Machine Learning Algorithm Case Study on Dog Breed Classification

Momin Chaudhry
Computer Science
Ryerson University
Toronto, Canada
momin.chaudhry@ryerson.ca

Hassaan Abbasi
Computer Science
Ryerson University
Toronto, Canada
hassaan.abbasi@ryerson.ca

Mohammad Saleh Joonghani
Computer Science
Ryerson University
Toronto, Canada
msalehjo@ryerson.ca

***Abstract* — This is the final project report for the CPS803 (Machine Learning) course. This report aims to compare and contrast some different machine learning algorithms while performing dog breed classification on various images.**

I. Introduction and Motivation

A. Significance of the Challenge

Choosing the appropriate algorithm for fine-grained classification under certain conditions such as limited resources, low computational power or memory, and having very similar images is a difficult challenge. The task of classifying dog breeds is a demanding fine-grained classification problem, since various canine breeds have similar body features and structure, like Whippets and Italian Greyhounds, yet other breeds, like German Shepherds, have a wide range of looks (Fig. 1). There are many algorithms and methods that can tackle this tough problem, and each of which have different requirements to be more accurate. These algorithms have different run-times and memory usage, and can also leverage different hardware such as GPUs to complete the job more efficiently.

The solutions and observations that arise from this problem will be relevant for other types of fine-grained classification problems. These algorithms are not made to only classify dog breeds, as they can also be used to identify many other classes of objects as well. For example, fine-grained classification algorithms can be used in self-driving cars in order to help them identify other vehicles on the road. Through our research, experiments, and observations, we hope to shed light on the performance of certain classification algorithms.

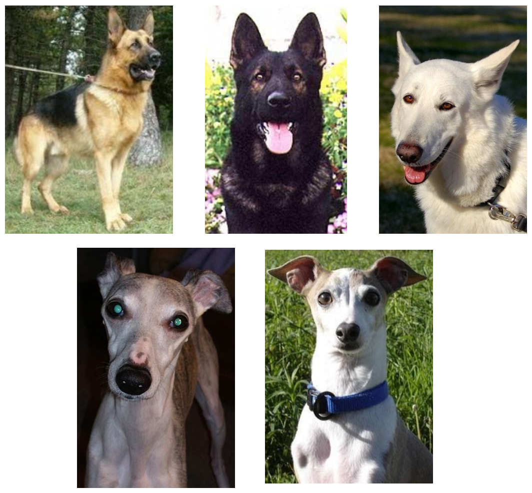


Fig. 1. The first row of images are all of German Shepherds, yet they all have different orientations and colours. The second row of images contains a Whippet (left) and an Italian Greyhound (right), which look very similar despite being different breeds.

B. Overview

This theoretical project provides a more in-depth answer to when one should use algorithms such as Support Vector Machines (SVM), K-means Clustering, and Convolutional Neural Networks (CNN) while identifying different dog breeds through images. In this project, we used the aforementioned algorithms on small and large datasets of grey scaled SIFT features of images and features extracted by a popular CNN architecture, and compare run-times, and accuracies using confusion matrices. These comparisons are then visualized by graphs in later sections.

Throughout this report, the following short-forms will be used to identify certain

[1] *GitHub Repository:* https://github.com/mominchaudhry/CPS803_project

algorithms: SVM_SIFT (SVM using SIFT features), SVM_CNN (SVM using CNN extracted features), kmeans_SIFT (K-means using SIFT features), kmeans_CNN (K-means using CNN extracted features), CNN (CNN with fully connected layers).

C. *Hypotheses*

K-means Clustering is an unsupervised algorithm which clusters data that are close to each other in euclidean space. SVM is a supervised algorithm which finds the optimal separation between classes of data. As the dimensions become too large, the chances of different breeds overlapping increases. So using SIFT features to train the K-means algorithm was expected to perform poorly on this fine-grained dataset. The SVM should perform better than K-means on the data since it knows the labels to find the optimal decision boundaries. The CNN will find the best features to use, so extracting image features from the CNN and using these to train other algorithms should perform much better than other human extracted features. We believe that the CNN will still be the best performing model overall.

II. Problem Statement and Dataset

A. *Problem Statement*

We would like to determine which algorithm performs best overall for this dataset, as well as with a very small number of training images. We plan to implement each of these algorithms using different feature sets and compare which algorithm performs best for each feature set with the full number of training images and with the subset of training images. We will also discuss which cases are optimal for each algorithm, and which cases each algorithm struggled with.

