Machine Learning Capstone Project
Dog Breed Classifier

Stefan Nae

June 2020

1 Definition

1.1 Project Overview

In the field of Computer Vision (CV) the dog breed classification task fits into the fine-grained image classification (FGIC) research area. FGIC covers a wide range of CV problems including but not limited to the classification of vehicle models (e.g. the FGVC-Aircraft dataset [1]), animal species (e.g. the Caltech-UCSD Birds-200-2011 dataset [2]), and plant diseases (e.g. the Plant Pathology 2020 challenge dataset [3]) where the intra-class variance is high while inter-class variance can be both high and low. FGIC-related datasets are relatively small with a limited number of examples per class leading to extra emphasis on model generalization techniques. For this project the Columbia Dog Breed dataset is used [4].

1.2 Problem Statement

This work is targeted at the dog breed classification task as posed by the Columbia Dog Breed dataset [4] characterized by high variance both intra- and inter-class. The task is to identify different dog breeds.

The solution explores the use of convolutional neural networks (CNNs) within both a custom solution (also called scratch) and by transfer learning using a pre-trained model, like ResNet, DenseNet, or Inception, as input feature extractor (encoder) for training a fully-connected classification layer. Furthermore, the implementation distinguishes between humans and dogs and outputs the top-1 dog breed class in either case.

1.3 Evaluation Metrics

Given the classification task at hand, the model performance will be assessed using the accuracy metric on the provided test dataset

$$Accuracy = \frac{Total\ number\ of\ correct\ predictions}{Total\ number\ of\ tested\ cases}. \tag{1}$$

The metric choice is driven by the relatively balanced distribution of images per class in the validation and test sets with a mean of 6.3 images per class and small $\sim$1.5 standard deviation containing most values within 3σ as shown in Figure 1 and summarized in Table 1.

2 Analysis

2.1 Data Exploration

The Columbia Dog Breed dataset [4], containing labels for 133 dog breeds across 8351 images accompanied by eight part labels, together with the Labeled Faces in the Wild dataset [5, 6], packaging images of 5749 people (1680 people with two or more images) for a total of 13233 images of 250 by 250 pixel mostly color with a few grayscale, both with Udacity-provided access through the dedicated Dog Project Workspace constitute the data for the project. The first dataset is used for training and testing the neural network solution for dog breed identification while the second dataset is used in testing the human face detection and input extraction prior to inference using the dog breed classifier on arbitrary inputs for a customized user experience able to distinguish between humans and non-humans.

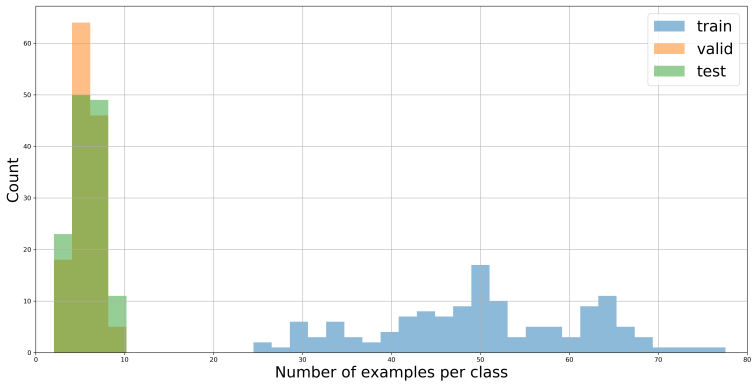


Figure 1: Distributions of per class breed example counts for the provided data split in train, validation, and test sets.

2.2 Exploratory Visualization

The dog dataset shows the characteristics of a fine-grained computer vision task as can be seen through Figures 2 and 3 with reduced variance between the German shepherd dog and the Bedlington terrier while the English setter images show high variance.

	classes	mean	std	min	max
train	133	50.2	11.9	26	77
validation	133	6.3	1.4	4	9
test	133	6.3	1.7	3	10

Table 1: Statistics on the distribution of per class breed example count.

