Gustavo Grinsteins
Date Submitted: 08/05/2022

Dog Breed Classifier Algorithm

I. Definition

Project Overview: Computer Vision and Convolutional Neural Networks

Humans can easily navigate and understand the three-dimensional (3D) world. We can recognize shape, colors, translucency, and other object features. After this recognition, we can combine this information instantaneously to classify a plethora of objects and people. The field of study that aims to simulates this human ability using computers is called Computer Vision.

Computer vision first started in the 1970s with universities attempting to add a visual input to machines in order to achieve intelligent behavior. These tasks proved to be highly complex and let to the development of the technological infrastructures in the following years. As a result, computer vision has made great strides in the last two decades. 3D models of an environment captured by several overlapping images can be reconstructed and algorithms can delineate people and objects in photographs (Szeliski, 2022). However, toddlers have an easier time explaining images in detail than computers. Computer vision is a challenging task because it seeks to recover the rich complexity of the visual world in just one image.

The dramatic increase in computational power during the last decades led to the implementation of Convolutional Neural Networks (CNNs) as the algorithm of choice for image recognition and semantic segmentation tasks (Szeliski, 2022). CNNs eliminate the need for manual feature extraction, the features are learned directly by the CNN. Furthermore, CNNs can be retrained for new recognition tasks, enabling transfer learning (MathWorks, 2022). In recent years, focus has been shifted to modify CNN architectures to improve image classification accuracy. These improvements have been showcased in competitions like the ImageNet Large-Scale Visual

Recognition Challenge and have resulted in impressive image recognition accuracy scores (Simonyan and Zisserman 2015).

Problem Statement: Dog Breed Classification

This project seeks to create an algorithm that correctly labels an image of a dog with the corresponding breed. If the algorithm is supplied an image of a human, the code will identify the image as a human and proceed to provide a resembling dog breed. The project will use publicly available datasets for reproducibility.

Solution Statement

The project solution consists of creating two CNNs: one from scratch and one using transfer learning. The CNNs need to achieve a minimum of 10% (from scratch) and 60% (transfer learning) accuracy in recognizing images from the dog data set. The higher accuracy CNN will be used to implement the algorithm for classifying a dog breed from an image input.

Datasets and Inputs: Dog and Human image datasets

The project will make use of two datasets provided by Udacity:

- Dog Images (DogImages.zip)
    - Total of 13,233 labeled images
    - Image folders partitioned for training, validating, and testing
        - Training set: 6,680 labeled images
        - Validation set: 835 labeled images
        - Testing set: 836 labeled images

- Labeled Faces in the Wild (lfw.zip)
    - Total of 8,351 labeled images

The input to the model will consists of the processed images according to the CNN architecture that is used.

Evaluation Metrics: Image Classification Accuracy

Image classification accuracy will be used as the metric to evaluate the algorithms. The algorithms predicted label will be compared to the ground truth label in the Dog image dataset. The test algorithm will sum the number of correct predictions in the testing partition and divide this over the total partition size to obtain accuracy results.

II. Analysis

Data Exploration

Data exploration for computer vision tasks can be challenging and less straight forward than other machine learning research areas. Is easy to treat computer vision models as “black boxes” and omit understanding of how the model process data. Furthermore, there is no standard software for image dataset exploration (Cieślik, 2022). It is crucial to asses the quality of the data and understand how it is structure in order to properly process it during the analysis.

First to asess the general data quality a small subset of the images are visualized to make sure that there are no strange artifacts or corruptions (Cieślik, 2022):

