dimensions h×w×d is reduced in size to have dimensions 1×1×d. GAP layers reduce each h×w feature map to a single number by simply taking the average of all hw values.

Probably due to this reason, the accuracy of the **Base+GAP** model is less than the accuracy of the **Base+BN** model in this study.

II) Transfer Learning

Chart A: Accuracy of various Model

Model	# of Epoch	Accuracy
ResNet50	10	82.78%
VGG16	**10**	**88.28%**
Inception V3	10	63.28%

From Chart A, VGG16 is proven to be the winner among the 3 models.

Chart B: Accuracy of various learning rate of the model VGG16

Learning Rate	# of Epoch	Accuracy
0.0005	10	86.65%
0.0008	10	87.68%
0.001	**10**	**88.28%**
0.005	10	87.7%

In the Refinement study in last section, I learn that by adjusting the **learning rate**, the accuracy of a model can get improved. I tried 4 learning rates: **0.001**(initial value), 0.0005, 0.0008, 0.005.

From chart B, we can see the accuracy is still the highest for the initial learning rate, that means 0.001 probably already the optimal learning rate I should use.

Actually different Optimizer (e.g. Adam, SGD, .. etc.) behave differently, the optimal learning rate for an optimizer may not be the same as the other optimizer. So, we need to do some tests to get the optimal learning rate.

Chart C: Accuracy for Augmented and Non-Augmented Dataset

Augmented?	Model	# of Epoch	Accuracy
Yes	ResNet50	10	84%
No	ResNet50	10	83%

From Chart C, we see the performance of the model ResNet50 for Augmented dataset is not very significant, yet the training time is longer. One of the reasons may be of the fact that the size of the original training set is not big (6680 dog images), even after it is augmented, still not representative enough.

We use both pre-trained models (VGG16, ResNet50, InceptionV3) with transfer learning as well as model build from scratch. From the accuracy shown in various charts provided above, obviously using transfer learning is a better solution. It's easy to understand. The pre-trained models have been trained on ImageNet, a very large, very popular dataset used for image classification and other vision tasks. ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories.

V. Conclusion

(approx. 1-2 pages)

Convolutional Neural Network (CNN) is a very useful architecture for image processing, classification, segmentation and for other auto correlated data. In this project, I used various pre-trained model such as VGG16, ResNet50, Inceptive V3 to do prediction on dog/human images, and I also developed different models from scratch which are based on a simple VGG structure(Base), and enhanced with different techniques (Batch Normalization (BN) and Global Average Pooling (GAP), By comparing the accuracy got from these models, I concluded The Base model with Batch Normalization(Base-BN) [Accuracy = 37.32%] outperformed the BASE-GAP model [Accuracy = 29.43%] and BASE model [Accuracy = 15.55%]. And they are all exceeded the requirement of accuracy > 10%.

Besides creating CNN from scratch, I also use transfer learning on pre-trained model (VGG16, ResNet50, InceptionV3) and able to achieve very good results; VGG16 with accuracy of 88.28%, ResNet50 with accuracy of 82.78% and Inceptive V3 with accuracy of 63.28%. Again, they are all exceeded the requirement of greater than 60%.

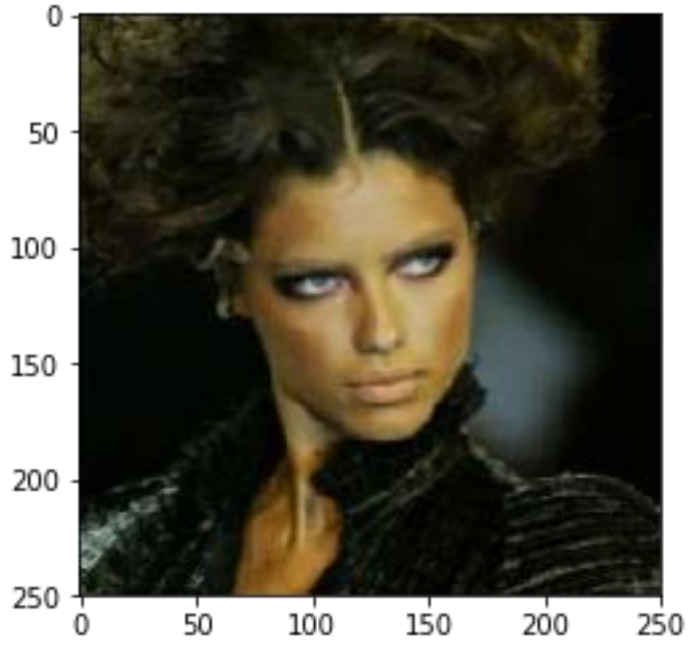
I have used the models that's been developed to predict the dog bleeds on human, dog and nature images, it works very well and make the predictions correctly.

Various techniques have been used to improve the results such as using augmented dataset, adjusting parameter, etc. There's still a lot of rooms to improve the performance.

Free-Form Visualization

The algorithm detects correctly on human, dog and nature images:

```
Human is detected : resembing dog breed is Poodle
```



```
Dog is detected: dog breed is Bichon frise
```

```
Neither Dog Nor Human
```

Reflection

The Udacity Machine Learning Degree opened my eyes in the vast area of Machine Learning, and I learned a lot during the course, from some basic topics such as Python, software Engineering fundamentals to various machine learning studies, different algorithms and techniques open the door to this exciting field for me.

In this capstone project, I can have hand-on experience to dig in the details of various CNN architecture, struggled to make the training of different models work and make the models to reach the goals. I learned the VGG16, ResNet50, Inceptive V3 models, what's augmentation, training loss, valid loss, accuracy, batch normalization, Global Average Pooling, and various layers in an architecture, etc.

After all these struggles, I had an overall picture of how these CNN architectures work and how powerful they are. During the research process of this project, I found there are a lot of applications of these models in the medical field.

There is always one concern when I worked on this project, the training time is always long. Since Udacity have some limitation on the GPU time that I can use, it's very easy exhausted the given time, so I cannot train my models with a lot of epochs. The time was a challenge since I have a full-time job as well. But overall, I think the final model and solution fit my expectations for the problem.

I am glad I have taken this course and I think I learned a lot from it.

Improvement

There are a few ways that I think the implementation can get improved:

1. For the CNN model from scratch, I can apply both the technique of Batch Normalization (BN) and Global Average Pooling (GAP) at the same time on the BASE model.
2. Use more epochs to get better performance.
3. Explore more settings in the Optimizer to get the optimal performance

4. Increase the size of the training dataset so that it will cover more categories of dogs and have more images for each category.