Figure 2: Example sample of different breeds. It shows reduced inter-class variance (similar colors, face, and ears) between the second and fifth image from the left, a German shepherd dog and a Bedlington terrier, and between the last three images from the left (pose similarity, orientation in this case).

Figure 3: Example of high intra-class variance between English setters.

2.3 Algorithms and Techniques

The implementation follows the machine learning life-cycle:

1. explore and process data (load and explore the datasets, identify the dog-breed image transforms useful in augmenting the limited dataset class-sizes – e.g. use random rotations, crops, and horizontal flips –);

2. modeling (train two CNN models – a scratch CNN based architecture and partially-custom CNN architecture based off of a pre-trained model – using the provided train/validation data split for hyper-parameter tuning and early stopping and prepare for generic model evaluation with both dog and human inputs to first identify if there is a human in the picture – using a Haar cascade for both identifying a face and extracting it for model inference –, then submit the image for inference, and provide a

customized display of the breed classification result correlated with the human of non-human character of the input);

3. deployment (create a REST API using Flask).

The main focus is on the modeling step with an emphasis on using the CNN state-of-the-art computer vision algorithm. Convolutions are used for learning feature extraction kernels at different scales. Our goal is to extract micro-scale features characteristic of the fine-grained CV tasks. A typical CNN architecture includes convolutional, pooling, and fully-connected layers [7, 8]. The (convolution, pooling) building block is usually repeated to enhance feature extraction over multiple scales.

The scratch network model implements three of similar CNN building blocks followed by two fully-connected layers while the transfer learning model uses the ResNet architecture. ResNet is a special kind of CNN with state-of-the-art results in computer vision tasks [9]. In particular, the implemetation will use a 50-layer pre-trained ResNet provided by PyTorch [10]. The choice for the ResNet-50 architecture is based on a benchmark showing it as one of the highest top-1 accuracy performers for a lighter model size in comparison with other available pre-trained models [11]. A further analysis could explore the performance comparison with other available pre-trained models including DenseNet-201, DenseNet-161, and Inception-v3.

Transfer learning is a technique in which the pre-trained weights of model are use either for initializing the weights of a custom model derived from the pre-trained model or are kept fixed and used to extract features for the following layers of a custom solution based on the initial pre-trained model [12]. Recent application using transfer learning show that such a pre-trained network can be used as a feature extractor for other computer vision problems, using the learned weights from one dataset to solve a different problem (classification or regression) on loosely related datasets [13].

2.3.1 Benchmark model

The publication of the Dog dataset used in this project was accompanied by an classification algorithm with 67% top-1 breed inference accuracy [4]. The architecture of this benchmark model does not employ the use of neural networks, including convolutional neural network, but breed-specific one-vs-all support vector machines trained on features extracted from detected dog faces, keypoints (e.g. eyes and nose position) and color histograms.

3 Methodology

3.1 Data Pre-processing

The input data requires separate pre-processing steps for the several models used in the implementation which can be separated into two pipelines:

- Haar cascade pipeline
- Neural network pipeline

The pre-processing for the Haar cascade requires a single step of converting the image to grayscale.

On the other hand, the neural network pipeline involves several steps:

1. Standardization (for training/testing/validation/inference) - Mandatory for compatibility with the expected network inputs.
   (a) initial over all dimensions resize to 256 pixel on the smaller edge;
   (b) 244 pixel-per-side input square image crop. This is the input size for the PyTorch pre-trained models used in the solution. It was kept the same for the scratch model in order to control for it in the performance comparison between networks;
   (c) conversion to PyTorch tensor which includes normalization in the $[0.0, 1.0]$ range;
   (d) normalization based on dataset mean and standard deviation over its dimensions, in this case over the three color channels. The pre-trained models have already defined and identical normalization values $mean = [0.485, 0.456, 0.406]$, $std = [0.229, 0.224, 0.225]$ while the calculated values for a selection of the Dogs Dataset are $mean = [0.487, 0.459, 0.394]$, $std = [0.229, 0.226, 0.224]$.
2. Augmentation (for training) - Optional but important because of the limited size of the dataset, though given the small number of training epochs it is not as important as when training for 10+ epochs.
   (a) square crop already augments the data by center crop for validation, testing and inference, nand by random crop for training
   (b) random rotations in a ± 30 degrees range around the center of the image
   (c) random horizontal flips