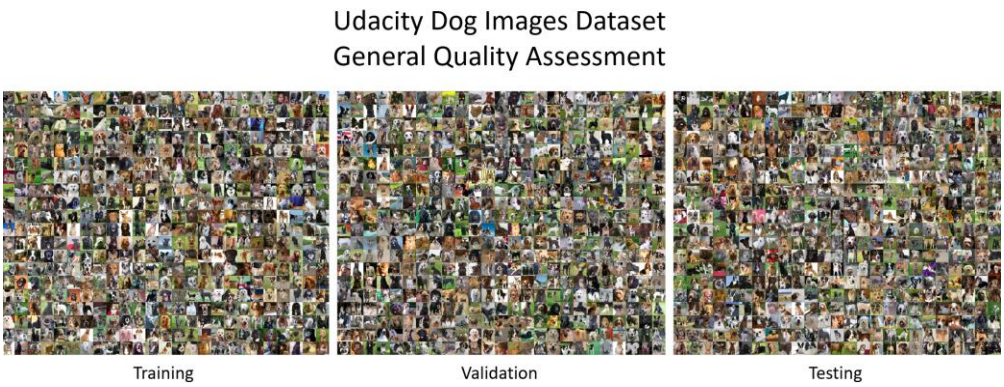
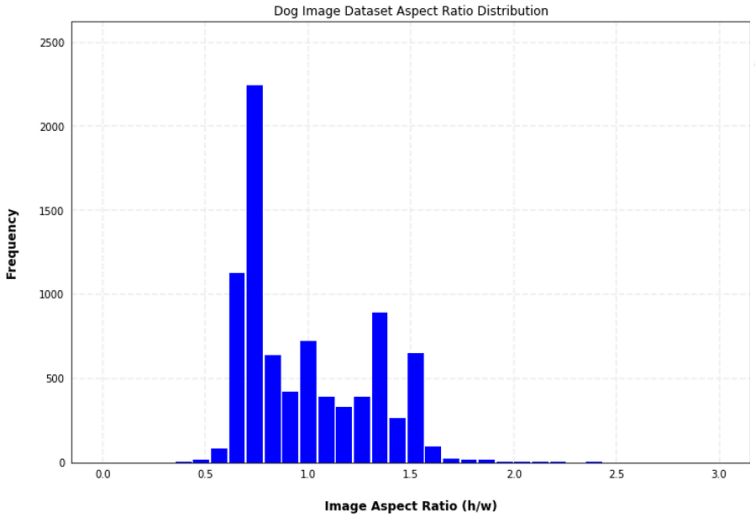


Figure 1- Dog Image Dataset General Quality Assessment

Figure 1 shows 500 images per dataset partition. This sample demonstrated there are no strange artifacts or corrupted images, and all images contain a dog in it. However, some dog images contain humans and/or text within them. The model will need to have the ability to distinguish these features in the images for accurate classification.

An image Aspect Ratio (AR) is a unitless descriptor calculated by dividing the image height by its width. The distribution of the AR will determine what type of processing should be done with the images. For example, it is recommended to perform non-destructing (i.e. not changing the AR) for datasets that images with different aspect ratios. One can maintain the aspect ratio by adding padding to the image. Furthermore, some deep learning image classification models might have hyper parameters dependent on ARs (Cieślik, 2022). The dog image dataset has the following AR distribution:

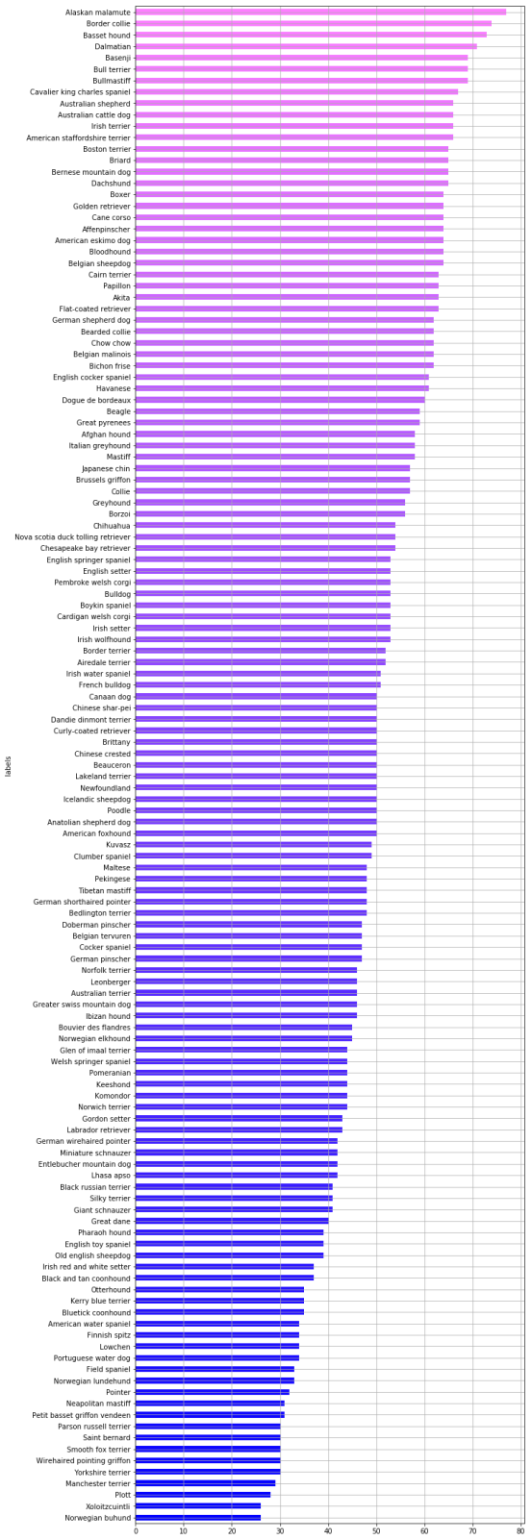


Aspect Ratio Frequencies (h/w)	
count	8351.00
mean	0.99
std	0.30
min	0.35
25%	0.75
50%	0.89
75%	1.27
max	2.52

Figure 2 - AR distribution for dog image dataset

Figure 2 demonstrates that the distribution is skew to the left. A good portion of the images have ARs in the range of 0.7 – 1.5 and the mean lies close to 1. This would indicate that non-destructive resizing is recommended for image processing given the variability of the images ARs.

For most image datasets there is a strong relation between an image and its label. Therefore, one needs to be careful that the processing done to the images does not affect this mapping (Cieślik, 2022). The Dog image dataset provides a folder structure that makes it simple to map images to their labels. It is important to determine if these labels are evenly distributed or if there are any imbalances. Label imbalances can skew the learning for the model and create conflicting signals that will degrade the performance of the model. Countermeasure for class imbalances include adding modifying the weights to certain labels and preprocessing the data to reduce the label imbalance (Cieślik, 2022). The label distribution for the dog image dataset is as follows:



	Label Frequency
count	133.00
mean	50.23
std	11.86
min	26.00
25%	42.00
50%	50.00
75%	61.00
max	77.00

Figure 3 - Dog image dataset label frequencies

There are a total of 133 labels (that is 133 dog breeds in the dataset). The label with most images is Alaskan Malamute with a total of 77 images. The labels with the least images are for Norwegian buhund and Xoloitzcuintli with a total of 26 images each. We can see that most labels have about 50 images associated with them.

Algorithms and Techniques

The workflow of this project can be separated into the following sections: dataset loading, image processing, development of CNNs, and final dog classification algorithm testing. The dataset loading will consist of loading the datasets making sure to take advantage of the conveniently labeled and partitioned (training, validating, and testing) images.

Following proper loading, the images will be preprocessed and processed using Pytorch (Torchvision), Open Computer Vision (OpenCV), and Python Imaging Library (PIL). Pytorch will be the backbone of the algorithm. This module will be used to create the VGG-Nets CNNs and properly transform and load the image dataset into the models (VGG is short for the Visual Geometry Group at the University of Oxford Department of Engineering Sciences). VGG-NETs were chosen given that they have impressive performances on the large image dataset ImageNet which contains images with 1000 possible classification labels (Simonyan and Zisserman 2015). OpenCV will be used to add the capability to detect humans in the images through a pretrained Haar feature-based cascade classifier (Viola & Jones, 2001). PIL will be used for general processing of the images. Data exploration is performed during this step since these modules (with the addition of Numpy, Pandas, and Matplotlib) allow for easy extraction and visualization of image features.

