Capstone Project: Machine Learning Nanodegree

Juan Arroyo-Miranda, 11.03.2020

Project Overview

The last decade has seen a steady rise in both the number and variety of of deep learning applications. From computer vision and image processing to speech detection, deep learning approaches have outperformed traditional machine learning algorithms. [1] Particularly relevant to this project are the developments related to the field of computer vision and object recognition. It wouldn't be an exageration to say that applications related to object detection and analysis have become part of our everyday life. Examples of this can be seen either in traffic detection models that capture environmental data from intelligent sensors for analysis and forecast [2] or in applications that use facial recognition software for identity or payment verification[3].

Building upon Udacity's project proposal, I created a machine learning pipeline that is able to predict a canine breed from user-supplied images. The classifier accepts images of both dogs and human beings and provides a prediction of the canine breed for the dog or the closest resembling dog breed in case of receiving a human image. To accomplish this task, the classifier relies on two pre-trained networks, one to detect humans and one two detect dogs, as well as on Convolutional Neural Network (CNN) that uses transfer learning to classify dog breeds.

Problem Statement

As mentioned before, the goal of this project is to build a machine learning pipeline that process and classifies user-supplied images into different canine breeds. In order to achieve this, the following tasks need to be completed:

1. Download the training datasets cointaining both human and dog images (See Data section for a detailed description).
2. Use a pre-trained face detector from **OpenCV** to create a function that detects images that belong to a human being.
3. Use the **VGG-16** pre-trained model to write a function that detects images of dogs.
4. Create a custom **CNN** that classifies images into different canine breeds.
   - Create a set of functions to load and transform the raw images for training and validation
   - Define the network architecture.
   - Specify Loss function and Optimizer
   - Perform model training and validation steps
   - Test model's accuracy
5. Use transfer learning to create a **CNN** that can identify dog breed from images.
   - Create a set of functions to load and transform the raw images for training and validation

    - Define the network architecture.
    - Specify Loss function and Optimizer
    - Perform model training and validation steps
    - Test model's accuracy
6. Write a custom algorithm that runs the model prediction in the following way:
    - 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** is detected in the image, provide output that indicates an error.

The first two components allow the classifier to quickly identify images of dogs and human from other animals or objects passed to the classifier. Once the correct images are identified, the transfer learning CNN will provide a prediction of the canine breed taking advantage of the information that the pre-trained model obtained in previous learning rounds.

Data Preparation and Methodology

This project relies on two different datasets, the dog dataset and the human dataset, which have been supplied by Udacity's team. There are a total of 8,351 images in the dog dataset and 13,233 images in the human dataset. Given these inputs, the final classifier will predict the canine breed for a given dog image or the resembling dog breed for a human image.

Figure 1 presents the distribution of class frequencies for a random sample of dog breeds in the training dataset. The random sample represents approximately 1/3 of the classes available for training. At first sight, it appears that some classes have a higher representation than others and we may be dealing with class imbalance problem. However, looking a the complete distribution of class frequecies (**Figure 2**) in the training dataset we observe that the average class represents ~0.75% of the images in the dataset and 95% of the clases have frequencies between 0.39% and 1.01%. The latter suggests that classes in the training set are, on average, balanced with a frequency close to the mean (which almost coincides with the median) of the distribution.