3.2 Implementation

The models used for the detection of dogs (Pytorch pre-trained VGG16) and humans (OpenCV pre-trained Haar cascade) are used without modification with the properly formatted data resulted from the preprocessing step. The implementation focuses on training the dog-breed classifier for which two solution are explored, a custom network trained from scratch and a pre-trained weights transfer network based architecture.

The scratch network uses three convolutional layers followed by two linear layers including the output layer as shown by the code below.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        layer_size_conv = [48, 48, 96]
        layer_size_lear = [2048]

        ### First layer - convolution 1
        # (3, 224, 224) three channels (RGB) input image
        # Xavier weight initialization
        # 48 5x5 square convolution kernel with stride 2
        # 48 extracted output channels/feature maps
        # (48, 110, 110) output
        # Batch normalization
        self.conv1 = nn.Conv2d(3, layer_size_conv[0],
                               5, stride=2)
        I.xavier_normal_(self.conv1.weight, gain=1.)
        self.norm1 = nn.BatchNorm2d(layer_size_conv[0])

        ### Second layer - convolution 2
        # (48, 110, 110) input
        # Xavier weight initialization
        # 48 3x3 square convolution kernel with stride 2
        # 48 extracted output channels/feature maps
        # (48, 54, 54) output
        # Batch normalization
        self.conv2 = nn.Conv2d(layer_size_conv[0],
                               layer_size_conv[1],
                               3, stride=2)
        I.xavier_normal_(self.conv2.weight, gain=1.)
        self.norm2 = nn.BatchNorm2d(layer_size_conv[1])

        ### Third layer - convolution 3
        # (48, 54, 54) input
        # Xavier weight initialization
        # 48 3x3 square convolution kernel with stride 2
        # 48 extracted output channels/feature maps
        # (96, 26, 26) intermediary output
        # Batch normalization
        # (2, 2) Pooling
        # (96, 13, 13) output
        self.conv3 = nn.Conv2d(layer_size_conv[1],
                               layer_size_conv[2],
                               3, stride=2)
        I.xavier_normal_(self.conv3.weight, gain=1.)
        self.norm3 = nn.BatchNorm2d(layer_size_conv[2])
```

```
        self.pool3 = nn.MaxPool2d(2, 2)

        ### Fourth layer - fully connected 1
        # 96 x 13 x 13 input size
        # Xavier weight initialization
        # Batch normalization
        # 30% dropout
        # 2048 output size
        self.fc1 = nn.Linear(layer_size_conv[2]*13*13,
                             layer_size_lear[0])
        I.xavier_normal_(self.fc1.weight, gain=1.)
        self.nf1 = nn.BatchNorm1d(layer_size_lear[0])
        self.df1 = nn.Dropout(p=0.3)

        ### Fifth/Output layer - fully connected 2
        # 2048 input size
        # 133 output size
        self.output = nn.Linear(layer_size_lear[-1], 133)

    def forward(self, x):
        # ReLU activation for all convolutional layers
        # and the first fully connected layer
        x = F.relu(self.norm1(self.conv1(x)))
        x = F.relu(self.norm2(self.conv2(x)))
        x = self.pool3(F.relu(self.norm3(self.conv3(x))))
        x = x.view(x.size(0), -1)
        x = F.relu(self.df1(self.nf1(self.fc1(x))))
        x = self.output(x)