After the data processing and exploration is complete, a CNN will be developed from scratch. The architecture for this CNN will resemble a VGG-11 (Simonyan and Zisserman 2015). Since this model is a simpler architecture in relation to other VGG-Nets it will reduce the training time. Following training, this CNN will be tested using the testing dataset and accuracy results will be outputted (project requirement calls for > 10% accuracy).

The low accuracy from the CNN created from scratch is due to the small dataset (for CNN training standards), lack of computing power, and time. The accuracy score of the model will be increased by implementing transfer learning on a pre-trained VGG-19 CNN (Simonyan and Zisserman 2015). This CNN is deeper than other VGG-Nets and has the advantage of recognizing a wide range of features from images given its deep network architecture. The transfer learning will consist of substituting the last classification layer with a layer that outputs the right number of label predictions from the provided dataset. Training will only happen on the classification portion of the deep network. The feature detection network weights will be kept intact to maintain this knowledge from the extensive ImageNet training. Following training, the transfer learning CNN will be tested on the testing partition and accuracy results will be outputted (project requirement calls for > 60% accuracy).

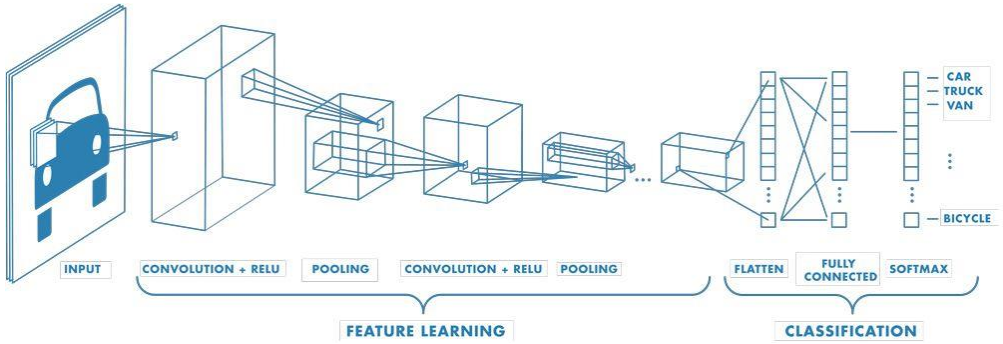


Figure 4 - Example of a network with many convolutional layers. Filters are applied to each training image at different resolutions, and the output of each convolved image is used as the input to the next layer. (MathWorks, 2022)

The classification algorithm will be employed using the transfer learning VGG-19 CNN in order to provide user-friendly feedback regarding the classification of the provided dog images. A smaller subset of images will be used to test the robustness of the created model and the algorithm logic. If a human image is fed into the algorithm, it will detect the human, provide an approximate dog breed, and include a sample image of the dog from the dog image dataset.

Benchmark Model: VGG-Nets

The benchmark model used for the project will be VGG-Nets architectures (Simonyan and Zisserman 2015). These are state-of-the-art architectures that have won top prizes in image

recognition competitions. Simonyan and Zisserman have demonstrated that this architecture generalizes well using other datasets. Furthermore, VGG-Nets are publicly available to facilitate further research.

III. Methodology

Data Preprocessing

The preprocessing of the images is done according to the VGG-Nets input specification (Simonyan and Zisserman 2015). According to the Pytorch documentation on VGG-Nets:

"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 must be loaded into a range of [0, 1] and then normalized using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225]."

(PyTorch VGG-NETS, 2022)

In the data exploration section, it was noted that the different sizes for the images calls for non-destructive resizing. However, given that VGG-Nets take in a specific aspect ratio for input this recommendation will not be followed.

This image pre-processing is applied to train, validation, and testing loaders. The training set data will go through image augmentation since CNNs benefit from large amounts of data to learn properly. This will add randomness to the existing data and help avoid overfitting. The images are cropped, flipped, and gray scaled. Examples of this image augmentation are depicted in **figure 5.**

Figure 5 - Training set image augmentation

Implementation

VGG-Nets CNNs were used for the implementation of the dog classification algorithm. The convolutional layers oversee transforming the input into the layer through a convolving filter with a given stride through the image's pixel data. Each convolution stride performs a mathematical operation between the filter and pixel data values. The result of this orchestration is pattern recognition that becomes more nuanced in the deeper layers of the network.
The convolutional layers use a Rectified Linear Unity (ReLU) activation function, this activation function is commonly used due to its simplicity and fast execution which is advantageous given the heavy processing created by the convolutions.