Figure 1. Sample Distribution for Class Frequencies in Training Dataset

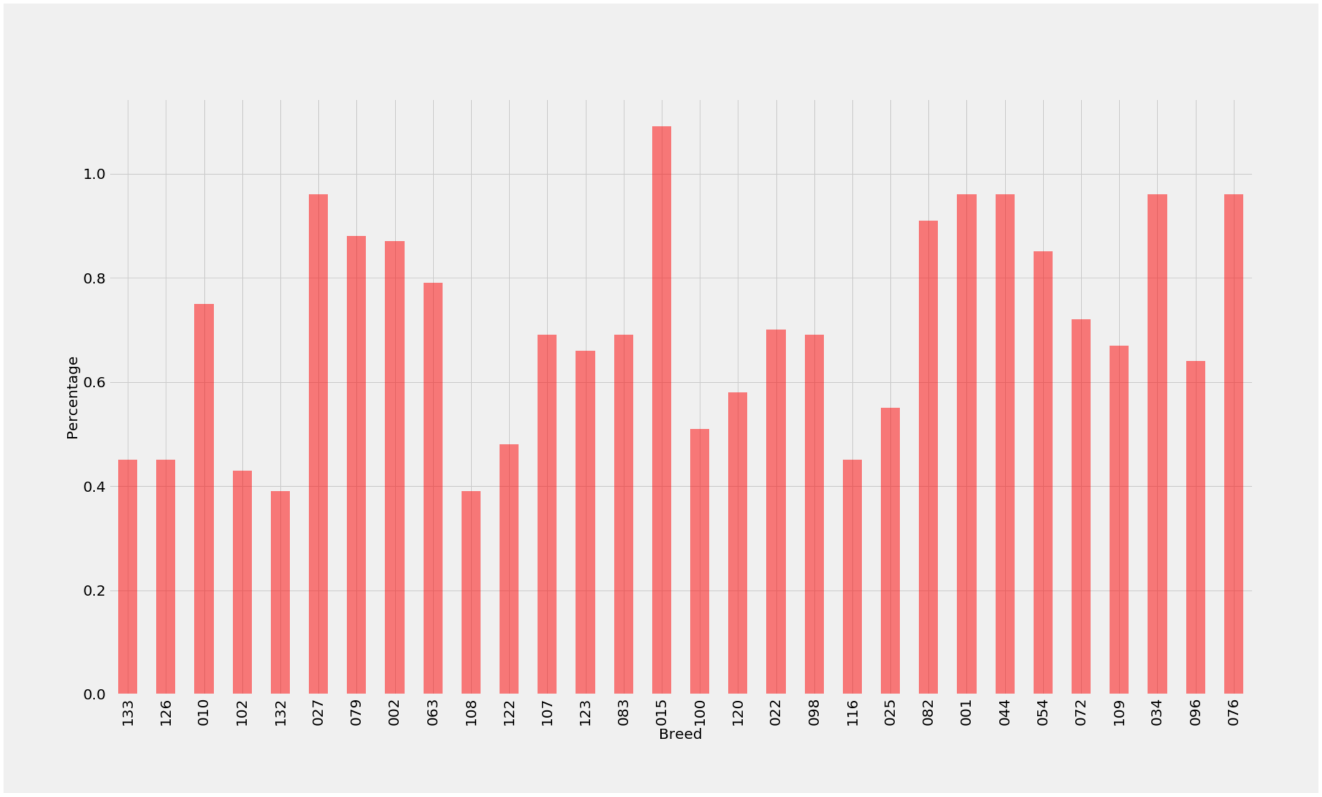
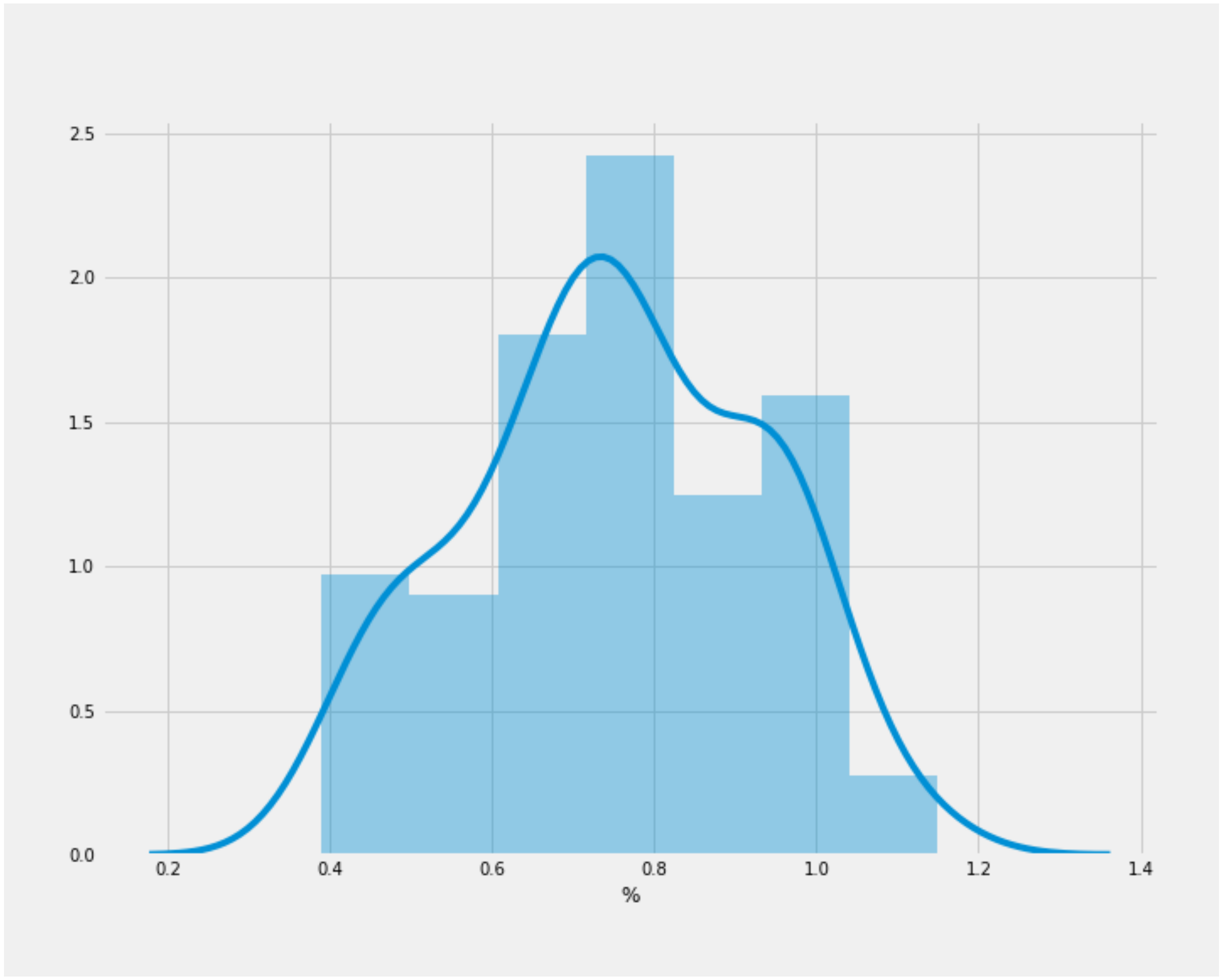


