Capstone Project — A dog breed classifier using convolutional neural networks

Stefan Oelsner

---

1. Definition

1.1. Project overview

Machine learning, computer vision in particular, has gained considerable momentum reaching every-day relevance for a large portion of society. When unlocking your phone, chances are a computer vision algorithm is running detection and learning specific properties of the individuals face[1]. When thinking about the future of mobility, self driving cars come to mind. At the heart of this technology lie sophisticated image recognition algorithms[2]. Medical applications like real-time skin cancer detection have the potential to make an impact on population health as well as providing self-diagnosing capabilities to individuals.[3]

In light of those developments, this project focuses on classifying dog breeds based on images using machine learning techniques. Due to the vast diversity in dog breeds and their associated attributes (the American Kennel Club (AKC) currently recognizes about 155 breeds[4]) this is a particularly fitting problem to solve using convolution neural networks.

1.2. Problem statement

The desired outcome of this project is to build a complete pipeline that allows to upload an image which will be then classified by an algorithm and outputs one of the following results:

- If a dog is detected, the corresponding breed will be returned

Email address: stefanoelsner@me.com (Stefan Oelsner)

- If a human face is detected, the algorithm will state it. Afterwards, the image will be processed by the dog breed classifier and the most similar breed will be returned
- If neither a human face nor a dog breed is detected, an error message will be returned

The implementation will consist of multiple elements working as an ensemble to provide the above mentioned functionality.

A pre-trained convolution neural network (CNN) will be used to determine, if a dog is present or not. A complementary face detection algorithm will classify a human face. For the classification itself, two different paths will be explored. First, a custom designed CNN will be developed and trained. Second, transfer learning techniques will be used to combine a pre-trained image classifier with a custom dense layer output.

1.3. Metrics

The metric to evaluate the performance of the classifier will be accuracy which is defined as follows:

$$accuracy = \frac{true\ positives + true\ negatives}{true\ positives + true\ negatives + false\ positives + false\ negatives} \quad (1)$$

[1]

The multi-class classification problem this project consists of will be measured by the percentage of correctly classified images on given images of known class. Alternative possible metrics include precision and recall that penalize false positive and false negatives respectively[5] . In this case, all misattribution of dog breeds are equally negative and no specific class is preferred. In conclusion, accuracy will be the metric of choice for this problem.

2. Analysis

2.1. Data exploration and visualization

All source data is provided by udacity. The data includes 13233 sample images of humans wrapped inside folders for the specific individual. 8350 images of dogs are provided. The folder structure is as follows:

[1] tp = true positve, tn = true negative, fp = false positive, fn = false negative

- train (training dataset) -> dog breed -> sample image
- valid (validation dataset) -> dog breed -> sample image
- test (testing dataset) -> dog breed -> sample image

80% of all dog images are contained within the train folder that will be used for training the neural network. The remaining 20% of images are split equally between validation and test data setsFigure 1. Figures 2 and 3 show examples of the source images at hand.

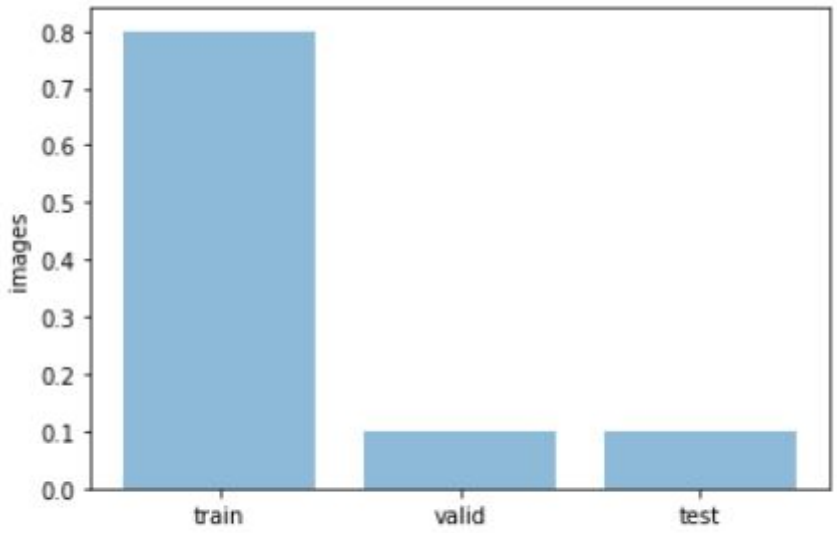


Figure 1: Train/ valid/ test split

2.2. Algorithms and techniques

2.2.1. Haar feature-based cascade classifiers

One pillar of this project consists of facial recognition algorithms to detected humans in images. Here, openCV[6] is used as framework to accomplish the task. The foundation of this library is object/ face detection using Haar feature-based cascade classifiers. This methodology is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features"