The Max pooling layers reduces the dimensionality of the data by shrinking the number of pixels in the output from a previous convolutional layer. Max pooling layers behave similarly to

convolutional layers except the mathematical operation performed extracts the maximum values from the pixel data within the filter size. These set of convolutional/max pooling layers oversee recognizing the features within images.

At the end of the VGG-Net there are three fully connected linear layers. These layers form an artificial neural network that take in a one-dimensional vector which itself comes from the "flatten" output of the last convolutional/max pooling layer. These linear layers hold the variable weight parameters that are updated during training. A dropout regularization is applied to some of these layers in order to prevent the model from overfitting. The last layers oversee classifying the recognized features into the label categories we are interested.

A SoftMax activation function is used at the end of this architecture. The SoftMax layer takes the outputs from the last fully connected layer and converts them into an array of probabilities assigned to each one of the labels within the output layer. This facilitates selecting the output with the highest probability and assigning that as the label for the original input into the CNN.

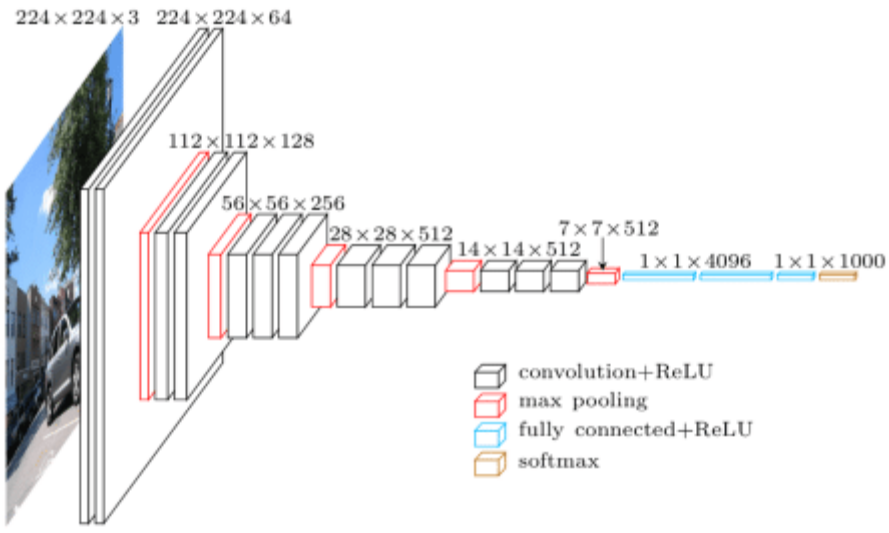


Figure 6 - VGG-16 Network Architecture (Smola, 2019)

Figure 6 visualizes the architecture of VGG-16. Observe the arrangement of convolutional layers paired up with Max pooling layers. At the end of the architecture, we see the linear layers ending with a soft max layer activation function applied to the last output. Lastly, note that the initial input for VGG-Nets is a 244x244 image with the value channel for red, blue, and green color numeric

data (RGB). The final output is a one-dimensional array of 1000 values corresponding to all the possible classifications for the ImageNet Images.

The VGG-Nets defined below do not explicitly have a SoftMax layer at the end. This activation function is applied in the background through the given loss function (VGG Output Layer - no softmax?, 2017).