Figure 2. Distribution for Class Frequencies in Training Dataset



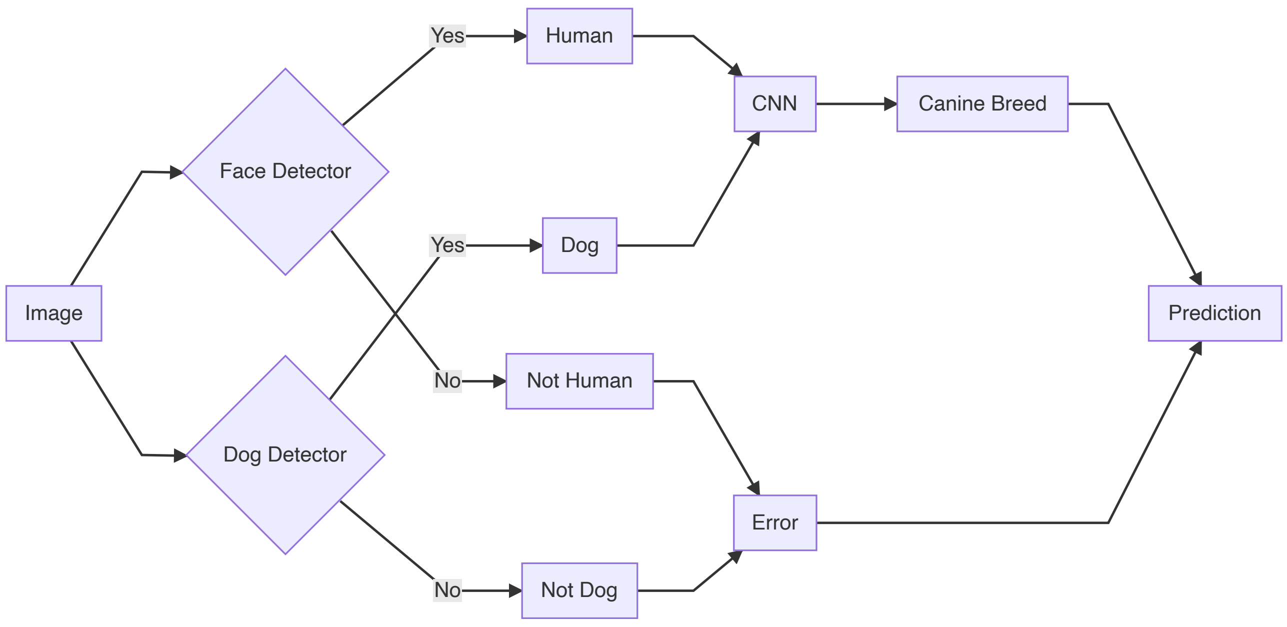
Count	Mean	Std	Min	25%	50%	75%	Max
133	0.7512	0.1777	0.39	0.63	0.75	0.91	1.15

The classifier for this project relies on 3 components that interact with each other to produce a prediction:

1. A face detection algorithm to detect human faces on a given image.
2. A dog detection algorithm to detect whether a given image belongs to a dog.
3. A custom CNN to predict the dog breed for a given image.
4. A CNN that uses transfer learning to predict the dog breed for a given image.

The architecture of the classifier can be roughly visualized as follows:

Figure 3. Classifier Architecture



1. Face Detector

In this project, we use OpenCV's implementation of the Haar feature-based cascade classifier. The pre-trained face detector, allows us to do the necessary data pre-processing steps before feeding them into the `face_detector` algorithm:

1. Convert colored images into grayscale
2. Detect the face in the image.

2. Dog Detector

For the dog detector, we use also a pre-trained network, the **VGG-16** model that been trained on the ImageNet dataset. The images in the dog dataset also require some pre-processing steps prior to feeding them into the pre-trained network. In particular, the Pytorch documentation, specifies that "[a]ll 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 22". In order to achieve this normalization, it will be necessary to resize the images and crop them at the center.

3. CNN

The project guidelines require that we define two different CNN architectures to predict the canine breed. The architecture of the first CNN has to be created from scratch, that is, I defined the structure of the CNN (number of convolutional layers, max pooling layers, etc). The second CNN will use **transfer learning** to make the prediction. From the project description, it is expected that the second CNN will outperform the first one. A key question to ask is: Why is this the case? The explanation has to do the concept of **transfer learning**. The idea is to take advantage of the "knowledge" of a model trained on a large dataset and *transfer* this knowledge to a smaller dataset. In this context, knowledge refers to weights of the convolutional layers of the trained network. Convolutional layers extract general, low-level features that are applicable across images (edges, patterns, gradients) and in this sense are universally applicable to any image. This is why we can use a network trained on unrelated categories and apply it to our own problem[4].

The general outline for transfer learning for object recognition can be summarized as follows[4]:

- Load in a pre-trained CNN model trained on a large dataset
- Freeze parameters (weights) in model's lower convolutional layers
- Add custom classifier with several layers of trainable parameters to model
- Train classifier layers on training data available for task
- Fine-tune hyperparameters and unfreeze more layers as needed

4. Model Evaluation and Benchmarks

For model evaluation, this model will rely on accuracy, that is, the number of correct predictions over the number of predicted observations. For multiclass classification, accuracy is defined as follows:

$$\text{accuracy} = \frac{\text{correctly predicted}}{\text{total observations}}$$

Accuracy is recommended as a performance measure when the classes that comprise the target variable are nearly balanced. In this particular case, the exploratory data analysis suggests that class frequencies in the training set are, on average, balanced with ~60% of the observations being one standard deviation from the mean class frequency. Given these characteristics, accuracy seems like a resonable performance metric for the problem at hand.