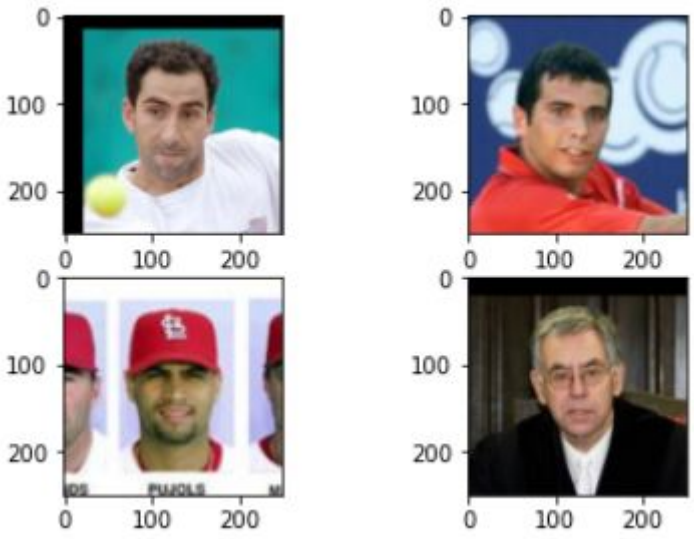


Figure 2: human faces - sample images

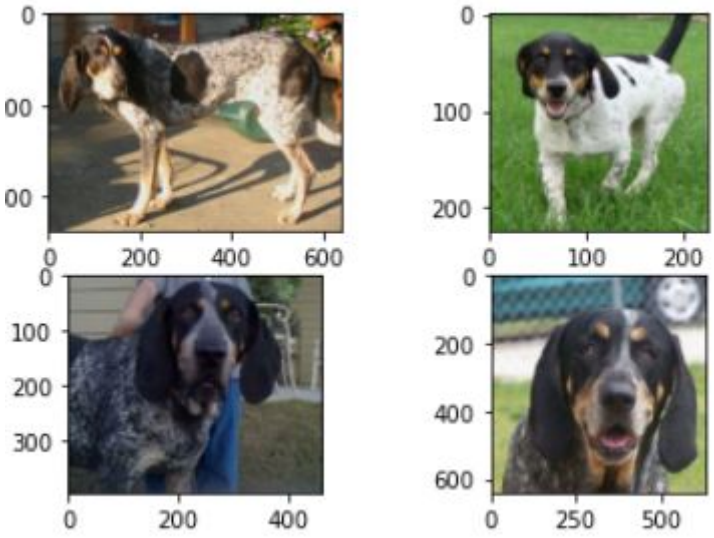


Figure 3: dog breeds - sample images

in 2001[6,7]. Much like Kernels used for convolutional neural networks, Haar features calculate single values for the image at hand by subtracting the sum of pixels under the dark portion of the feature from the sum of pixels under the white portion.

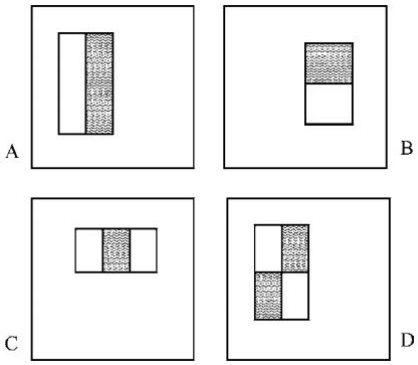


Figure 4: Haar features as described in[7] .

Those features are applied on all possible scales to all areas of the underlying image. On a base image of 24x24 pixels, this results in 160000+ possible features. For demonstration purposes, Figure 5 shows the application of two features (edge detection and line detection) labeling the eye portion of the image.

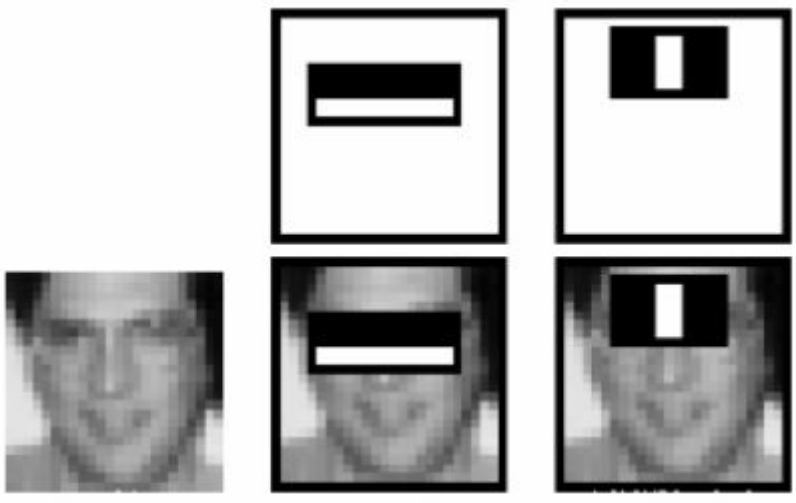


Figure 5: Application of Haar feature on sample image

The algorithm is trained on positive and negative examples of faces using Adaboost. The

predictions made by the algorithm are compared with the ground truth and, on mislabeling, weights are adjusted accordingly. This combination of boosted weak learners results in a reduction to about 6000 features. The resulting model will include features that best classify and differentiate a face from the background. In order to achieve good performance on inference, a degenerate decision tree scheme is applied pooling together features based on their relative importance into stages (38 stages with 1, 10, 25, 25 and 50 features in the first five stages were originally proposed).

For each application of a stage of features on the sample image, a negative outcome results in a stop of computation on that particular area of the image. Only positive results leads to the cascaded application of additional stages of features to other portions of the image. A positive result (detected face) will have run through all features and all stages successfully. This approach, discarding unpromising regions from calculation, makes the inference faster and more effective. Indeed, variants of this algorithm are being used today on smart phones and digital cameras.

