Dog Breed Identification

Yerbol Aussat
School of Computer Science
University of Waterloo
Waterloo, ON, N2L 3G1
yaussat@uwaterloo.ca
December 3, 2017

Abstract

In this project we attempted to build a classifier capable of identifying a dog's breed based on its photo. Considering that on a high level dogs of different breeds can look very similar, it is a challenging task even for humans. To train and evaluate the model the "Columbia Dogs with Parts" dataset, containing 8,351 real-world images of 133 dog breeds, was used. The number of photos per breed ranged from 33 to 96. To tackle this problem, transfer learning, data augmentation and ensemble learning techniques were employed. As a result, the ensemble meta model averaging predictions of five transfer learning models that was trained on an augmented training dataset achieved a testing accuracy rate of 88.40%. In addition, a small CNN that was trained from scratch on an augmented training dataset achieved a testing accuracy rate of 54.42%.

1 Introduction

1.1 Task Definition

A lot of work has been done in the field of image recognition over the past few years. In this project a specific object recognition problem was examined. The goal is to build a model capable of identifying to which breed a dog belongs, based only on its photo. To train and evaluate the model the "Columbia Dogs with Parts" dataset, containing 8,351 real-world images of 133 dog breeds, was used.

There are several factors that make the problem of dogs categorization challenging. Firstly, there many different breeds of dogs exist, and all breeds on a high level look alike, i.e. there can be only subtle differences in appearance between some breeds. In fact, a dog's breed might be not obvious right away even for people who have an expertise in the domain. Therefore, determination of dog breeds provides an excellent domain for fine grained visual categorization experiments. Secondly, dog images are very rich in their variety, showing dogs of all shapes, sizes, and colors, under differing illumination, in innumerable poses, and in just about any location. The photos have different resolutions, backgrounds, and scales. On some images the dogs are are partially covered by other objects or wear clothes such as hats, scarfs, glasses. Thus, there is a lot of noise on many of the photos that makes the problem more challenging. Thirdly, our data set is small in terms of the number of photos per breed.

If humans want to distinguish a dog from a cat, or from some other object, then they need to look at things like ears, whiskers, tails, tongues, fur textures, and so forth. These are the features that we (humans) use to discriminate. But none of these features are available to a computer, which receive only a matrix of independent integers as an input, as illustrated on Figure 1.

Submitted to Introduction to Machine Learning (CS698, 2017 Fall). Do not distribute.

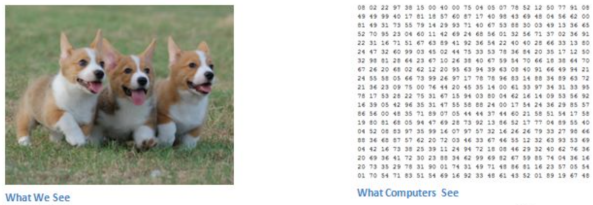

Figure 1: What we see vs What computers see [10]

Models such as Convolutional Neural Networks (CNN) allow computers to automatically extract hierarchies of features from raw pixels. These techniques proved to be successful in a variety of visual analysis tasks. In this work CNN - based approaches were applied to the problem of dog breed categorization. A number of techniques for improving performance of the model were used. These techniques are discussed in Section 3. Next, I experimented with data augmentation and transfer learning - two techniques that can potentially achieve high performance on small datasets. In this work it was shown that the models that use these two techniques can perform very well on a dog categorization task. Finally, I used ensemble methods to combine predictions of several separately trained CNN architectures.

2 Related Work

Before people learned how to use the power of deep learning, the best known approach was to carefully manually design the features. Typically, these features could only be applied to some specific narrow set of problems. To tackle the problem of image categorization [5] used a multiple kernel framework to combine kernels in a way that is most discriminative for each class. [6] also used the framework as part of an interactive system for bird species identification.