From Scratch: VGG-11

The VGG-11 network created from scratch follows the architecture depicted in **table 1:**

Table 1 - VGG-11 Architecture (Simonyan and Zisserman 2015)

VGG-11 Architecture
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 3, Output Size: 64 (ReLU)
Max Pooling (Filter 2x2, Stride: 2)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 64, Output Size: 128 (ReLU)
Max Pooling (Filter 2x2, Stride: 2)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 128, Output Size: 256 (ReLU)
Max Pooling (Filter 2x2, Stride: 2)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 256, Output Size: 512 (ReLU)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 512, Output Size: 512 (ReLU)
Max Pooling (Filter 2x2, Stride: 2)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 512, Output Size: 512 (ReLU)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 512, Output Size: 512 (ReLU)
Max Pooling (Filter 2x2, Stride: 2)
Fully Connected Linear Layer - Input Size: 25,088, Output Size: 4096 (ReLU, Dropout: 0.3)
Fully Connected Linear Layer - Input Size: 4096, Output Size: 4096 (ReLU, Dropout: 0.3)
Fully Connected Linear Layer - Input Size: 4096, Output Size: 133 (Dropout: 0.3)

Cross entropy loss was selected as the loss function with the Adam optimizer as the optimizer. The training was performed using 20 epochs with batch sizes of 32 images. The training took around 60 minutes using the Udacity GPU enabled workspace.

Transfer Learning: VGG-19

The VGG-19 network created using transfer learning follows the architecture depicted in **table 2:**

VGG-19 Architecture
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 3, Output Size: 64 (ReLU)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 64, Output Size: 64 (ReLU)
Max Pooling (Filter 2x2, Stride: 2)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 64, Output Size: 128 (ReLU)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 128, Output Size: 128 (ReLU)
Max Pooling (Filter 2x2, Stride: 2)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 128, Output Size: 256 (ReLU)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 256, Output Size: 256 (ReLU)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 256, Output Size: 256 (ReLU)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 256, Output Size: 256 (ReLU)
Max Pooling (Filter 2x2, Stride: 2)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 256, Output Size: 512 (ReLU)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 512, Output Size: 512 (ReLU)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 512, Output Size: 512 (ReLU)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 512, Output Size: 512 (ReLU)
Max Pooling (Filter 2x2, Stride: 2)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 512, Output Size: 512 (ReLU)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 512, Output Size: 512 (ReLU)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 512, Output Size: 512 (ReLU)
Convolution Layer (Filter 3x3, Stride: 1) – Input Size: 512, Output Size: 512 (ReLU)

Max Pooling (Filter 2x2, Stride: 2)
Fully Connected Linear Layer - Input Size: 25,088, Output Size: 4096 (ReLU, Dropout: 0.3)
Fully Connected Linear Layer - Input Size: 4096, Output Size: 4096 (ReLU, Dropout: 0.3)
Fully Connected Linear Layer - Input Size: 4096, Output Size: 133 (Dropout: 0.3)

Cross entropy loss was selected as the loss function with Stochastic Gradient Descent as the optimizer. The training was performed using 20 epochs with batch sizes of 32 images. The training took around 140 minutes using the Udacity GPU enabled workspace.

Complications during the coding process are detailed in the following section.

Refinement

There were several areas of refinement when training the VGG-Models:

- Selecting an appropriate image batch size
- Trying out different optimizers
    - Selecting an appropriate learning rate
- Selecting an appropriate number of epochs
- Training on a GPU rather than a CPU

Initial selection of these parameters was made arbitrarily. The results were disappointing consisting on plateaued training losses, excessive time to train, poor accuracy results, and workspace running out of memory.

After several rounds of trial and error, the appropriate batch size was set to 32 which provided descent accuracy scores and did not make the workspace run out of memory. The Adam optimizer worked better than the Stochastic Gradient descent for the VGG-11 network providing lower validation loss scores. The learning rates were set in the range of 0.001-0.0001, larger values than these cause the training to stall. The final number of training epochs were set to 20. Numbers

lower than this provided poor accuracy scores and larger epochs did not provide any improvements.

Initially the training was done using a CPU. However, the training times were halved when using the GPU provided in the Udacity workspace.

IV. Results

Model Evaluation and Validation

As mentioned in the project description, image classification accuracy will be used as the metric to evaluate the algorithms. The CNN predicted label is compared to the ground truth label in the Dog image dataset. The test algorithm will sum the number of correct predictions in the testing dataset and divide this over the total partition size to obtain accuracy results.