The application of openCV will be discussed in the methodology section.

2.2.2. Covolutional Neural Networks (CNNs)

Besides recognizing faces/ humans in images, the project also calls for the classification of 133 different dog breeds. A frequently used solution to multi-class classification problems involving images are convolutional neural networks (CNNs). The underlying mechanics of those specialized neural network will be discussed briefly.

CNNs for image classification receive images as matrices where the brightness of each pixel is represented as a value from 0 to 255 (for 8 bit images). In case of RGB-color images are used, the matrix will have a third dimension representing red, green and blue intensity levels respectively.

CNNs typically have three main type of layers[8] :

- Convolution layers
- Pooling layers

- Fully-connected (dense) layers

A sample configuration with all three layers present is demonstrated in Figure 6.

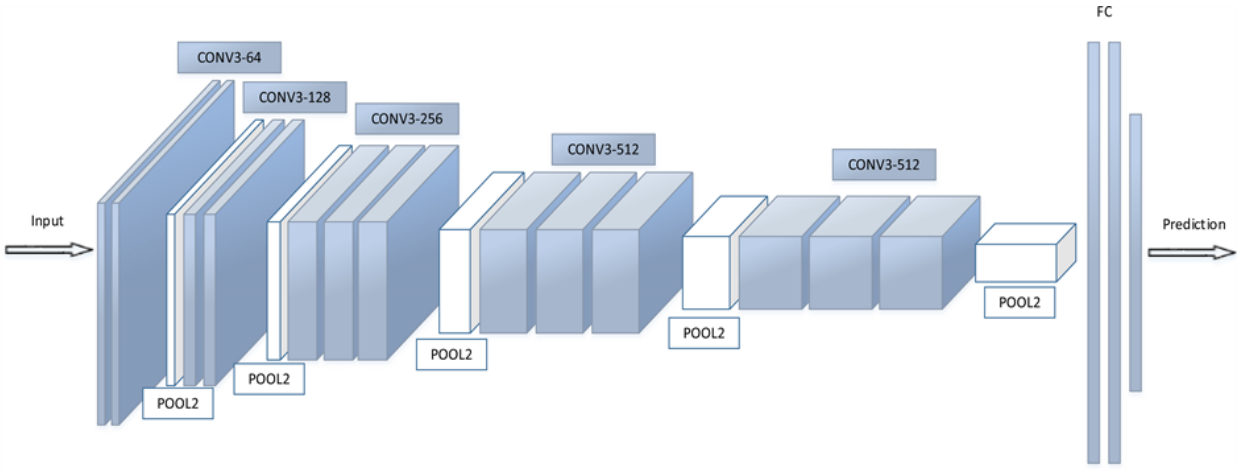


Figure 6: Sample architecture of a CNN[9]

The convolutional layer is the main computation component of the network. In order to operate, the layer will need input data in a form discussed earlier, a filter (or kernel) and a feature map[8]. The kernel represents the feature detector and is usually a 2d array of smaller size then the original image. The kernel is then moved across the image in strides applying the weights. Those weights are initially set randomly and are then consecutively updated by back-propagation. This step is where CNNs learn. Computationally, the dot product is applied between the focused upon area of the image and the kernel matrix, often 3x3 in size. This movement over the image is called convolution which gives CNNs their name. One convolution step is visualized inFigure 7.

The output of this layer is usually fed through an activation function (ReLu or rectified linear activation function is often times used) to a pooing layer. The activation function introduces non-linearity and helps with differentiating and back-propagating derivatives through the network. The pooling layer downsamples and reduces dimensionality, increasing computational efficiency and preventing over-fitting. Possible pooling methods are max pooling where filters keep the maximum value of a matrix whereas avg pooling averages

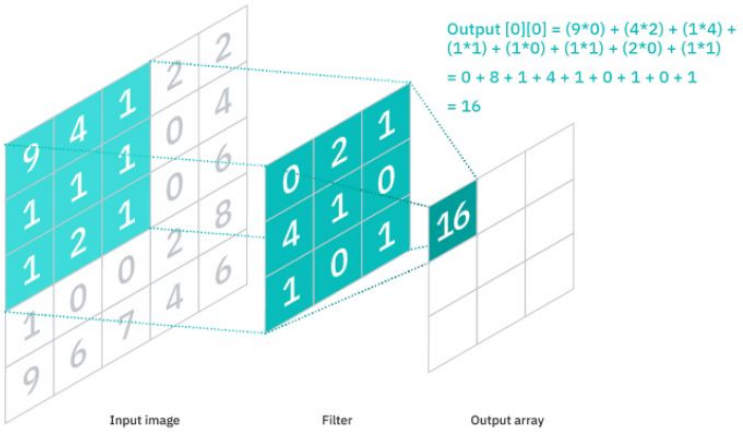


Figure 7: Computation step of a 3x3 kernel on an image [8]

the figures. Both filters output one value per iteration. A max pooling step is visualized inFigure 8.

