Machine Learning Engineer Nanodegree

Capstone Project

Dog Breed Classifier with CNNs

"Kirolos Wahba Moheeb"

October 31st, 2020

I. Definition

Project Overview

The Dog breed classifier is a well-known problem in Machine Learning. The problem is to identify a breed of dog if dog image is given as input. So if dog owners are unsure about their dog's breed, a dog-breed classifier can help them identify their dog's breed. This would be especially helpful for inexperienced dog owners.

This is a multi-class classification problem where we can use supervised machine learning to solve this problem.

In this project I created a CNN model that can classify dog breeds. A dataset from Udacity is available for training the model, which includes 8351 dog images with exactly 133 dog breeds and 13233 human images.

The project is composed on different components:

- Model selection & testing.
- Algorithm development.
- Test the used algorithm.

Problem Statement

In this project it is important to build a data processing pipeline to classify real-world, user-supplied images. On the basis of a image of a dog, an algorithm is supposed to give an estimate of the breed of the dog. If the image of a person is given, the algorithm should reproduce the most similar dog breed. In addition, the accuracy should approach the benchmark model accuracy

To achieve this goal transfer learning is used. In transfer learning approach feature part of the network is freezed and only classifier is trained.

The following therefore applies:

- It must be recognized on an image whether a person or a dog is present.
- If neither of the two is the case, an error message is issued
- The breed of the dog must be estimated or the resembling dog breed for humans.
- The accuracy should approach the benchmark model accuracy.
- Transfer learning will be used to reach this goal

Metrics

I would use test accuracy as an evaluation metric to compare my models with the benchmark model. Additionally, the performance will be compared between the following convolutional neural networks: dog breed's classifier trained from scratch and dog breed's classifier trained with transfer learning**.**

$$\text{Accuracy} = \frac{Number\ of\ items\ correctly\ classified}{All\ classified\ items}$$

II. Analysis

Data Exploration

For this project, the input format must be of image type, because we want to input an image and identify the breed of the dog.

The following datasets provided by Udacity will be used:

- **The dog dataset** with 8351 dog images which are sorted into train (6,680 Images), test (836 Images) and valid (835 Images) directories. Each of this directory (train, test, valid) have 133 folders corresponding to dog breeds. The images are of different sizes and different backgrounds, some images are not full-sized. The data is not balanced because the number of images provided for each breed varies. Few have 4 images while some have 8 images.
    - **Examples:**