The accuracy scores for the implemented models are summarized below:

Table 2 - VGG-Nets Accuracy Scores on the Dog Image Test Dataset

Model	Accuracy Score (Requirement)
VGG-11 (from scratch)	13% (> 10%)
VGG-19 (transfer learning)	81% (> 60%)

The transfer learning model outperforms the model trained from scratch. This is because the model using transfer learning has been trained on a large dataset that contains 1.2 million labeled images. The VGG-11 CNN was only trained using 8,000 images. Therefore, the transfer learning model was used to create the algorithm for the dog breed classification from an image.

Both models scored higher than the required capstone project accuracy scores meeting solution expectations. Furthermore, the algorithm was tested using a holdout set containing several images. This set had a mixture of pictures of dogs, humans, and graph images. The algorithm was able to correctly identify the dog breed for the dog images, identify humans in pictures and provide

resembling dog breed and picture, and alert the user if the image does not contain a human or a dog.

Justification

The research study performed by Simonyan and Zisserman boasts mean average precision scores for several image datasets ranging from 85-92%. The VGG-19 model presented in this paper returns an accuracy score of 81% using the dog image dataset which is comparable with the high scores from VGG-Nets trained on other image datasets.

It is important to note that the evaluation metrics we are comparing are not calculated the same way (Accuracy vs. mean average precision). Furthermore, we are comparing performances on different datasets. A good follow up to this project would be calculating mean average precision scores and compare the results from the CNN models (if publicly available) compared in the Simonyan and Zisserman using the dog image dataset.

V. Conclusion

Free-Form Visualization

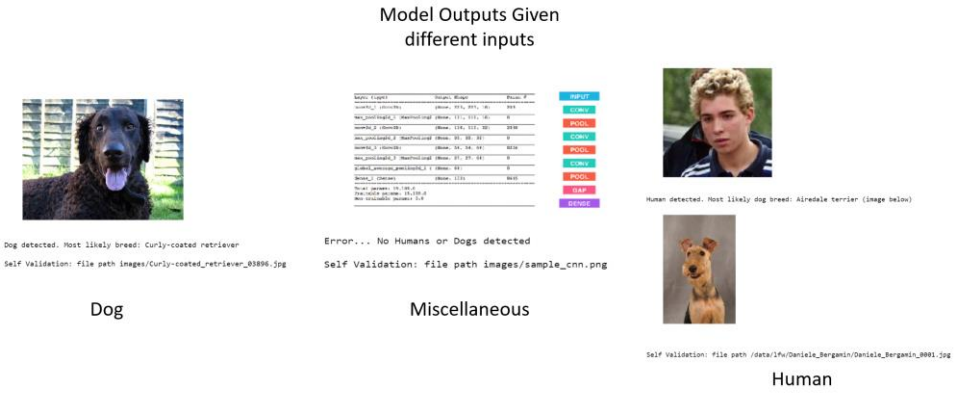


Figure 7 - Input/Output Visualization

Figure 7 shows sample outputs given a dog, miscellaneous, and human input. For a dog image input the algorithm returns the pictures that was inputted classifies it as a dog long its predicted breed. For a miscellaneous input, meaning no human or dog, the algorithms warn the user accordingly. For a human image input, the algorithms classify the image as a human and provides its most likely dog breed along with an image of that dog with that breed. Since some user won't necessarily know what the breeds look like they can have a sample image right on their application, so they don't have to go out and google it.

Reflection

The dog image classification journey started with the provided problem statement and a dataset. As the person implementing a solution to the problem, I had to investigate the given datasets in order to understand how to approach and find a solution. Following this investigation to questions had to be answered: How to detect a human in a picture and how to detect a dog in a picture. To detect humans the algorithm uses pretrained Haar cascade classifier along with some other python modules to facilitate the image processing and logic. To detect dogs in images a pretrained VGG-16 CNN was used which outputted 1000 probability values for the different categories in the ImageNet dataset. A subset of these categories contained dog breed labels; therefore, the algorithm would use this label identification to detect the dog within the image. Following this, the dog detection network was improved by only outputting 133 probability values corresponding to the 133 breed labels originally within the 1000 label category in the ImageNet dataset. A VGG-11 was created from scratch resulting in an accuracy score of 13% which was not good enough for a robust dog classification algorithm. Therefore, transfer learning was used in a VGG-19 model resulting in an 81% accuracy score making the model robust enough for the solution. Following the selection of the right CNN the logic of classifying the image as a dog, human, or none had to be implemented using the algorithms described above. The final solution consists of outputs visualized in **Figure 7**.