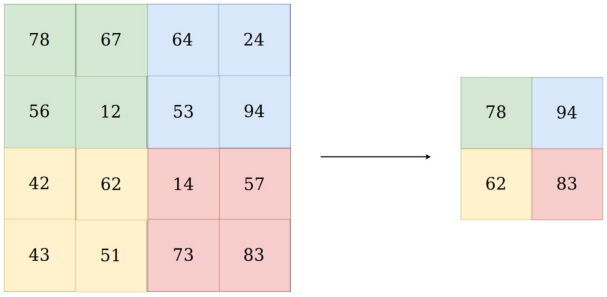


Figure 8: max pooling step reducing dimensionality [10]

Lastly, the feature maps at the output of the convolutional layers are flattened and fed into fully connected layer(s) that are responsible for the classification based on the extracted features created. Softmax activation function is often used as last activation step to convert the values to relative probabilities for each class. For this project, a pre-trained CNN (VGG-16[11]) will be used.

2.3. Benchmarks

The objective of this project is to surpass 10% accuracy (with 133 classes, random baseline would be 0.75% accuracy) on the custom build CNN and 60% accuracy on the transfer

learning approach using VGG-16 as pre-trained CNN. Both objectives will be measured against the classification accuracy.

3. Methodology

3.1. Data preprocessing

No general data preprocessing was performed other than gray-scaling for facial recognition and normalizing, cropping for the input of the CNNs. Also, image augmentation is applied on training. Those steps will be discussed in the implementation section.

3.2. Implementation

3.2.1. Facial recognition

For facial recognition, openCV library is used as discussed. Multiple pre-trained cascade classifiers are available. "haarcascade_frontalface_alt.xml" is used in this instance.

Figure 9 shows the function that outputs "True" if a human face was detected and "False" if not. The steps performed are:

- Read in image
- Convert image to gray scale
- Apply face cascade algorithm
- Return true if more than 0 faces have been detected, else false

```
# returns "True" if face is detected in image stored at img_path
def face_detector(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)
    return len(faces) > 0
```

Figure 9: face detection function

3.2.2. Dog recognition

After implementing an approach to human face detection, a similar methodology is used to identify dogs more generally in images. This does not include classifying dog breeds yet.

The framework used on this project is pytorch[12] which provides all high level functionalities. VGG-16 pre-trained CNN is used on inference mode. VGG-16 includes 118 dog breeds as standard (index 151 - 268) that can be used directly and with no modification. If an image returns an index corresponding to a dog breed, the function will return a positive result.

VGG-16 is loaded and pushed onto GPU (if cuda is available) for faster inference. Two functions are then cascaded to facilitate inference. First, an image processing function is applied. Images are cropped to expected size (224x224 pixels) but keeping the aspect ratio. The image is then normalized as VGG-16 expects from inputs. Means are [0.485, 0.456, 0.406] and standard deviations are [0.229, 0.224, 0.225]. Those pre-processed images are then fed into the prediction function. The image is pushed to GPU and a forward pass through the network is computed. The resulting values are restricted to the highest probability class. The resulting index of this prediction is then returned. If the resulting index corresponds to a previously defined dog breed, the function will return "True". Otherwise, "False" which means that no dog is detected.

3.2.3. Custom CNN architecture

After implementing a general binary classifier using a pre-trained convolutional neural network, the next challenge is to build a custom CNN capable of differentiating 133 different dog breeds.

The network uses pre-processing steps for training, validation and testing data sets. For validation and testing data, images are resized to 255x255 pixels, then cropped to 224x224 pixels and finally normalized with means of [0.485, 0.456, 0.406] and standard deviations of [0.229, 0.224, 0.225] Each value in the list corresponds to one of three color channels. Additionally, for the training data set, image augmentation is used. An augmentation step is meant to add randomness and variability into the training images to avoid over-fitting.

This technique is implemented by using random resizing and cropping as well as horizontal flipping of training images.

The architecture consists of 4 convolutional layers activated by leaky ReLu functions in order to avoid zero gradients (dead nodes) on back-propagation[13]. The kernel size was chosen as 3x3 pixels with a padding of 1 and a stride of 1. The setup was engineered so that there is no change in output shape from each layer to the next. The output channels are set to 6 (six individual filters) for the first layer. The channels are then doubled with every next layer. The convolutional layers are downsampled between each layer by pooling with a kernel of 2. This reduces the shape of the output by a factor of 2 after each layer.