Akita

Border collie

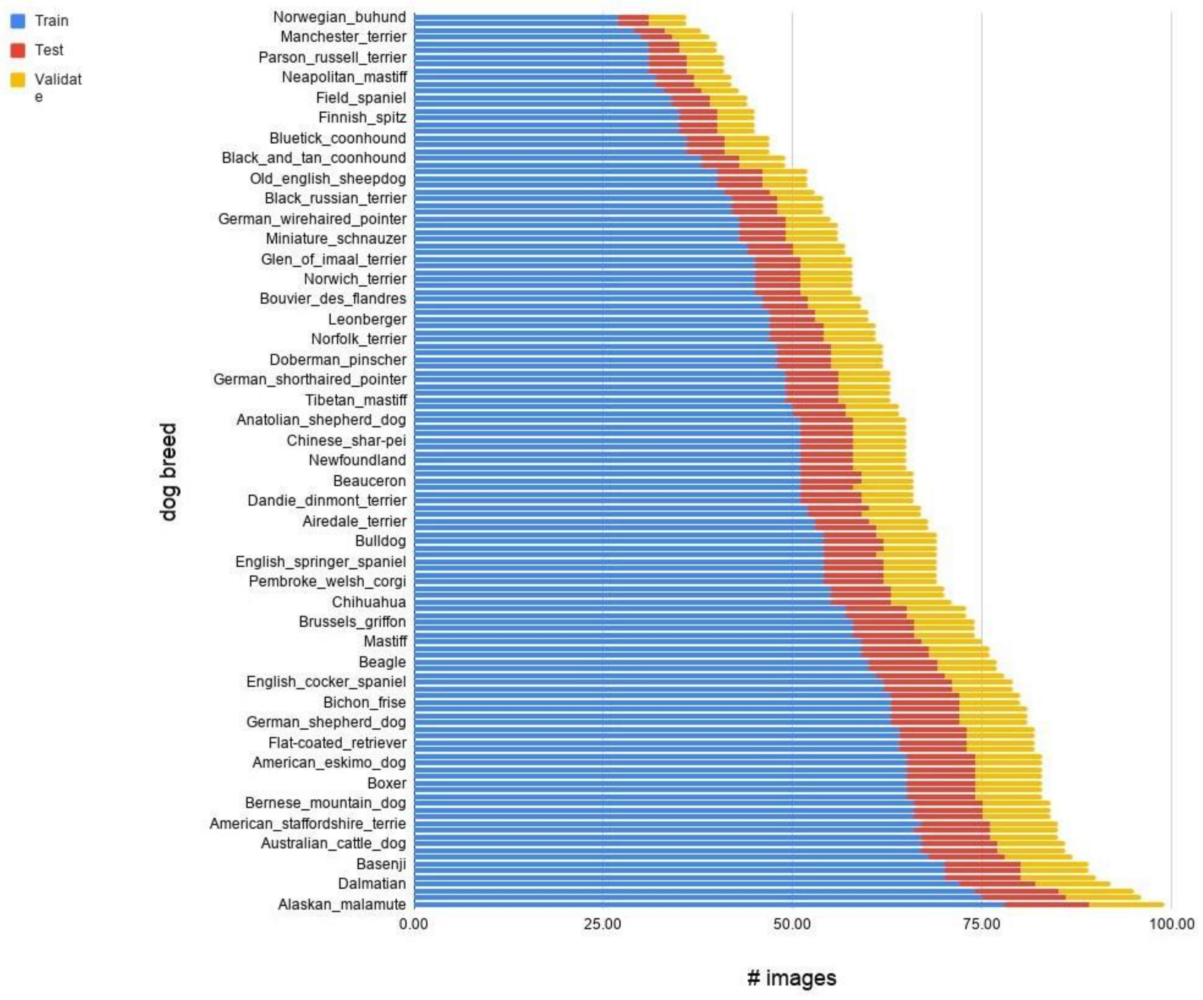
- **The human dataset** with 13233 human images which are structured in folders by celebrities' name (5750 folders). All images are of size 250x250. Images have different background and different angles. The data is not balanced because we have 1 image for some people and many images for some.
    - **Examples:**

Aaron Patterson

Art Hoffmann

Exploratory Visualization

There are 133 different breeds of dog in this dataset, which represent 133 classes. There are 8351 images in total, and these are split into 3 groups of images, for training, testing and validation. Not all dog breeds have the same number of images, and thus there is an imbalance in the classes. For example: there are 36 images of Norwegian Buhunds, whereas there are 99 images of Alaskan Malamutes. The stacked bar chart below shows the number of images of each breed. The next visualization shows the total number of images in each class and it also shows the breakdown between training, testing and validation. Approximately 75% of the images for each class are dedicated to training, while the other 25% is split between training and validation.



Algorithms and Techniques

I will divide the used algorithms into the following 5 sections as these are the skeleton of the project:

- **Human Face Detector** Here I will use Cascade Classifiers from Opencv to recognize the face of a person on an image. If a face is recognized this means that a person is present in the picture.
- **Dog detector** To be able to recognize dogs on an image I will use a pretrained VGG16 model. This will be implemented with the open source machine learning framework PyTorch9. The VGG16 model won the 2014 Imagenet competition and achieves 92.7% top-5 test accuracy in the ImageNet dataset, which is a dataset of over 14 million images belonging to 1000 classes. This pretrained model is only used for prediction and will not be trained or similar.
- **Dog Breed Classifier from Scratch** Here I will use PyTorch to design, train and test a model from scratch.
- **Dog Breed Classifier with transfer learning** In this section I will apply the technique transfer learning. In transfer learning, a pretrained model is used which is then only modified according to the intended use. In my case I will use the pretrained ResNeXt-101-32x8d model10. The model won the second place of the Imagenet competition in 2016. This will be implemented with PyTorch. I will freeze the feature part of the model and only modify and train the classifier again.
- **Running the Application** Finally, I will bring together everything above, except the Dog Breed Classifier from Scratch. Here, first of all it is recognized whether a dog or a human is present in image. If both are not present in an image, an error message will be returned to the user. The image is then passed to the Dog Breed Classifier algorithm, which will then return the predicted breed of the dog. The predicted breed is then displayed with the image to the user. It is also shown whether the image is of a dog or a human.