        return x
```

Convolutions are used for learning feature extraction kernels. With several of these layers acting on different sized inputs the network learns to distinguish between features available at multiple scales. While normally the deeper the convolutional layer is the smaller the size of the input layer and the higher the number of filters in the kernel, because the distinguishing features of fine-grained computer vision problems show at micro-scales the first two convolutional layers have the same number of kernels. A combination of pooling and strides are used for downsizing the inputs. Xavier weight initialization and batch normalization are used to achieve faster convergence. The size of the first linear layer ensures the network has enough capacity for combining the features extracted by the convolutional layers while the dropout of the first linear layer tries to mitigate the risk of overfitting. The standard ReLU function is used for activations. The output layer size matches the number of classes in the dataset.

The transfer learning network is based on the pre-trained ResNet-50 model which is a convolution-based very-deep residual network with state-of-the-art

results in computer vision tasks. Recent application using transfer learning show that such a pre-trained network can be used as a feature extractor for other computer vision problems, using the learned weights from one dataset to solve a different problem (classification or regression) on an unrelated dataset. To speed up training the weights of the pre-trained network are kept fixed. The linear layers following the pre-trained model are the same as the ones following the convolutions of the scratch model to allow performance comparison on the convolutional layers.

The weights of both networks are optimized to minimize the cross-entropy loss using the Adam algorithm with a 0.001 learning rate for a total of 10 epochs.

3.3 Refinement

During the implementation of the scratch model the associated kernel size configuration was changed from $[24, 48, 144]$ to $[48, 48, 96]$ to enhance micro-scale feature extraction in the first convolutional layer by increasing its size and to lower the training time and model size by lowering the final convolutional layer size.

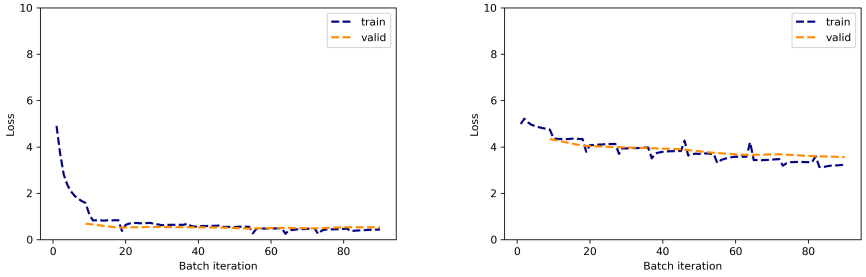
For both the scratch model and transfer one the batch size was decreased from 32 to 16 to enhance learning of fine-grained features while the learning rate was lowered from 0.05 to 0.001 to improve generalization.

These changes raised the accuracy of the scratch model from 8% to 18% and that of the transfer model from 64% to 84%.

4 Results

The end-to-end solution has a detection test accuracy, with separate models for human (Haar cascade) and dog (VGG16) entities, in input images of 98% for humans and 100% for dogs. Following detection the dog-breed classification model has 84% classification accuracy on dog entities while providing dog-breed similarity results for human entities. This is the dog-breed classifier based on the pre-trained ResNet-50 architecture which considerably outperformed the 18% accuracy of the custom CNN architecture. The comparison between the two dog breed classifiers is based on the performance of the convolutional layers, given that the rest of the parameters were kept constant in training (e.g. final linear layers, optimizer, loss function, batch size, learning rate etc.), on the test dataset between the best performing models on the validation set as checked after each epoch out of 10. These are the 10th epoch scratch model and the 6th epoch transfer model. An overview of the training process based on training and validation loss is shown in Figures 4b and 4a. The validation loss is calculated after each epoch while the testing loss is calculated every 50 batches (9 times per epoch).

The proposed solution shows a 17% increase in dog-breed classification accuracy compared to the benchmark model proving its adequacy.



(a) The transfer model shows a fast decrease of the loss during the start of the training an almost constant validation loss with a slowly decreasing training loss at the end.

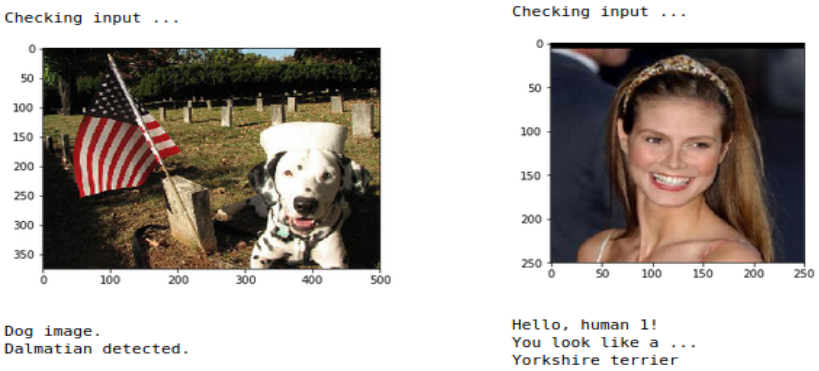