The convolutional layers are followed by a three layer deep fully connected network with an input size of 48*14*14 = 9,408. This size is stepped down from 8,192 to 1,024 and finally to 133 target classes. The layers are also using ReLu activation and additionally, to decrease over-fitting, a dropout with p=0.4. For a condensed view of the architecture, refer to Figure 10.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Conv2d: 1-1                            [-1, 6, 224, 224]         168
├─MaxPool2d: 1-2                         [-1, 6, 112, 112]         --
├─Conv2d: 1-3                            [-1, 12, 112, 112]        660
├─MaxPool2d: 1-4                         [-1, 12, 56, 56]          --
├─Conv2d: 1-5                            [-1, 24, 56, 56]          2,616
├─MaxPool2d: 1-6                         [-1, 24, 28, 28]          --
├─Conv2d: 1-7                            [-1, 48, 28, 28]          10,416
├─MaxPool2d: 1-8                         [-1, 48, 14, 14]          --
├─Linear: 1-9                            [-1, 8192]                77,078,528
├─Dropout: 1-10                          [-1, 8192]                --
├─Linear: 1-11                           [-1, 1024]                8,389,632
├─Dropout: 1-12                          [-1, 1024]                --
├─Linear: 1-13                           [-1, 133]                 136,325
==========================================================================================
Total params: 85,618,345
Trainable params: 85,618,345
Non-trainable params: 0
Total mult-adds (M): 118.11
==========================================================================================
Input size (MB): 0.57
Forward/backward pass size (MB): 4.38
Params size (MB): 326.61
Estimated Total Size (MB): 331.56
==========================================================================================
```

Figure 10: custom CNN architecure

The criterion by which the output is judged is cross entropy loss. Cross-entropy will calculate a score that summarizes the average difference between the actual and predicted probability distributions for all classes.The score is minimized with a perfect cross-entropy value being 0[14]. ADAM is used as optimizer for its good overall performance implementing momentum especially on computer vision problems[15].

3.2.4. Transfer learning

The custom CNN can be improved by using a large, pre-trained network specialized in image classification. VGG-16 was used in this case as mentioned. Transfer learning is the application of a pre-trained network with an individual adaption of the classifier segment of the network. In this case, VGG-16 has been trained with 1000 different classes in mind. In this project, 133 different classes are being classified. VGG-16 fully connected network consists of 3 layers. Only the last layer is replaced by a custom layer with 4096 input size and 133 classes as output. As cross entropy loss is used as loss function, no softmax function is used on the output layer.

All parameters of the completed network are frozen except the output classifier. The network will use all pre-trained weights of the convolution part of the network without updating them. The loss will only back-propagate through the classifier.Figure 11 shows the final completed architecture.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [-1, 512, 7, 7]           --
|    └─Conv2d: 2-1                       [-1, 64, 224, 224]        (1,792)
|    └─ReLU: 2-2                         [-1, 64, 224, 224]        --
|    └─Conv2d: 2-3                       [-1, 64, 224, 224]        (36,928)
|    └─ReLU: 2-4                         [-1, 64, 224, 224]        --
|    └─MaxPool2d: 2-5                    [-1, 64, 112, 112]        --
|    └─Conv2d: 2-6                       [-1, 128, 112, 112]       (73,856)
|    └─ReLU: 2-7                         [-1, 128, 112, 112]       --
|    └─Conv2d: 2-8                       [-1, 128, 112, 112]       (147,584)
|    └─ReLU: 2-9                         [-1, 128, 112, 112]       --
|    └─MaxPool2d: 2-10                   [-1, 128, 56, 56]         --
|    └─Conv2d: 2-11                      [-1, 256, 56, 56]         (295,168)
|    └─ReLU: 2-12                        [-1, 256, 56, 56]         --
|    └─Conv2d: 2-13                      [-1, 256, 56, 56]         (590,080)
|    └─ReLU: 2-14                        [-1, 256, 56, 56]         --
|    └─Conv2d: 2-15                      [-1, 256, 56, 56]         (590,080)
|    └─ReLU: 2-16                        [-1, 256, 56, 56]         --
|    └─MaxPool2d: 2-17                   [-1, 256, 28, 28]         --
|    └─Conv2d: 2-18                      [-1, 512, 28, 28]         (1,180,160)
|    └─ReLU: 2-19                        [-1, 512, 28, 28]         --
|    └─Conv2d: 2-20                      [-1, 512, 28, 28]         (2,359,808)
|    └─ReLU: 2-21                        [-1, 512, 28, 28]         --
|    └─Conv2d: 2-22                      [-1, 512, 28, 28]         (2,359,808)
|    └─ReLU: 2-23                        [-1, 512, 28, 28]         --
|    └─MaxPool2d: 2-24                   [-1, 512, 14, 14]         --
|    └─Conv2d: 2-25                      [-1, 512, 14, 14]         (2,359,808)
|    └─ReLU: 2-26                        [-1, 512, 14, 14]         --
|    └─Conv2d: 2-27                      [-1, 512, 14, 14]         (2,359,808)
|    └─ReLU: 2-28                        [-1, 512, 14, 14]         --
|    └─Conv2d: 2-29                      [-1, 512, 14, 14]         (2,359,808)
|    └─ReLU: 2-30                        [-1, 512, 14, 14]         --
|    └─MaxPool2d: 2-31                   [-1, 512, 7, 7]           --
├─AdaptiveAvgPool2d: 1-2                 [-1, 512, 7, 7]           --
├─Sequential: 1-3                        [-1, 133]                 --
|    └─Linear: 2-32                      [-1, 4096]                102,764,544
|    └─ReLU: 2-33                        [-1, 4096]                --
|    └─Dropout: 2-34                     [-1, 4096]                --
|    └─Linear: 2-35                      [-1, 4096]                16,781,312
|    └─ReLU: 2-36                        [-1, 4096]                --
|    └─Dropout: 2-37                     [-1, 4096]                --
|    └─Linear: 2-38                      [-1, 133]                 544,901
==========================================================================================
Total params: 134,805,445
Trainable params: 120,090,757
Non-trainable params: 14,714,688
Total mult-adds (G): 15.60
==========================================================================================
Input size (MB): 0.57
Forward/backward pass size (MB): 103.42
Params size (MB): 514.24
Estimated Total Size (MB): 618.24
==========================================================================================
```

Figure 11: transfere learning architecture

3.2.5. Completed algorithm

Finally, all components mentioned are combined to achieve the desired result. The steps necessary are outlined as follows.