Since our data is split in training and validation, we are able to test the accuracy of our model on unseen data during training. As the training advances, the model learns the underlying features of the training data and is better able to generalize, which translates in a lower loss in both training and validations sets. This allow us to select the model with the highest accuracy on the the validation dataset, which is a close proxy for performance on the test set.

The Udacity team defined two different thresholds that should serve as benchmarks, one for the CNN created from scratch and another one for the CNN that uses transfer learning. Given the complexity of the task at hand, it is expected that the CNN created from scratch achieves at least a **10%** accuracy on the test set, while the CNN that uses transfer learning should achieve at least **60%** accuracy on the test set.

Implementation

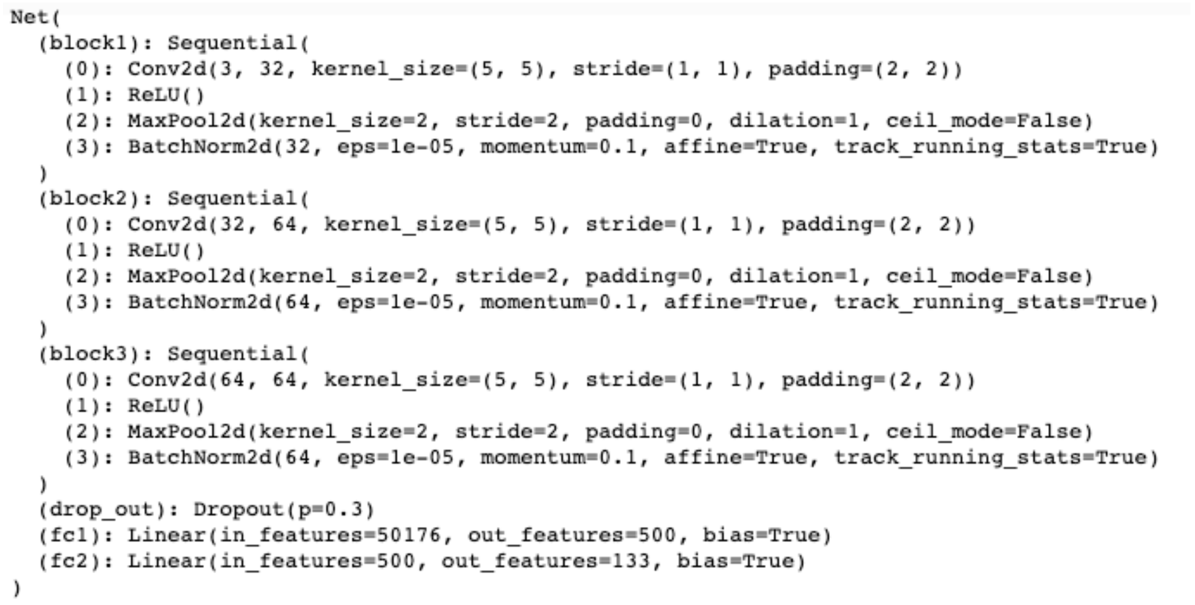
As described in previous sections, the implementation of the dog breed classifier requires the following steps:

1. Face Detector
    - The default face detector algorithm uses OpenCv's **haarcascade_frontalface_alt.xml**. The algorithm was tested on the first 100 images in the human dataset and predicted 98% of the sample correctly. When tested against the the first 100 images in the dog dataset, 17% of the images where identified as human faces.
    - An alternative face detector function was provided, which enables us to use a prefered face detection algorithm. The percentage of correct predictions was very similar to the default algorithm.
2. Dog Detector
    - As required, the default dog detector algorithm uses the **VGG16** model to identify images of dogs. The detector was tested on the first 100 images in the dog dataset and predicted 100% of the sample correctly.
    - In addition to the default detector, an alternative function was provided to load any pre-trained network from Pytorch. As an example, **VGG11** model was loaded and tested against the sample files with similar results to the default detector.
    - In terms of data preparation, the images needed to be resized (224 x 224) and the color channles normalized before feeding them to the models.
3. Custom CNN
    - Define data preprocessing and augmentation
        - For the data augmentation part, the code applies the following transformations to the **training** images:
            - Randomly rotates the image in a 30 degree angle.
            - Randomly flips the image horizontally.
            - Takes a 224x224 square out of the center of the image.
            - Normalizes the color channels to the defaults of Pytorch's pretrained networks.

          These transformations allow the network to better generalize by introducing randomness as it sees the same images in different scales, with different locations, sizes and orientations.
        - With respect to the validation and test datasets:
            - Resize the images to a 256x256
            - Takes a 224x224 square out of the center of the image.