A simpler problem that is similar to dog categorization, is the problem of cats vs dogs classification, and a lot of work has been done on it. In [1], a classifier based on color features got 56.9% accuracy on the Asirra "Dogs vs Cats" dataset. In [2] Golle et al achieved an accuracy of 82.7% using a Support Vector Machines classifier based on a combination of color and texture features. In [3], researchers used the SIFT (Scale-Invariant Feature Transform) features to train a classifier and finally got an accuracy of 92.9% on the Dogs vs Cats problem. With advancements of CNNs, the accuracy rates increased significantly. Thus, B. Liu et al achieved the accuracy of 94% by using CNNs to learn features of images, and SVMs for classification [13]. On the "Cats vs Dogs" Kaggle competition highest result of 98.9% was achieved by P. Sermanet in 2013 [4].

However, a problem of dogs categorization has a number of major differences that make it significantly more challenging. Firstly, it is a multi-class classification problem, in which the algorithm would need to identify a dog breed out of 133 choices. Secondly, much more fine-grained photo categorization is required, because dogs of different breeds look more similar to each other than to cats. Finally, Dogs vs Cats dataset had several orders of magnitude more data available for each category: 12,500 vs 60.

In [7] Liu et al tried to tackle the dog breed identification problem using part localization. They built exemplar-based geometric and appearance models of dog breeds and their face parts. Their approach also featured a hierarchy of parts and breed-specific part localization. A recognition rate of 67%was achieved on a large real-world data set. Also, they experimentally demonstrated that accurate part localization significantly increases classification performance.

Tremendous progress has been made in fine-grained categorization with the advancement of deep CNNs. [8] discusses data augmentation as an effective technique for improving the performance of CNNs when dealing with limited datasets.

[9] points out that deep learning models are by nature highly repurposable, and therefore a technique called transfer learning (transferring learned general knowledge from one, usually larger, problem to another one) is possible. It is emphasized that it is a very effective method for cases when only a small dataset is available for training.

3 Techniques

Convolutional Neural Networks

A convolutional neural network is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery. CNNs take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. The layers of a CNN have neurons arranged in 3 dimensions in so called convolution layers. In the convolution layer, the filters are passed across the input, row by row, and they activate when they see some type of visual feature. Now, rather than treating each of the raw pixel values in isolation, CNN's treat them in small local groups. These sliding filters are how the CNN can learn meaningful features and locate them in any part of the image [10].

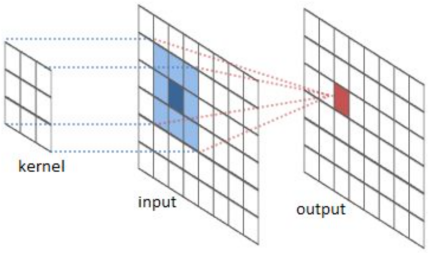

Figure 2: Kernel [10]

These is only a high level overview of the CNN. There are also many other details related to other layers (pooling, relu, fully-connected, dropout), loss functions (cross-entropy, MSE, MAE, cosine similarity), optimizers (SGD, RMV, Adagrad), etc. These details won't be discussed in this paper.

There is almost a linear relationship in the amount of data required and the size of the model. General idea is that the model should be large enough to capture relations in the data along with specifics of our problem. Early layers of the model capture high level relations between the different parts of the input (like edges and patterns), whereas later layers capture information that usually helps to discriminate between the desired outputs. Therefore if the complexity of the problem is high (like Image Classification) the number of parameters and the amount of data required is also large [11].

A recurring problem faced when training neural networks is that there is typically not enough data to maximize the generalization capability of CNN. Some of the techniques to address it include data augmentation, transfer learning and regularization / dropout.

Data augmentation

A powerful tool for dealing with cases when the data set is small or unbalanced is data augmentation. Essentially, data augmentation is the process of artificially increasing the size of your dataset (and model generalizability) via transformations. An example of data augmentation can be seen on Figure 3 [11].