Benchmark

I found an interesting benchmark model in this paper (2). On page 4 in Figure 5 the accuracy of different state of the art models is compared. So I choose the ResNet50+FT+DA with test accuracy of 89.66% as my benchmark model.

III. Methodology

Data Preprocessing

As we have already noticed, not all images have the same resolution. Also the distribution of the images is not the same in the various dog breeds. Here there is a slight data imbalance. For the data preprocessing step the PyTorch module torchvision.transforms is used.

All images of the section test and valid are first reduced to 256x256 pixels. Then a CenterCrop to 224x224 pixels is performed.

All images of the section train are first reduced to 256x256 pixels. Afterwards, data augmentation is applied to counteract the data imbalance. After the images are reduced in size, a RandomResizedCrop to 224x224 pixels is performed. Then a RandomHorizontalFlip is performed and finally a RandomRotation of 10 degrees.

The mentioned reduction to 224x224 pixels serves as a preparation for the ResNeXt-101-32x8d model. Because this was trained to this input size of the images. In addition, the images are normalized also for the same reasons of preparation for the ResNeXt-101-32x8d model.

Implementation

In this section I will explain in depth which algorithms and techniques I have used. I will divide the used algorithms into the following 5 sections as these are the core of the project:

- **Human Face Detector** Here I used one of the Cascade Classifiers from Opencv to recognize the faces. This Cascade Classifiers checks whether there is a face of a person in a image. First I used the cascade classifier of the file haarcascade_frontalface_alt.xml and implemented it in the function face_detector. This already brought good results. At 100 test images with humans 99% were recognized as humans. After that I created another cascade classifier with the file haarcascade_frontalface_alt2.xml and implemented it into the function face_detector_ext. This one is able to recognize 100% of humans as humans. I will use the latter cascade classifier in the further course.

- **Dog detector** To be able to recognize dogs on an image I used a pretrained VGG16 model. This is implemented with the open source machine learning framework PyTorch. This pretrained model is only used for prediction and will not be trained or similar. First of all, I transform the image for the VGG16 model to 224x224 pixels since it is the input size the model was trained for. After that the image is converted it to a PyTorch tensor and is normalized. The parameters for normalization are the same as those used when training the VGG16 model. Then I pass this image to the prediction and get an index value back. As a reminder, the VGG16 model was trained on the ImageNet dataset. The record has 1000 categories. The indexes between 151 and 268 are dog categories. This means that if the VGG16 model returns an index in this range, a dog is detected on the image. The function VGG16_predict takes care of the prediction and the function dog_detector evaluates the index and returns true or false accordingly.

- **Dog Breed Classifier from Scratch** Here I used PyTorch to design, train and test a model from scratch. The architecture of the model will be based on the maximum possible hardware of my PC. When designing the architecture of the model from scratch I followed the architecture of established models like VGG16. But I had to keep my model very simple, because I quickly reached the limits of my graphics card when experimenting and reconstructing VGG16. My PC has an Nvidia GTX 1080Ti with 11GB of graphics card memory and just loading the reconstruction of VGG16 has already used over 8GB of graphics memory. So the training was no longer possible. So I was forced to simplify my model and to add a pooling layer after each convolutional layer to reduce the features. And based on my available hardware I decided to use the following model:
    - Convolutional Layer with 3 input dimensions and 32 output dimensions - kernel size 3
    - Relu Activation function
    - Pooling layer - kernel size 2
    - Convolutional Layer with 32 input dimensions and 64 output dimensions - kernel size 3
    - Relu Activation function
    - Pooling layer - kernel size 2
    - Convolutional Layer with 64 input dimensions and 128 output dimensions - kernel size 3
    - Relu Activation function Pooling layer - kernel size 2
    - Convolutional Layer with 128 input dimensions and 128 output dimensions - kernel size 3
    - Relu Activation function
    - Pooling layer - kernel size 2
    - Flatten layer to convert the pooled feature maps to a single vector with a length of 25088