        - Normalized the color channels to the defaults of Pytorch's pretrained networks.
    - Specify model architecture [5]:

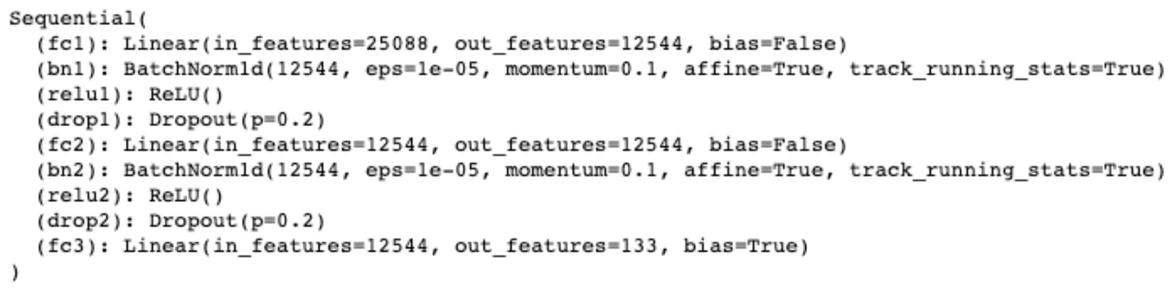
      Figure 4. Custom CNN Architecture

```
Net(
  (block1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (block2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (block3): Sequential(
    (0): Conv2d(64, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (drop_out): Dropout(p=0.3)
  (fc1): Linear(in_features=50176, out_features=500, bias=True)
  (fc2): Linear(in_features=500, out_features=133, bias=True)
)
```

    - Define loss function and optimizer.
    - Train network and display both training and validations loss.
    - Save trained network.
    - Compute network's accuracy on the testing dataset.
4. CNN with Transfer Learning.
    - Define data preprocessing and augmentation (same as custom CNN).
    - Specify model architecture

      Figure 5. Classifier for Transfer Learning CNN

```
Sequential(
  (fc1): Linear(in_features=25088, out_features=12544, bias=False)
  (bn1): BatchNorm1d(12544, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): ReLU()
  (drop1): Dropout(p=0.2)
  (fc2): Linear(in_features=12544, out_features=12544, bias=False)
  (bn2): BatchNorm1d(12544, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2): ReLU()
  (drop2): Dropout(p=0.2)
  (fc3): Linear(in_features=12544, out_features=133, bias=True)
)
```

    - Define loss function and optimizer.
    - Train network and display both training and validations loss.
    - Save trained network.
    - Compute network's accuracy on the testing dataset.
5. Dog Breed Algorithm

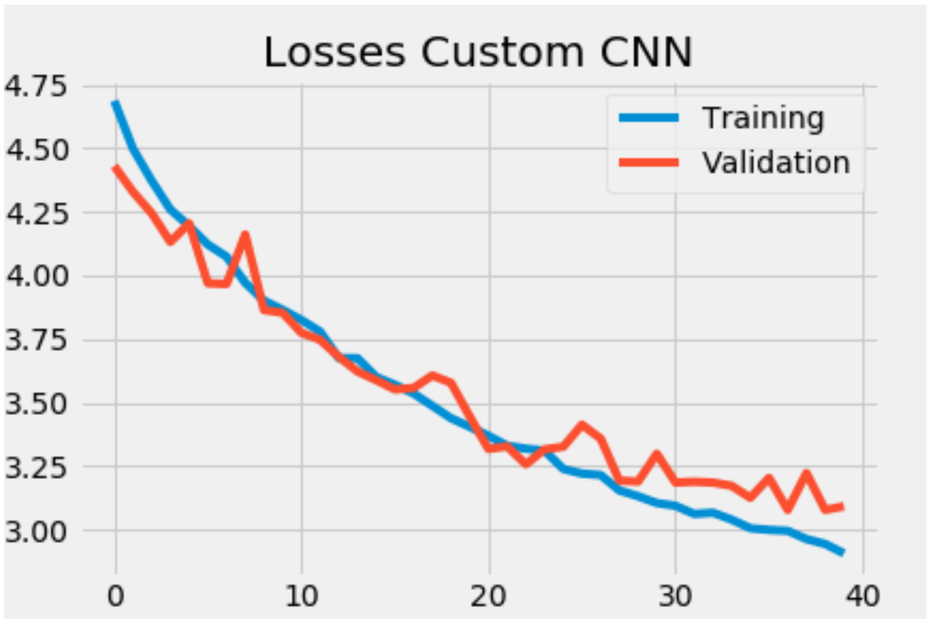
- Define internal logic for classification
- Test algorithm with sample images from the human/dog dataset.

Refinement

As mentioned in the Model Evaluation and Benchmark section, the project required that the custom CNN architecture achieved at least **10%** accuracy on the testing dataset. The initial architecture was very similar to the on presented above with one important exception, there weren't any Batch Normalization layers. The accuracy of this initial model on the testing set was ~16%, which was enough to meet the project requirements. This result was significantly improved by adding Batch Normalization layers between blocks without modifying the initial architecture or increasing the numbers of epochs. The proposed change increased the accuracy on the testing set to 25%. After the model was saved, it was retrained for an additional 40 epochs to achieve an accuracy of 31% on the testing set.