Figure 3: Data Augmentation [11]

Data augmentation includes things like randomly rotating the image, zooming in, adding a color filter etc. It is typically only applied to the training set and not on the validation/test set. Still care should be taken when doing data augmentation, since the augmented samples are still highly correlated [8].

Transfer Learning

As it was discussed above, deep learning requires the ability to learn features automatically from the data, which is generally only possible when lots of training data is available - especially for problems where the input samples are very high-dimensional, like images.

The transfer learning technique is using an existing CNN feature extraction and the associated trained network weights, and transferring the learned general knowledge from one problem to another one. As it was mentioned earlier CNNs exploit the hierarchical distributed representations. The lower layers of a CNN contain more generic features that should be useful to many tasks, but higher layers of the CNN becomes more specific to the details of the domain classes contained in the original dataset. Transfer learning is schematically illustrated on Figure 4 [10].

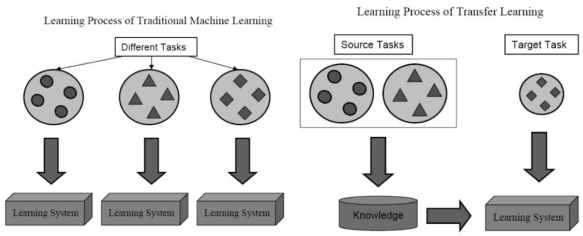

Figure 4: Transfer Learning

Since modern CNNs take very long time and computational power to train on ImageNet, it is common to see people release their final CNN trained weight for the benefit of others who can apply the pre-trained networks in a wide range of vision-related problems.

A basic strategy for transfer-learning is to take a CNN pre-trained on some very large dataset, remove the last fully-connected layer, and then treat the rest of the CNN as a fixed feature extractor for the new problem-specidic dataset. Once the features are extracted, a linear classifier can be trained for the new dataset. [10].

Fine-Tuning

Another transfer-learning strategy is to not only replace and retrain the classifier on top of the CNN on the new dataset, but to also fine-tune the weights of the pre-trained network by continuing the backpropagation. In practice, usually only the last convolutional layers are fine-tuned, and the the earlier layers remain frozen (due to overfitting concerns). This is again explained by the observation that the earlier features of a CNN tend to contain more generic features that can be useful to many tasks. In order to perform fine-tuning, all layers (both layers of the pre-trained model and the layers of added fully-connected layers) should start with properly trained weights.

The following rules are usually used for deciding whether to fine-tune the model [10]:

- Similar and small dataset: avoid overfitting by not fine-tuning the weights on a small dataset, and use extracted features from the highest levels of the CNN to leverage dataset similarity.
- Different and small dataset: avoid overfitting by not fine-tuning the weights on a small dataset, and use extracted features from lower levels of the ConvNet which are more generalizable.
- Similar and large dataset: with a large dataset we can fine-tune the weights with less of a chance to overfit the training data.
- Different and large dataset: with a large dataset we again can fine-tune the weights with less of a chance to overfit.

Ensembles

Ensemble learning is a widely-used technique that helps to improve machine learning results by combining several models. This approach allows the production of better predictive performance

compared to a single model. Ensemble learning is often associated with the concept of ”wisdom of crowds”. It is a meta-algorithm that combines several machine learning techniques into one predictive model in order to improve performance.

Voting and averaging are two of the easiest and yet arguably most widely used ensemble methods [8].

General Techniques for Improving CNN Performance

There are a number of good deep learning practices that help to improve performance of models. Regularization helps to deal with overfitting. The three most popular options are: dropout, L1 and L2 regularization. Dropout is the most commonly used regularization in deep learning. It essentially deletes a random sample of the activations (makes them zero) in training. Dropout makes the model more robust but making sure that it doesn't rely too much on any individual activations. Usually it is applied to fully connected layers at the end of the model [8].