If a **dog** is detected in the image, return the predicted breed:

- Preprocess the image
- Run image through dog detector using VGG-16 pre-trained network
- If dog is detected, run the image through the classifier and return the detected breed

If a **human** is detected in the image, return the resembling dog breed:

- If no dog was detected, run image through face detector
- If human face is detected, run image through dog breed classifier
- Return dog breed resembling human face

If neither a dog nor a human face is detected, return error message. Please refer to Figure 12 for implementation details.

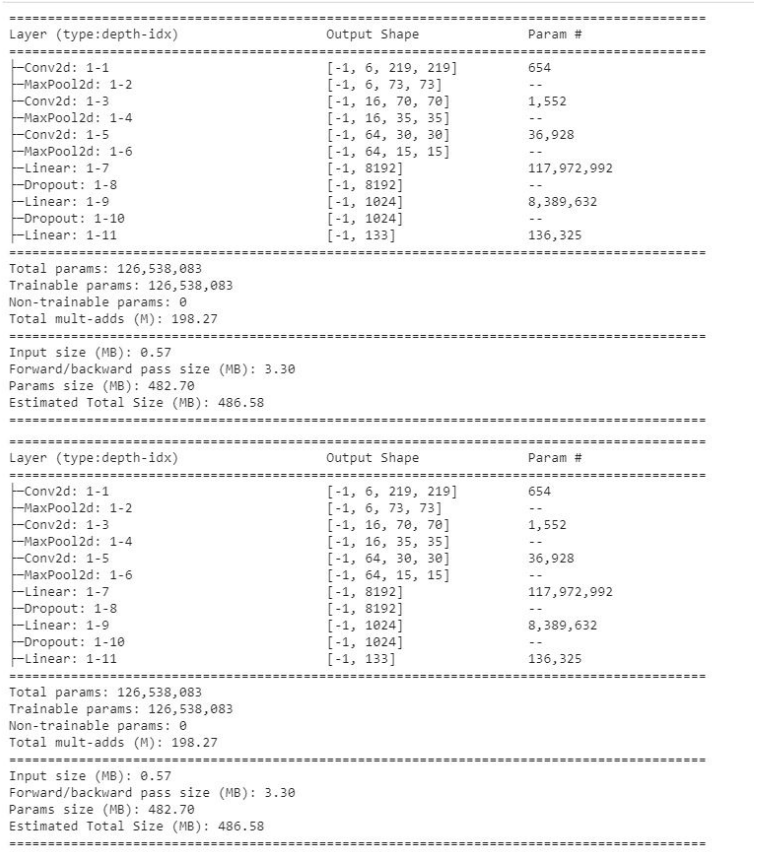
```
def run_app(img_path):
    ## handle cases for a human face, dog, and neither
    # first, run image through standard VGG16 to detect dog vs human
    img = Image.open(img_path)
    plt.imshow(img)
    plt.show()
    if dog_detector(img_path) == True:
        print('Detected a dog from file: {}'.format(img_path))
        # run through dog breed classification
        dog_breed = predict_breed_transfer(img_path)
        print('The breed seems to be: {}'.format(dog_breed))
    else:
        # if no dog is detected, then run it through the face algorithm
        if face_detector(img_path) == True:
            human_dog_breed = predict_breed_transfer(img_path)
            print('Detected a human face! You look like a {}!'.format(human_dog_breed))
        else:
            print('No human or dog was detected in the image!')
```

Figure 12: run_app function

3.3. Refinement

3.3.1. Three layer vs. four layer custom CNN

The custom CNN was also implemented with a three layer convolutional network as displayed inFigure 13.