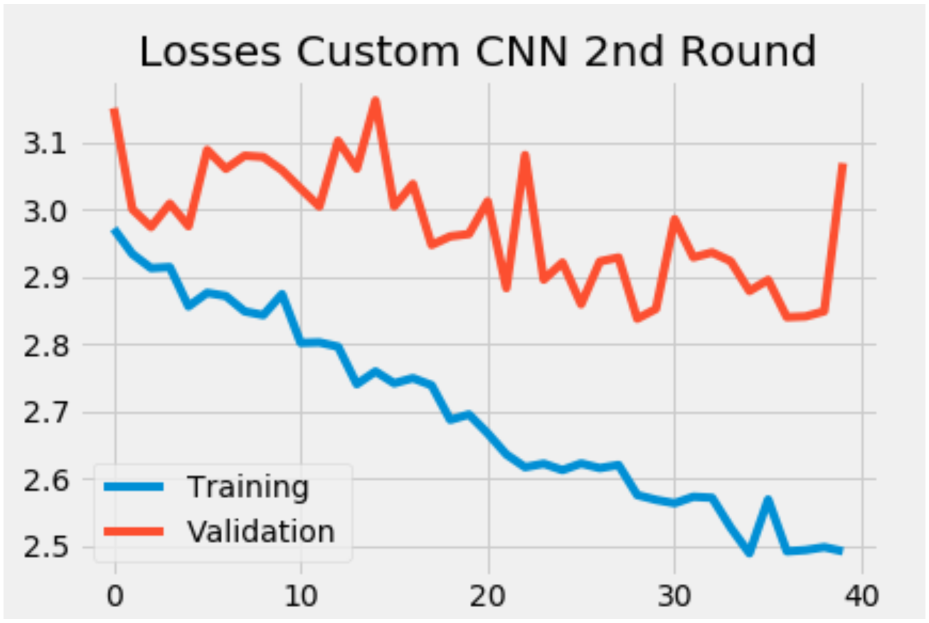
The following figure presents the evolution of the average training and validation loss for the first 40 epochs:

Figure 6. Training and Validation Loss in Custom CNN (First 40 Epochs)



The divergence of training and validation loss after second training round is in an indication of overfitting. This suggests that additional modifications would be needed to the custom CNN to achieve greater accuracy.

Figure 7. Training and Validation Loss in Custom CNN (Additional 40 Epochs)



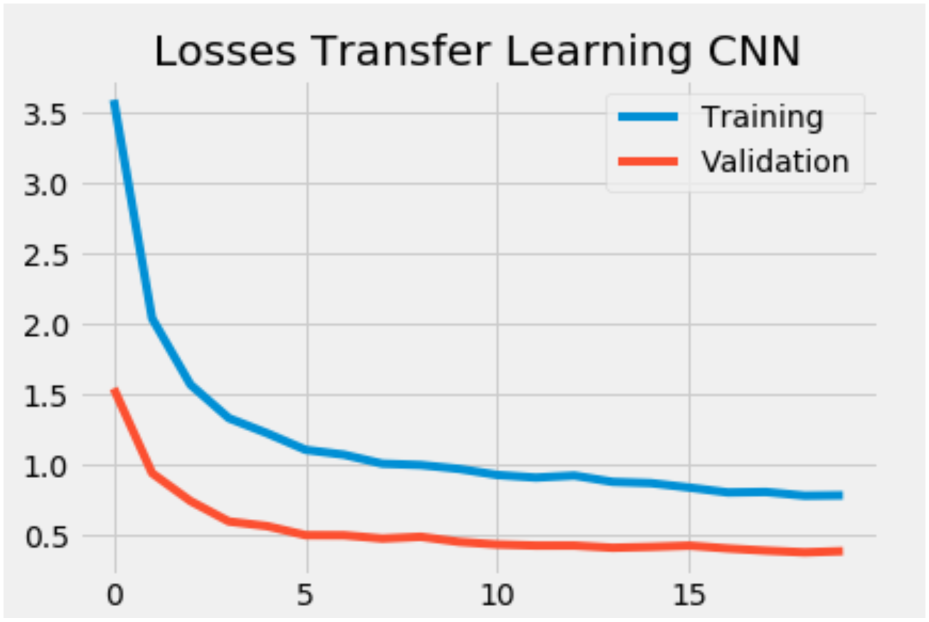
With respect to the transfer learning solution, the project required that the model achieved at least **60%** accuracy on testing set. The final architecture for this solution was achieved in the following way:

- Freezing the weight parameters of the **VGG16** model
- Attach to the **VGG16** model a fully connected network with
    - 1 hidden layer
    - Batch Normalization layers
    - ReLU activation functions.
    - Drop-out Layers with 0.2 probability to avoid overfitting.

  The proposed solution achieved an 84% accuracy on the testing dataset, which might be considered a good result given the close resemblance between different dog breeds (mininal inter-class variation) and also the high variation within some dog breeds (intra-class variation). As an example of the latter, think about the different types of Shiba Inu (White, Brown, Sesame, and Black).

In a similar way to the custom CNN architecture, the following figure presents thhe evolution of the average training and validation loss for the transfer learning model:

Figure 8. Training and Validation Loss in Transfer Learning CNN (20 Epochs)



Model Evaluation and Validation

Custom CNN

The custom CNN model was trained for 40 epochs with a learning rate of 0.01 and a batch size of 32. Given the relative simplicity of the model, the training time is a little bit over an hour. The final architecture of custom model was presented in the figure above , however a more detailed description of the components is in order.