Batch normalization is a relatively new concept and therefore was not yet implemented in earlier models such as VGG. It adds a ”normalization layer” after each convolutional layer, which allows the model to converge much faster in training and therefore also allows us to use higher learning rates. It is generally advised to always use it. [8]

Other two simple techniques that should be always used are 1) early stopping, which is essentially stopping training when the loss function stopped to decrease, and 2) saving best performing models on the validation set during training.

4 Empirical Evaluation

4.1 Dataset

To train and evaluate the models the ”Columbia Dogs with Parts” dataset was used [https://people.eecs.berkeley.edu/ kanazawa/].

The dataset contains 8,351 real-world images of 133 American Kennel Club (AKC) recognized dog breeds. The images were collected from different sources such as Google image search, Flickr and ImageNet. Also, 8 dog part locations are annotated for each image. These parts include both eyes, the nose, the tips of both ears, and three points marking the top of the dogs head between the ears. This additional information on dog parts was not used in our project, because CNNs implicitly extracted all features / dog parts for us.

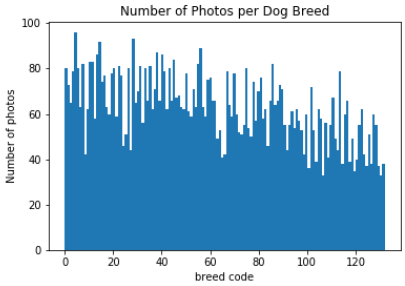

Figure 5: Distribution of photos among classes

Distribution of images among different classes is shown on Figure 5. As we can see the dataset is not very balanced. The most frequently occurring breed in the dataset is Alaskan Malamute with the total of 96 photos, and the frequently occurring breed is Norwegian buhund with the total of 33 images. The dataset was split into three parts: 1) training dataset consisting of 6680 photos(80%), validation dataset consisting of 835photos (10%) and testing dataset consisting of 836 photos (10%). The number of images per breed in each of three formed datasets is proportional to the number of images per breed in the original dataset.

4.2 Training a small CNN from scratch

As an initial baseline, we constructed a small CNN from scratch, and trained it. Since we had a very small number of examples per class, the biggest concern in this experiment was overfitting: the model exposed to too few examples could learn patterns that do not generalize well to new data. First, the number of parameters in the model, i.e. the number of layers and the size of each layer, should be carefully chosen.A model that can store a lot of information has higher expressivity and higher potential to be more accurate by leveraging more features, but it is also at a higher risk to start storing irrelevant features, i.e. overfitting. Since our dataset is small, the neural network shouldn't be too deep and wide to make sure that the model generalize, and not just "memorize" the training data.

After conducting a few experiments, we came up with an architecture that is a stack of 5 convolution layers with a ReLU activation. This is our feature extractor. This feature extractor was followed by one fully-connected layer that does classification. Following the best practices for building deep neural networks, we added max-pooling and batch normalization layers after all convolutional layers. Dropout was used before the fully connected layer to help to further reduce overfitting by preventing a layer from seeing twice the exact same pattern.

Also, in all experiments the best performing model on the validation set during training were saved, and used for testing. Also early stopping was used in all experiments. As a result, validation accuracy of **35.34%** and testing accuracy of **35.89%** was achieved.

4.3 Training s small CNN from scratch + Data Augmentation

As it was discussed in Section 3, another way to fight overfitting for cases when there is only a small training dataset is data augmentation. In the next experiment the same neural network as in part 4.2 was used with the only difference that data augmentation of the training data was performed. Many deep learning frameworks can generate augmented data. For instance, Keras has a built-in ImageDataGenerator method to randomly augment the data during training, so that at each epoch, a model sees slightly different images. Slight rotations, shear transformations, slight horizontal and vertical shifts, and random horizontal image flips were used to augment the training data. Using data validation resulted in the validation accuracy of **56.76%** and testing accuracy of **54.42%**, which is a significant improvement.