    - Dropout with a probability of 0.25
    - Fully connected Linear Layer with an input size of 25088 and an output size of 4096
    - Relu Activation function
    - Dropout with a probability of 0.25
    - Fully connected Linear Layer with an input size of 4096 and an output size of 133

- **Dog Breed Classifier with transfer learning** In this section I applied the technique transfer learning. In my case I will use the pretrained ResNeXt-101-32x8d model. This will be implemented with PyTorch. I will freeze the feature part of the model including the Adaptive Average Pooling layer and only replace and train the classifier part. First of all I loaded the pre-trained model and freezed all layers using requires_grad = False. Then I replaced the only fully connected layer (fc), added a dropout layer (drop) and inserted a fully connected layer (fc1) as last layer. This last layer has 133 output neurons (output features). If layers are replaced in this way, they are automatically set back to requires_grad = True and can therefore be trained.

- **Running the Application** In this section the run_app function brings everything together. The Human Face Detector, the Dog Detector and the Dog Breed Classifier with transfer learning. Here, first of all it is recognized whether a dog or a human is present in a image. If both are not present in an image, an error message will be returned to the user. The image is then passed to the Dog Breed Classifier function predict_breed_transfer which transforms the image, makes the predictions, and returns the predicted breed. The predicted breed is then displayed to the user with the image. It is also shown whether the image is of a dog or a human.

Refinement

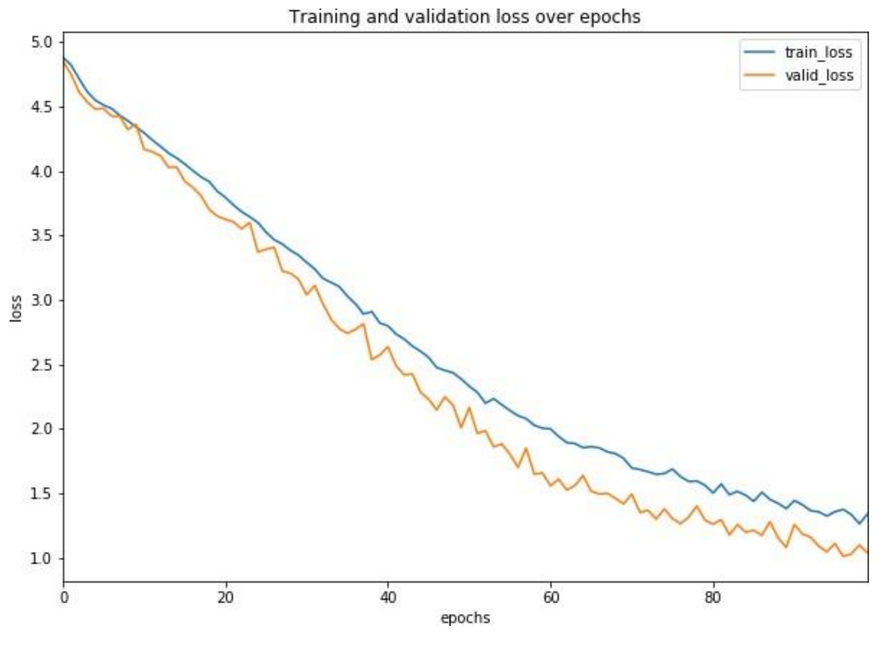
During training of the ResNeXt-101 model, I tuned various model hyperparameters in order to improve the performance of the model. I initially trained the model for 10 epochs, with a learning rate of 0.04. after tuning the number of training epochs to 40, and reducing the learning rate to 0.01 I achieved better performance. And also went from ResNeXt50 model to ResNeXt-101 model, As this model has a better performance and gives me a higher accuracy.

IV. Results

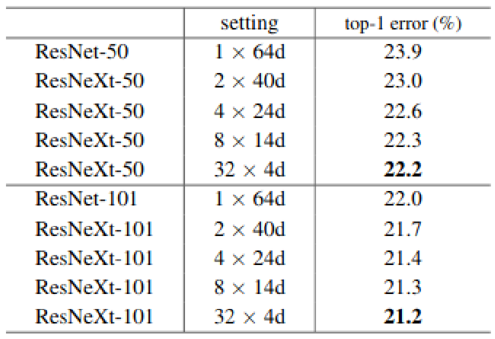
Model Evaluation and Validation

During the training of both models the model was validated with the Validation set after each epoch. After the training the models were tested with a test dataset. I will describe the training process for each model here.