The first Convolutional layer consists of 32 channels of 5 x 5 filters, stride 1 and padding. I chose these values for the stride and padding because I wanted to keep the size of the input and output tensors constant. The following formula [5] allows you to compute the output size from either a convolutional filtering or pooling operation:

$$Wout = \frac{Win - \text{Kernel Size} + 2 \cdot \text{Padding}}{\text{Stride}} + 1$$

Therefore, to the ouput for any channel of the convolutional layer will be:

$$W_{\text{out\_conv}} = \frac{224 - 5 + 2 \cdot 2}{1} + 1 = \frac{223}{1} + 1 = 224$$

The Maxpooling layer samples down the data by a factor equal to the size of the stride, in this case equal to 2. The size of the output after the maxpooling operation is given by the formula:

$$W_{\text{out\_max}} = \frac{224 - 2 + 2 \cdot 0}{2} + 1 = \frac{222}{2} + 1 = 112$$

After the maxpooling operation takes place, the output is normalized before being fed into the next block. The reason for doing this is that "the distribution of each layer's inputs changes during training, as the parameters of previous layers change" Ioffe, S., & Szegedy, C. (2015). This phenomenon is known as *internal covariate shift* and Batch Normalization was proposed as a solution to this problem. Although Batch Normalization has proven to be very effective in training image classifiers, the exact reasons why it works are still a matter of theoretical discussions. However, among its many side effects, it appears that the primary one is that of regularization [6].

Block 2 and 3 of the CNN architecture follow a similar logic by keeping the input and output tensors constant. The only difference is that both blocks double the number of channels in the Convolutional layer to 64. Here is a summary of how kernel size, stride and padding values change the input/output size through these blocks:

Conv Block2

$$W_{\text{out\_conv}} = \frac{112 - 5 + 2 \cdot 2}{1} + 1 = \frac{111}{1} + 1 = 112$$

MaxPool Block2

$$W_{\text{out\_max}} = \frac{112 - 2 + 2 \cdot 0}{2} + 1 = \frac{110}{2} + 1 = 56$$

Conv Block3

$$W_{\text{out\_conv}} = \frac{56 - 5 + 2 \cdot 2}{1} + 1 = \frac{55}{1} + 6 = 56$$

MaxPool Block3

$$W_{\text{out\_max}} = \frac{56 - 2 + 2 \cdot 0}{2} + 1 = \frac{54}{2} + 1 = 28$$

After passing the third block of this architecture, the output is flattened and passed to an intermediate layer of 500 fully connected nodes before being fed into the final output layer with 133 nodes corresponding to the different dog breeds. As mentioned before, the final model met project requirements by achieving a 25% accuracy on testing set.

Transfer Learning CNN

Compared to the custom CNN model, the transfer learning CNN was only trained for 20 epochs with the same learning rate and batch size. As described in the Model Evaluation and Validation section, the task was to train a new classifier by leveraging the information already contained in the **VGG16** model. The classifier contained only 1 hidden layer, which allowed for a quick training time ~40 min.

Despite the lower number of epochs compared to the custom CNN, the transfer model is able to achieve an accuracy of 83% on the testing dataset. This is a clear illustration of the advantages of transfer learning. By explointing the information contained in the weights of the layers of the **VGG16** model, the transfer model is able to quicky identify images of dogs, while the (relatively simple) classifier focuses on the differences accross breeds.

Justification

In this project, I created a classifier accepts images of both dogs and human beings and provides a prediction of the canine breed for the dog or the closest resembling dog breed in case of receiving a human image. To accomplish this task, the classifier relies on two pre-trained networks, one to detect humans and one two detect dogs, as well as on Convolutional Neural Network (CNN) that uses transfer learning to classify dog breeds.

In particular, the project had two different benchmarks:

1. To build a custom CNN that achieved at least 10% accuracy on the testing set
2. To build a transfer learning CNN that achieved at least 60% accuracy on the testing set

With respect to the first point, the architecture described in the Implementation section for the custom CNN achieved a 25% percent accuracy after the first training round and 31% after a second round of training. In this sense, the proposed solution surpased the established benchmark by adding a few refinements to the architecture (Batch Normalization and Dropout layers) and increasing the training period. However, this architecture is too simple to achieve a higher accuracy on the testing set, which can be observed by the behavior of the validation loss during the second training period.

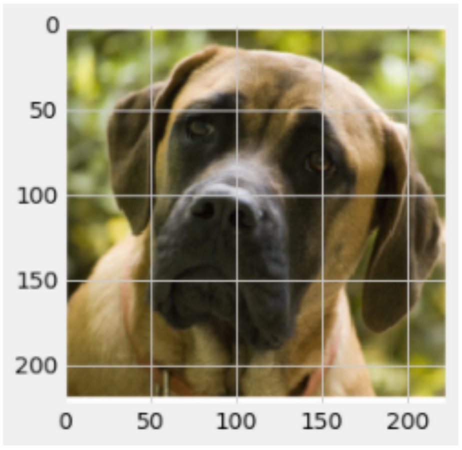
With regards to the second point, the new classifier attached to the **VGG16** model was able to achieve a 84% accuracy on the testing set, surpassing the established benchmark. Even though the new classifier had a relatively simple architecture, the addition of Batch Normalization and Dropout layers reduced overfitting and provided an adequate solution to the problem.

Finally, the algorithm comprised of the human detector, dog detector and the dog breed classifier performed as expected by providing a prediction of the canine breed in case of receiving a dog image or the closest resembling dog breed in case of receiving a human image.