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Conv2d: 1-1                            [-1, 6, 219, 219]         654
├─MaxPool2d: 1-2                         [-1, 6, 73, 73]           --
├─Conv2d: 1-3                            [-1, 16, 70, 70]          1,552
├─MaxPool2d: 1-4                         [-1, 16, 35, 35]          --
├─Conv2d: 1-5                            [-1, 64, 30, 30]          36,928
├─MaxPool2d: 1-6                         [-1, 64, 15, 15]          --
├─Linear: 1-7                            [-1, 8192]                117,972,992
├─Dropout: 1-8                           [-1, 8192]                --
├─Linear: 1-9                            [-1, 1024]                8,389,632
├─Dropout: 1-10                          [-1, 1024]                --
├─Linear: 1-11                           [-1, 133]                 136,325
==========================================================================================
Total params: 126,538,083
Trainable params: 126,538,083
Non-trainable params: 0
Total mult-adds (M): 198.27
==========================================================================================
Input size (MB): 0.57
Forward/backward pass size (MB): 3.30
Params size (MB): 482.70
Estimated Total Size (MB): 486.58
==========================================================================================
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Conv2d: 1-1                            [-1, 6, 219, 219]         654
├─MaxPool2d: 1-2                         [-1, 6, 73, 73]           --
├─Conv2d: 1-3                            [-1, 16, 70, 70]          1,552
├─MaxPool2d: 1-4                         [-1, 16, 35, 35]          --
├─Conv2d: 1-5                            [-1, 64, 30, 30]          36,928
├─MaxPool2d: 1-6                         [-1, 64, 15, 15]          --
├─Linear: 1-7                            [-1, 8192]                117,972,992
├─Dropout: 1-8                           [-1, 8192]                --
├─Linear: 1-9                            [-1, 1024]                8,389,632
├─Dropout: 1-10                          [-1, 1024]                --
├─Linear: 1-11                           [-1, 133]                 136,325
==========================================================================================
Total params: 126,538,083
Trainable params: 126,538,083
Non-trainable params: 0
Total mult-adds (M): 198.27
==========================================================================================
Input size (MB): 0.57
Forward/backward pass size (MB): 3.30
Params size (MB): 482.70
Estimated Total Size (MB): 486.58
==========================================================================================
```

Figure 13: 3 layer CNN

On training, this layout showed signs of over-fitting as training and validation loss diverged at about 60 epochs. The forth layer in the final solution helps with this problem. Figures 14 and 15 show this effect. The validation loss tracks training loss closly and could have been propably decreased even further with additional epochs of training.

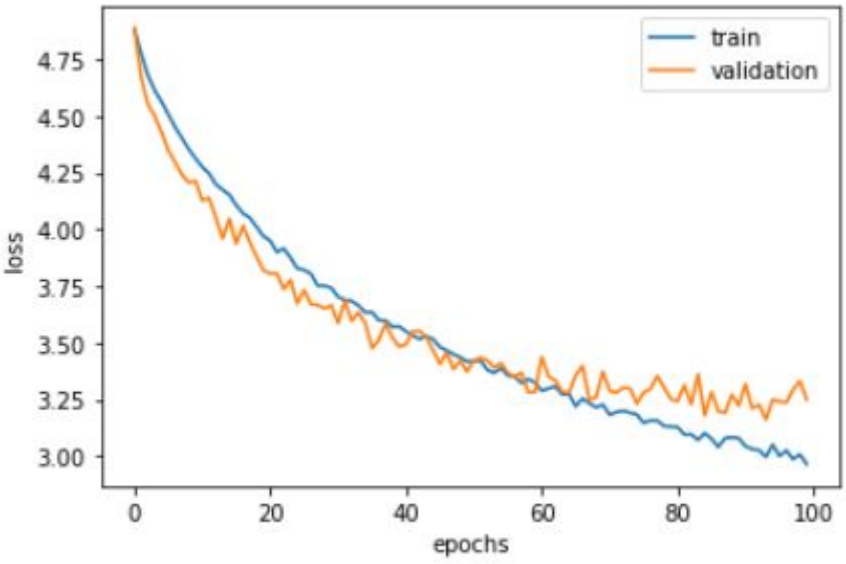


Figure 14: 3 layer CNN loss vs. epochs

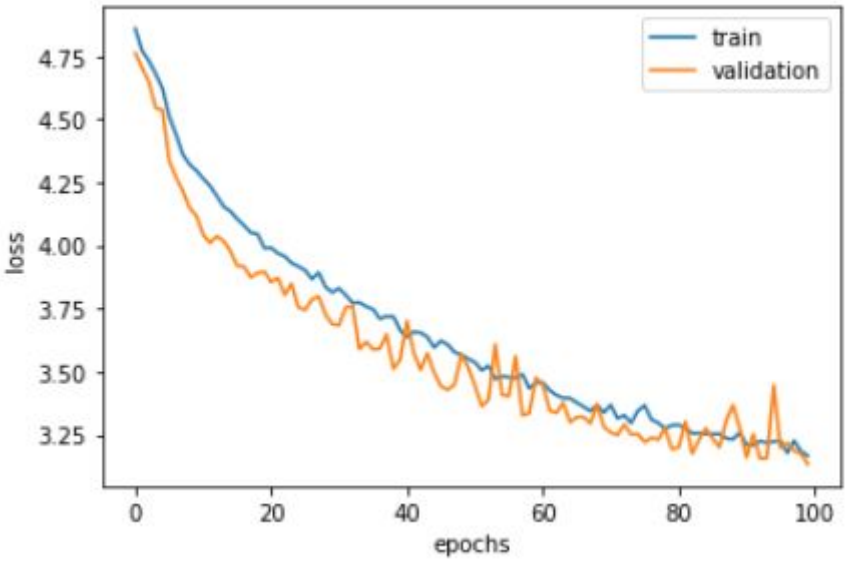


Figure 15: 4 layer CNN loss vs. epochs

4. Results

4.1. Model evaluation and validation

All relevant components of this project include metrics representing their performance.

Face detection:

- 98.7 % of human faces where successfully detected