B. *Dataset*

We used the Stanford Dogs dataset to conduct our experiments which contains 20,580 images from across 120 different classes of dog breeds [2]. There are over 100 images per class, however the dataset was still imbalanced with some classes having more images than others. The dataset we used also came with pre-extracted SIFT features, both before and after applying a histogram intersection kernel. It also included annotations for each image containing the labels and bounding boxes for the dogs in each image. This dataset was made up from the dog images from the ImageNet dataset.

III. Methods and Models

A. *Implementation and Code*

We used Python to implement all of our models and experiments for this project, which can be found in our [1]GitHub repository. For the SVMs, we used the scikit-learn library. We also used scikit-learn’s cluster module for the K-means models as well as the metrics module for confusion matrix and accuracy score. We used scikit-learn’s model_selection module as well for the train_test_split method in each of our SIFT experiments to split the testing features into a validation and testing set. We used PyTorch’s torch library for our CNN, the nn module within torch, and the optim module for mini-batch gradient descent. We also used PyTorch’s torchvision library, the transforms module in torchvision to perform our data augmentation, and the models module to get our models for transfer learning. We used NumPy arrays to hold our SIFT features and other data throughout our experiments. We also used the Matplotlib library to plot our results as well as some other libraries including time, copy, cv2, and os for various tasks such as preprocessing, recoding runtimes, etc.

B. *Preprocessing and Feature Extraction*

All of the images in the dataset were very large with the dogs being only a small portion of each image, outlined by the bounding box in that image's corresponding annotations file. We wrote a script to crop all images by their bounding boxes and saved them. In order to run our experiments on a subset of this raw image data, we also wrote a script to get the first 15 images from each class, and save them into a new folder.

The SIFT features we used were provided in the dataset in .mat files (HDF5), with training and testing data already split (12,000 training images and 8580 testing images). As mentioned in the dataset section, the HDF5 file contained the SIFT features before applying the kernel as well as after. Since we needed to extract a subset of the features to run some of our experiments, we used the features prior to the kernel application. Based on the labels, we discovered that the data was structured in an ordered list of 100 images per class, followed by the next 100, and so on. We wrote a script to get the first 15 examples of every 100 examples of SIFT features to match our subset of raw image data extracted earlier.

Once we implemented and trained the CNN architecture which we found best for our dataset, we loaded the model and ran our set of 100 images per class through it once, while copying the embedding after the avgpool layer, which contained the features generated by the model right before passing it to the FC layer. We also ran our subset folder of raw image data through this feature extractor once to get a subset of features (15 images per class) generated by the CNN model.

With all our features ready, we had 12,000 images for training for each feature set in the large dataset, and 1800 images for training for each feature set in the subset. We also had 8580 images for testing. Since we experimented a lot with each model, we thought it would be best to split the testing data into a validation set, and set aside the rest for testing at the end. We split our 8580 testing images in half to make validation and test sets of 4290 images, for each feature set. The reason we chose to split our testing data instead of training was because our training data was only about 60% of our total dataset, and we found predicting a validation set of 4290 images was enough to get a good sense of each model's performance since the accuracies were very close to testing with the full set of 8580 images. So a 60/20/20 split would save enough data for testing. This way we were also able to avoid losing any training data.

C. *SVM Methods*

The first model we implemented was an SVM. We chose SVM as our supervised learning model since it is a common method for multi-class classification that is known to perform better than most other non-neural network models. We experimented with different kernels and implementations, for example we tried using Sklearn's SVM with a precomputed kernel, using the histogram intersection kernel that was already applied to our SIFT features. We also tried using the SIFT features before the kernel, and used the SVM's default rbf, as well as a linear kernel. We also experimented with different multi-class methods such as one-vs-one and one-vs-rest. The one-vs-one classification method was expected to outperform the one-vs-rest since we thought it would find finer differences among similar breeds by comparing each with each other, however it took significantly longer to run and had a bit worse results. Overall we found one-vs-rest classification with the default rbf kernel to be ideal for classifying our SIFT features.

We experimented with different SVM implementations for our CNN extracted features as well after we chose our optimal CNN model described in the next section, and found one-vs-rest classification with the default rbf kernel to be ideal in this case too. This model turned out to be our best model with the highest f1-score, as discussed in our results section on the next page.

D. *CNN Methods*

It is known that neural networks are very well at image classification, especially with many convolutional layers to extract the best features of images before passing them to some fully connected layers. We first tried to implement a simple CNN with 2 convolution and pooling layers and 2 fully connected layers (network 1), but this turned out to be worse than our SVM with SIFT features. We then went on to implementing a deeper network with 5 convolution and pooling layers and 3 fully connected layers (network 2) which we noticed was significantly better than the previous architecture and showed that the deeper the CNN, the better performance was.