Conclusion

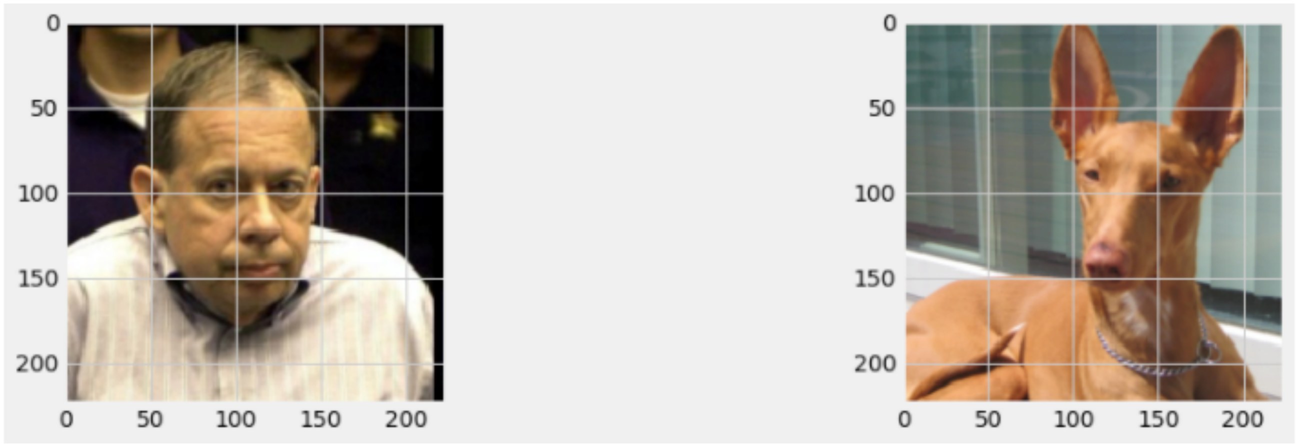
Given the accuracy of the model for predicting canine breeds, 84%, the algorithm performed as expected on the sample test images.

```
predict_breed_transfer(dog_files_short[8])
Predicted class index: 102
'Mastiff'
```

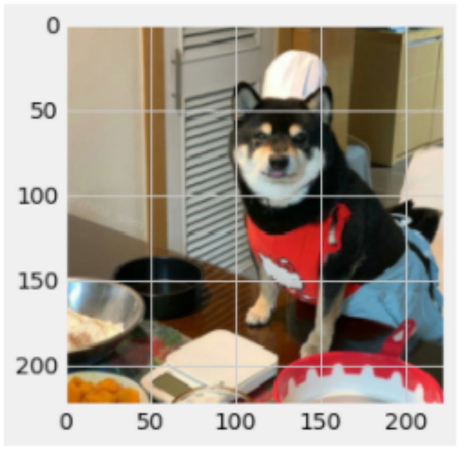


```
human_example = process_image(human_files[17])
dog_example = human_to_dog(human_files[17])

"Predicted class index: 119"
"Predicted Dog Breed: Pharaoh hound"
```



However, there is a lot of room for improvement, this can be seen in the sample image downloaded from the internet, a **Shiba Inu** wearing cooking clothes. Even though it is simple for the human eye to identify a dog in the image, the addition of clothing already causes the `dog_detector` to give a wrong prediction.



Improvements

Even though it is simple for the human eye to identify a dog in the image, the addition of clothing already causes the `dog_detector` to give a wrong prediction.

Therefore, the first step to improve the algorithm would be to use a better dog detector for a given image. An example can be seen in the `alternative_dog_detector` function, which allows us to use any pre-trained network available in Pytorch for image classification, for example, **VGG11**. This alternative detector allows us to correctly predict that the image belongs to a dog.

A second step would be to increase the number of images in our training set by adding images of different dog breeds wearing clothes. It is not uncommon to see images in social media of pets wearing some form of clothing or costume, we would like our model to be able to abstract these details and focus on identifying the breed. This poses an interesting challenge to the current implementation as seen in the model prediction. The model prediction for this image is a Chihuahua!!! We know this couldn't be further from the truth.

Therefore, the third improvement would be to retrain the transfer model with this enhanced dataset. Models like **VGG16** were trained on the Imagenet dataset, which contains images for common clothing pieces like t-shirts, sweatshirts, etc. In this sense, we can rely on the weights of the pre-trained network to identify features that belong to clothing pieces, while the classifier can focus on learning the specific weights related to dog-breed-clothing combinations.

Finally, we could build a more complex classifier by increasing the number of hidden layers to better learn the specific features that differentiate dog breeds.

---

1. Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., ... & Asari, V. K. (2018). The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*. ↩

2. Sebe, N., Cohen, I., Garg, A., & Huang, T. S. (2005). *Machine learning in computer vision* (Vol. 29). Springer Science & Business Media. ↩

3. Cox, L. (2017, July 09). 5 Applications of Facial Recognition Technology. Retrieved October 12, 2020, from https://disruptionhub.com/5-applications-facial-recognition-technology/ ↩

4. Koehrsen, W. (2018, November 26). Transfer Learning with Convolutional Neural Networks in PyTorch. Retrieved October 12, 2020, from https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd09190245ce ↩ ↩

5. The architecture ws inspired by the following tutorial: Thomas, A., 2020. *Convolutional Neural Networks Tutorial In Pytorch - Adventures In Machine Learning*. [online] Adventures in Machine Learning. Available at: https://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-in-pytorch/ [Accessed 30 October 2020]. ↩ ↩

6. D2l.ai. 2020. *7.5. Batch Normalization — Dive Into Deep Learning 0.15.0 Documentation*. [online] Available at: http://d2l.ai/chapter_convolutional-modern/batch-norm.html [Accessed 30 October 2020]. ↩