(b) The scratch model shows a slow and steady decrease in the loss over the entire training time still learning after 10 epochs with a validation loss decreasing more slowly than the training loss at the end.

Figure 4: Model training validation loss evolution after 10 epochs calculated after each epoch compared to 9 in-epoch train loss measurements.

5 Conclusion

5.1 Visualization

These app snapshots in Figure 5 show it can distinguish between humans and dog while tailoring the reply to the input accordingly. For the human case it runs the algorithm for each detected face which is why humans have an associated count number.



(a) App result on a dog image.

(b) App result on a human image.

Figure 5: Dog breed classification app snapshots.

5.2 Reflection

The end-to-end application built for the FGCV dog breed classification task includes processing of arbitrarily sized inputs through a pipeline consisting of

1. Pre-processing arbitrarily sized inputs
2. Human and dog detection
    - human face detection via an OpenCV pre-trained Haar cascade classifier
    - dog detection via the PyTorch pre-trained VGG16 network
3. Dog breed classification via transfer learning using a PyTorch pre-trained ResNet-50 network
4. Standardized result aggregation

with results exposed through both a Python function call on an input image path and a REST API call on an input image.

The solution to the challenging FGCV task proves adequate given the accuracy of 84% which goes beyond the 67% performance of the benchmark model and its flexibility provides the ground for further improvements.

5.3 Improvement

Looking beyond the current result both the model and the model serving can be further improved. The model architecture and training convergence can benefit from:

- hyper-parameter tuning via random grid search of
    - the size and number of linear layers,
    - the learning rate,
    - the batch size;
- longer training by increasing the number of epochs which could yield generalization through augmentation and dropout;
- comparison between results based on different feature extractors based on other pre-trained network like DenseNet and Inception;
- gradient-boosted decision trees (e.g. LightGBM or XGBoost) or a random forest acting on the features extracted from the pre-trained model, replacing the last linear layer, for classifying the dog breed;
- validation dataset augmentation to prevent overfitting.

On top of algorithm enhancements, serving the model can easily be enhanced with a HTML page with access to the REST API with functionality for

- user input;
- templating the response to the input

while the size and speed of the app can be improved by using the same pre-trained model for both dog detection an dog-breed classification since the result of the convolutional layers can be in turn be fed through both the pre-trained output layer for ImageNet or through the two custom output layers for the dog breed classifier.

References

[1] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft, 2013.

[2] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[3] Ranjita Thapa, Noah Snavely, Serge Belongie, and Awais Khan. The plant pathology 2020 challenge dataset to classify foliar disease of apples, 2020.

[4] Jiongxin Liu, Angjoo Kanazawa, David Jacobs, and Peter Belhumeur. Dog breed classification using part localization. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, pages 172–185, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[5] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[6] Gary B. Huang Erik Learned-Miller. Labeled faces in the wild: Updates and new reporting procedures. Technical Report UM-CS-2014-003, University of Massachusetts, Amherst, May 2014.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[8] Convolutional neural networks. https://cs231n.github.io/transfer-learning/.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[10] torchvision.models. https://pytorch.org/docs/0.4.0/torchvision/models.html.

[11] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018.

[12] Transfer learning. https://cs231n.github.io/transfer-learning/.

[13] Li Shen, Laurie R. Margolies, Joseph H. Rothstein, Eugene Fluder, Russell McBride, and Weiva Sieh. Deep learning to improve breast cancer detection on screening mammography. *Scientific Reports*, 9(1), Aug 2019.