An interesting aspect of the project was using the Pytorch library for computer vision. This gave me a real-world application of this open-source software and forced me to read the documentation in order to understand how it is implemented. The project also provided a tangible

experience that explains why GPUs are needed to process the massive amount of information within computer vision algorithms. Additionally, going through the training process made me realize that this step takes time, and you need to approach it effectively, so time is not wasted training an underperforming model.

A difficult portion of the project was trying to understand why the validation loses during the training were plateauing and not decreasing as they should. This trial-and-error process was frustrating given the fact that you needed to run the training through a couple of epochs (10-20 min of waiting time) to realize your set up is not working. This forced me to dive deeper into the matter and really understand what parameters need to be tuned to improve the training.

The final model and solution meet the expectation for the problem. I believe that the model as it is currently standing can be turned into a fun phone application people can use to see what dog breed they look like or test the dog classification using their own dogs.

Improvement

The following ideas could improve the model accuracy score and make the algorithm more general:

- During the data exploration it was found that the aspect ratio for the dog images was skewed and not uniform and that there was slight label imbalance. Implementing solutions discussed in the data exploration section and comparing their implementation by analyzing differences in the accuracy score could improve the project.

- Explore more human detection algorithms that could help identify humans in more complex images.

- Calculate mean average precision scores and contrast the results from the CNN models mentioned in the Simonyan and Zisserman journal article using the dog image dataset.

- The model only has 133 dog breed labels when there are 200 dog breeds recognized by the American Kennel Club (Breeds by Year Recognized, 2022). Expanding the original dataset to include images of the missing breeds would improve the algorithm.

- Performing transfer learning by finetuning some of the feature extraction layers. Especially the deeper layers since they oversee recognizing more unique image features.

- Perform ensemble learning to reduce the variability of the algorithm predictions.

- Use transfer learning on more state-of-the art CNNs and see which one perform the best.

- Implement this algorithm within a phone app or a web application.

Even though the implemented model has high accuracy scores, there is room for improvement. Applying a combination of the ideas mentioned could lead to higher accuracy scores.

References

Breeds by Year Recognized. (2022). Retrieved from American Kennel Club: https://www.akc.org/press-center/articles-resources/facts-and-stats/breeds-year-recognized/#:~:text=The%20AKC%20currently%20registers%20200%20dog%20breeds.

Cieślik, J. (2022, July 21). *How to Do Data Exploration for Image Segmentation and Object Detection*. Retrieved from Neptune.ai: https://neptune.ai/blog/data-exploration-for-image-segmentation-and-object-detection

PyTorch VGG-NETS. (2022). Retrieved from PyTorch.org: https://pytorch.org/hub/pytorch_vision_vgg/

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations.*

Smola, A. (2019). *Introduction to Deep Learning - Lecture 12.* Retrieved from http://courses.d2l.ai/: https://www.youtube.com/watch?v=Nf1BlOR8kVY

Szeliski, R. (2022). *Computer Vision: Algorithms and Applications.* Springer Cham.

Udacity. (2022). *Dog Image Dataset.* Retrieved from https://www.udacity.com/: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip

Udacity. (2022). *Labeled Faces in the Wild Dataset.* Retrieved from https://www.udacity.com/: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip

VGG Output Layer - no softmax? (2017, October). Retrieved from https://discuss.pytorch.org/: https://discuss.pytorch.org/t/vgg-output-layer-no-softmax/9273

Viola, P., & Jones, M. J. (2001). Robust real-time face detection. *International Journal of Computer Vision*, 137–154.

What is a Convolutional Neural Network? 3 things you need to know. (2022). Retrieved from MathWorks: https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html