4.4 Transfer Learning: Extracting bottleneck features and using a classifier on them

The next, more refined approach that we experimented with was to leverage a network pretrained on a large dataset. In the next set of experiments VGG16, VGG19, ResNet50, Inception-V3 and Xception architectures pre-trained on the ImageNet dataset were used as feature extractors. Because the ImageNet dataset contains many "dog" classes among its total of 1000 classes, this model would already have learned features that are very relevant to our dog breed classification problem.

In the first part of transfer learning, the "bottleneck features" were extracted using each of the pre-trained models in the following way. We only instantiated the convolutional part of the model, everything up to the fully-connected layers. This model was then run on our training, validation and testing data once, and the output was recorded. These are the "bottleneck features" from the model, corresponding to the last activation maps before the fully-connected layers. In the second part predictions based on the extracted "bottleneck features" were calculated. A small fully-connected model was trained on the on top of the stored "bottleneck" features using the original labels. The reason why we were storing the botleneck features offline rather than adding our fully-connected model directly on top of a frozen convolutional base and running the whole thing, was computational effiency.

Running deep neural networks such as ResNet, Inception or Xception that have more than hundred of layers is very expensive, and therefore, we want to only do it once. However, this prevented us from using data augmentation using Keras's ImageDataGenerator method.

The features extracted from all of the pre-trained models were classified using the same fully-connected classifier illustrated on Figure 6. The first fully-connected layer consists of 266 neurons, and the second (prediction) fully-connected layer consists of 133 neurons. The Xception-based and Inception-V3-based transfer learning models demonstrated the best results with validation accuracy

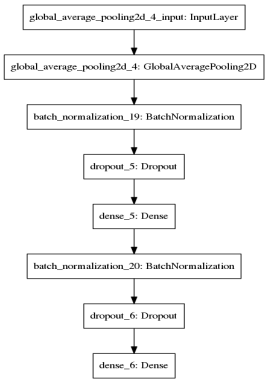

Figure 6: Fully connected classifier on the top of the pre-trained model

rates of **84.19%** and **85.15%**, and testing accuracy rates of **84.21%** and **83.13%** respectively. Finally, we built the ensemble of all 5 models that was averaging the predictions. Using the ensemble of all five models significantly improved the result.The ensemble meta-model achieved the validation accuracy of **89.58%** and the testing accuracy of **87.55%**, which is more than thee percent higher than the best individual model. The obtained results for this experiment are summarized in Table 1.

Note: We also experimented with first adding the fully connected classifier from Figure 7 on the top of the pre-trained model, then freezing the pre-trained layers and running hte entire network, but the achieved performance was consistently lower by 5-10%.

4.5 Transfer Learning + Data Augmentation

In the net set of experiments transfer learning was performed on the augmented data. As it was mentioned above, we couldn't use the built-in Keras's ImageDataGenerator method with transfer learning, because of the implementation details related to the fact that we were separately extracting the bottleneck features first. By randomly applying training slight rotations, shear transformations, slight horizontal and vertical shifts, and random horizontal image flips, the training dataset was increased by 3 times. Then the transfer learning was applied to this augmented training dataset exactly as described in Section 4.4. Again, the Xception-based and Inception-V3-based transfer learning models demonstrated the best results with validation accuracy rates of **86.11%** and **86.82%**, and testing accuracy rates of **85.29%** and **85.05%** respectively. The ensemble meta-model, averaging the results of all 5 models achieved the validation accuracy of **90.78%** and testing accuracy of **88.40%**. The obtained accuracy rates are summarized in Table 1.

Pre-trained Architecture	Validation	Test	Validation (Data Augmentation)	Test (Data Augmentation)
VGG-16	79.52%	80.26%	80.48%	80.14%
VGG-19	81.08%	79.43%	80.48%	79.54%
ResNet-50	83.71%	82.66%	85.15%	83.13%
Inception-V3	85.15%	83.13%	86.82%	85.05%
Xception	84.19%	**84.21%**	86.11%	**85.29%**
Ensemble	89.58%	**87.55%**	90.78%	**88.40%**

Table 1. Accuracy rates for transfer learning-based models. Results for five pre-trained architectures and their averaging ensemble are shown.