- **Model from Scratch**
  Here I would like to discuss the selected hyperparameters:
  - A learning rate of 0.02 worked very well here.
  - 100 epochs were chosen for this task.
  - As optimizer I chose the stochastic gradient descent (SGD) optimize because it works very well for image classification tasks. It turned out to work also very well for the given task.
  - The following plot shows the progress of the training loss and the validation loss over the epochs during training:



- **Model with transfer learning**
  I have chosen the ResNeXt-101 model because it is one of the best performing pre-trained models available. It performs better than the ResNet50 model in the Top-1 error comparison based on the ImageNet dataset:

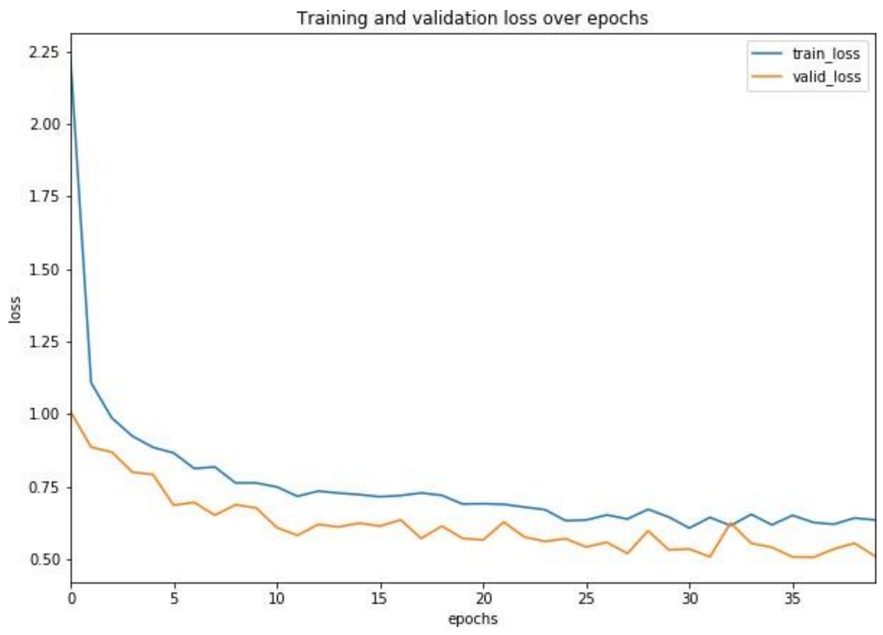


	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

Training and validation loss over the epochs

Here I would like to discuss the selected hyperparameters:

- A learning rate of 0.01 worked very well here.
- A momentum of 0.9 helps for faster converging.
- 40 epochs were enough for this task.
- As optimizer I chose the stochastic gradient descent (SGD) optimizer because it works very well for image classification tasks. It turned out to work also very well for the given task.
- The following plot shows the progress of the training loss and the validation loss over the epochs during training:



Justification

Here I will show the results of both models and finally give a comparison to the chosen benchmark:

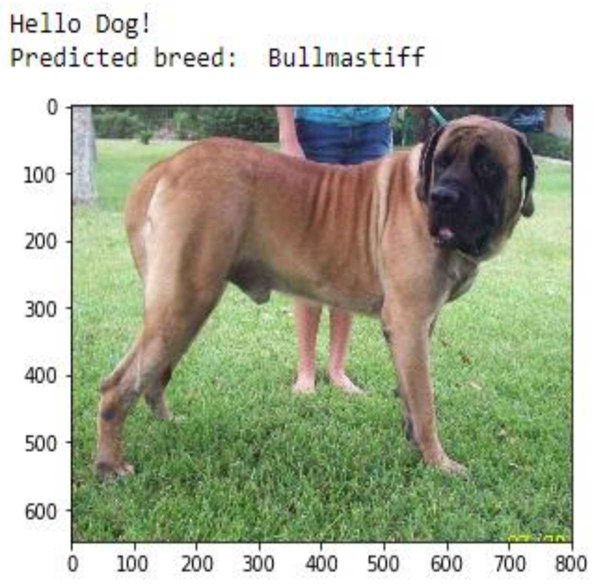
- **Model from Scratch**
    - Test Accuracy: 74.85%
- **Model with transfer learning**
    - With this model the results are also better than expected. Here is the test accuracy result:
        - Test Accuracy: 86.26%