- 10.97% of dog images where falsely detected as being human.

Performance of dog detector:

- 99.1% of dogs images are correctly classified
- 0.6% of human faces were falsely classified as dogs

Classification on custom CNN:

- The classification accuracy on 133 classes is 25% (209 out of 836)

Classification on transfer learned CNN

- Classification accuracy on VGG-17 pre-trained network with custom classifier is 87%(731 out of 836)

Validation loss drops quickly and stays about the same converging at around 50 epochsFigure 16.

4.2. Validation of final algorithm

The combination of all before mentioned components are now tested on images not included in the original data sets. All images were classified correctly. A validation image with no human or dog present is also used as a control. Refer to Figure 17 for a sample output.

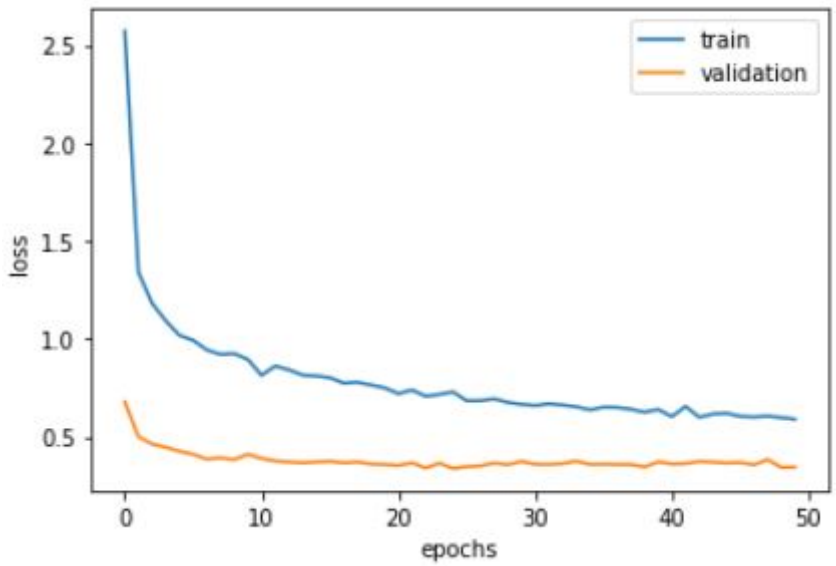


Figure 16: training vs. validation loss for transfere learning

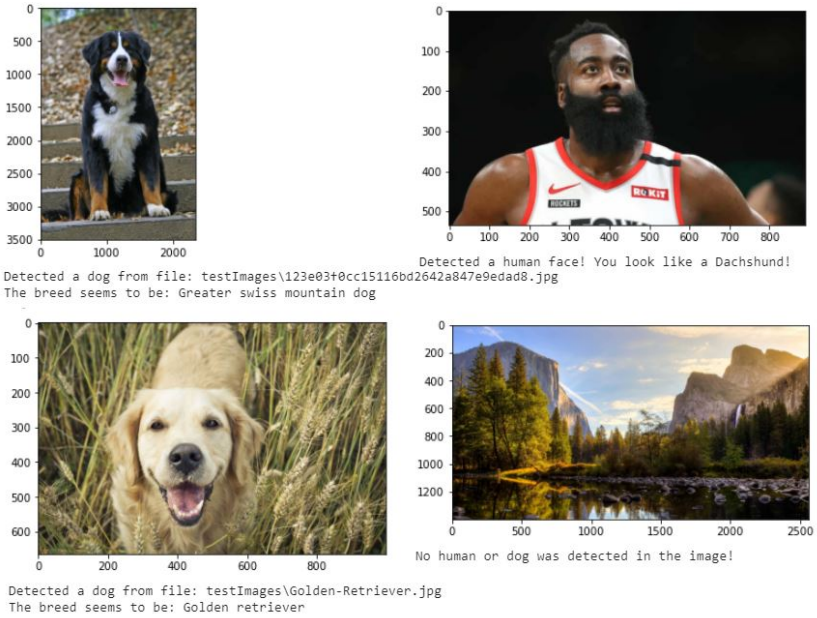


Figure 17: sample output

4.3. Justification

The performance of the custom designed CNN should surpass 10%. The approach discussed achieves this goal with an accuracy of 25%.

The transfer learning approach is meant to surpass 60% accuracy. The described implementation achieves 87% accuracy.

Both benchmarks could be met.

References

[1] J. V. Chamary, How Face ID Works On iPhone X (09 2017).
URL https://www.forbes.com/sites/jvchamary/2017/09/16/how-face-id-works-apple-iphone-x/

[2] A. Lai, How do Self-Driving Cars See? (12 2018).
URL https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503

[3] W. F. Cueva, F. Muñoz, G. Vásquez, G. Delgado, Detection of skin cancer "Melanoma" through computer vision, in: 2017 IEEE XXIV International Conference on Electronics, Electrical Engineering and Computing (INTERCON), 2017, pp. 1–4. doi:10.1109/INTERCON.2017.8079674.

[4] E. A. OSTRANDER, Genetics and the Shape of Dogs.
URL https://www.americanscientist.org/article/genetics-and-the-shape-of-dogs

[5] B. Shmueli, Multi-Class Metrics Made Simple, Part I: Precision and Recall (07 2019).
URL https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2

[6] Cascade Classifier Training.
URL https://docs.opencv.org/master/db/d28/tutorial_cascade_classifier.html

[7] P. Viola, M. Jones, Robust Real-Time Face Detection, International Journal of Computer Vision 57 (2004) 137–154. doi:10.1023/B:VISI.0000013087.49260.fb.

[8] I. C. Education, Convolutional neural networks (10 2020).
URL https://www.ibm.com/cloud/learn/convolutional-neural-networks

[9] H. E. Khiyari, H. Wechsler, Face Recognition across Time Lapse Using Convolutional Neural Networks, Journal of Information Security 07 (03) (2016) 141–151. doi:10.4236/jis.2016.73010.
URL https://dx.doi.org/10.4236/jis.2016.73010

[10] Arunava, convolutional neural network (12 2019).
URL https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05

[11] K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition (2015).

[12] Pytorch.
URL https://pytorch.org/

[13] H. S, Activation Functions : Sigmoid, ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep Learning (01 2019).
URL https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e

[14] J. Brownlee, How to Choose Loss Functions When Training Deep Learning Neural Networks (01 2019).
URL https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

[15] S. Ruder, An overview of gradient descent optimization algorithms (2017).