4.6 Fine-Tuning

Since our dog breeds data set is very small in terms of number of photos per class, generally, fine-tuning the CNN is not a good idea due to overfitting concerns [10]. Since the data is similar to the original data, we expect higher-level features in the CNN to be very relevant to our "Dog breeds"

dataset too, and therefore we expect that the best result would be achieved just by training a simple classifier on the features extracted from the pre-trained CNNs.

For the sake of experiment, fine-tuning was performed on ResNet-50 architecture, and as expected, the model overfitted.

5 Conclusion

In this work we tried to tackle the dog breed identification problem using a very small dataset with only a few dozens of images per breed.

First, a small convolutional neural network was built and trained on the dataset from scratch, and the accuracy of 35.89% was achieved on the testing dataset. Then this result was further improved by applying data augmentation to the training dataset. The testing accuracy of 54.42% was achieved. This result is very impressive considering the complexity of the problem, and the limited amount of data.

Next, the transfer learning technique was applied to the problem. Xception-based transfer learning model demonstrated the best performance achieving the accuracy of 84.21% and 85.29% on the testing dataset without and with data augmentation respectively. The ensemble of all five models demonstrated the testing accuracy of 87.55% and 88.40% without and with data augmentation respectively.

Even though at first glance the dog breed identification seems to be a very challenging problem, it was shown that a powerful and highly accurate CNN-based image classification model can be built with help of data augmentation, transfer learning and ensemble methods.

Some of the approaches that could potentially further improve the accuracy are: 1) More aggressive data augmentation, 2) More aggressive dropout, 3) Use of L1 and L2 regularization. 4) Adding more images to the training dataset. With a large dataset we again can fine-tune the weights with less of a chance to overfit.

References

[1] J. Howell J. Elson, J. Douceur and J. Saul. Asirra: a captcha that exploits interest-aligned manual image categorization. In *Proc. of ACM CCS*, pages 366–374, 2007.

[2] P Golle. Machine learning attacks against the asirra captcha. In *In Proceedings of the 15th ACM conference on Computer and communications security*, pages 535–542, 2008.

[3] Vedaldi A. Zisserman A. Jawahar C. V. Parkhi, O. M. Cats and dogs. in computer vision and pattern recognition (cvpr). In *IEEE Conference*, pages 3498–3505, 2012.

[4] Kaggle cat vs dog competition's public leaderboard, . https://www.kaggle.com/c/dogs-vs-cats/leaderboard.

[5] Zisserman A. Nilsback, M.E. Automated flower classification over a large number of classes. In *Proc. 6th Indian Conf. on Computer Vision*, pages 722–729, 2008.

[6] Wah C. Schroff F. Babenko B. Welinder P. Perona P. Belongie S. Branson, S. Visual recognition with humans in the loop. In *Proc. ECCV*, 2010.

[7] D. W. Jacobs J. Liu, A. Kanazawa and P. N. Belhumeur. Dog breed classification using part localization. In *Proc. ECCV*, 2012.

[8] D. Steinkraus P. Y. Simard and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, 2003.

[9] Sinno Jialin Pan Qiang Yang. A survey on transfer learning. In *IEEE Transactions on Knowledge Data Engineering*, 2010.

[10] Stanford computer vision course. lecture notes, . http://cs231n.github.io/.

[11] Francois Chollet. Tutorial: Building powerful image classification models using very little data.

[12] K. Zhou B. Liu, Y. Liu. 4image classification for dogs and cats. Technical report, University of Alberta, 2013. https://sites.ualberta.ca/ bang3/files/DogCat$_r eport.pdf$.