Benchmark Comparison: The Benchmark Model ResNet-50 + FT + DA (2)has a test accuracy of 89.66%. Unfortunately, I could not reach this accuracy with my two models. In the Model with transfer learning I really approached this goal. This is most likely due to the fact that the Benchmark Model had 20580 images for training and testing with only 120 dog breeds. This is more than twice as many images as I have available.

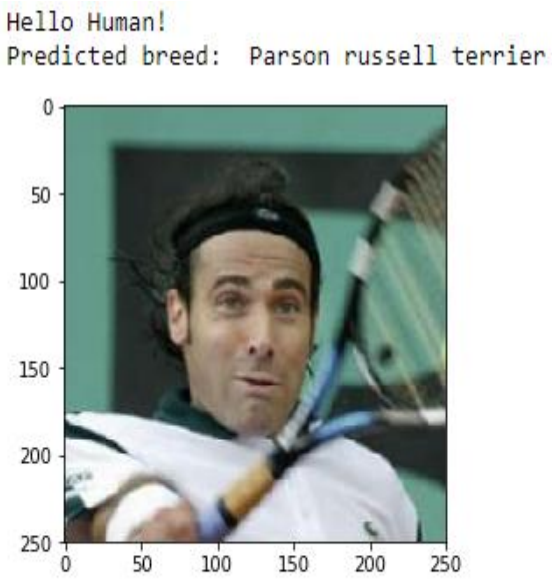
V. Conclusion

Free-Form Visualization

Here are some outputs of the implemented algorithm:



```
Hello Dog!
Predicted breed:  Bullmastiff
```



```
Hello Human!
Predicted breed:  Parson russell terrier
```

We can see that the algorithm works as intended, correctly identifies dog breeds and gives a breed estimation for humans.

Reflection

The process used for this project can be summarized into the following steps:

1. An initial problem and relevant public dataset of images were found.
2. The images were downloaded.
3. A Human Face Detector was developed.
4. A dog detector was developed.
5. The images were preprocessed and data augmentation was applied.
6. A Model from Scratch was developed, trained and tested.
7. A Model with transfer learning was developed, trained and tested.
8. An application was designed and developed to show the user the top 3 predictions for a given image.
9. Every function and every part of the project has been clearly documented.
10. The GitHub repository has been clearly documented and provided with README files.

I found the whole project very interesting. But the only difficult step was step 6, as I first tried to reconstruct the VGG16 model but that did not work for me. The final solution works as expected and seems to be a good choice for problems like these. As it can easily be implemented in web app or other platforms like android or windows or IOS.

Improvement

The improvements can be:

1. More training data can improve the accuracy of the trained model.
2. Hyper parameter tuning will also help in improving performance.
3. If a dog and a human can be seen in an image, an additional function could be implemented here to make a prediction for both
4. Identify multiple dogs or multiple human faces.
5. Increase the number of dog breeds in the dataset. As the FCI (3) currently recognizes over 352 dog breeds. If this diversity were represented in the data set, the algorithm would be able to provide an estimate for all these dog breeds.

References

1. Original repo for Project - GitHub: https://github.com/udacity/deep-learning-v2- pytorch/blob/master/project-dog-classification/
2. Ayanzadeh, A.; Vahidnia, S. Modified Deep Neural Networks for Dog Breeds Identification. Preprints 2018, 2018120232 (doi:10.20944/preprints201812.0232.v1).
3. FCI. FCI Website. url: http://www.fci.be/en/Presentation-of-ourorganisation-4.html (visited on 10/25/2020).
4. Pytorch Documentation: https://pytorch.org/docs/master/
5. Udacity. The dog dataset. url: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip (visited on 10/20/2020).
   — The human dataset. url: https://s3- us- west- 1.amazonaws.com/udacity-aind/dog-project/lfw.zip (visited on 10/20/2020).