After some research we decided to implement transfer learning using some pre-trained models. For this we also performed some data augmentation, we scaled the images to 224x224, performed some random horizontal flips, normalized the data, and converted to tensors. We tried a few different architectures including AlexNet, DenseNet-121, and ResNet-18. We found residual networks to perform best so we tried implementing deeper versions of ResNet (ResNet-50 and ResNet-152). We found ResNet-50 to have the best results with the least variance among the training and validation set compared to the other models, all models reached above 99% accuracy on the training sets but ResNet-50 had the best score on the validation set. The deeper ResNet-152 model did not perform as well as the smaller ResNet-50 model because it overfitted to the training data, even after adding weight decay.

Since the convolutional layers of CNN's are very well feature extractors, we used our fine tuned ResNet-50 model on our training data once more but copied the embeddings after the avgpool layer. We then used these embeddings as features with our other models to see how well they performed compared to the fully connected layers of this model, which we will talk about in the results section.

E. *K-Means Methods*

We also wanted to implement an unsupervised learning algorithm for classifying different breeds without knowing their labels. We first tried training a basic Sklearn k-means clustering model on our SIFT features. From here we ran into an obstacle because the cluster labels generated by k-means did not correspond to the expected labels we used to compute the models accuracy. In order to work around this, we plotted a confusion matrix for the cluster labels of all training examples vs the true training labels.

There was a noticeable mapping between the cluster labels and the true labels so we implemented some post-processing code to map the predictions of our model on the test data to the corresponding class label. This gave us a more reliable accuracy estimate, but we still found k-means to perform poorly compared to the SVM model using these same SIFT features. This was expected for a clustering algorithm since data overlaps which causes it to cluster similar breeds into the same group even if they are different breeds.

We tried this unsupervised clustering approach with the features we extracted through our CNN model. Although the CNN model was trained using the image labels, which would be considered supervised learning, we still wanted to see how well an unsupervised clustering algorithm performs on these features without the labels since the SVM performed better than expected. Using this strategy and the same post-processing on the predictions as before, we had a significantly improved accuracy which was even comparable to the SVM and fully connected layer of the CNN.

F. Performance Metrics

To evaluate the performance of our model we decided to use f1-score measure, which is the harmonic mean of the precision and recall. We chose this measure since our testing data was very imbalanced. Throughout our research and experiments, we used accuracy per class and mean accuracy to visualize the performance of our models, and while evaluating the test set, since we noticed it was imbalanced we decided to calculate the f1-scores as well. They reflected the mean accuracies of each model very closely, but gave some interesting observations, so we decided to use these to compare our results.

IV. Results and Discussion

A. Models Trained with SIFT Features

With the SVM we noticed that implementing the histogram intersection kernel had the best mean accuracy of 18%, but since we wanted to compare our results with these features on a smaller dataset, we were unable to get a subset of the data from the feature matrix after the kernel application. However we found the SIFT features before the kernel application provided in the HDF5 file had a close 17% mean accuracy with an rbf kernel, and these features also worked better with our unsupervised model. As mentioned in the methods section, we also tried using different kernels such as a linear kernel and different classification methods like one-vs-one and one-vs-rest. Through our research we found other work done in dog breed identification, where using facial keypoints as features worked best with a linear kernel SVM [1]. However in our case we found a linear kernel could not separate our data well enough. As mentioned we also found the one-vs-rest classifier to work better and a lot faster than the one-vs-one. We think this was because it had a lot more models to create, and each model was not as accurate as the models created by the one-vs-rest classifier.

When we implemented the K-Means model with these SIFT features, we decided to try using the feature with the histogram intersection kernel applied, but this gave us a very poor accuracy of around 6%. However the SIFT features before the histogram intersection kernel gave a mean accuracy of almost 9%. We think this was because the K-means clustering algorithm doesn't perform as well on features which use a kernel trick, these kernel tricks are meant for SVMs.

B. CNN Results

For the basic CNN architecture we implemented at first (network 1), we got a validation accuracy of around 12%, with training accuracy still under 40%, which indicated large bias and variance. However we knew that CNN's could do much better than this, so we expanded the CNN by adding more layers and making them wider. With the network with 5 convolutional and pooling layers and 3 fully connected layers, we got a mean accuracy of 34% with training accuracy around 60%, which was much better than our current best model, SVM_SIFT, as expected according to our hypothesis. But these results still indicated large bias and variance.

When we tried transfer learning with different architectures, we first tried AlexNet, which gave a 63% mean accuracy on the validation set. We decided to try with a different architecture, and from our research we discovered DenseNet performs better and faster than AlexNet, so we decided to implement DenseNet-121 since it was the most common DenseNet architecture. This gave us a mean validation accuracy of around 80%. After this we wanted to try one more type of architecture which was residual networks since DenseNet is quite similar to ResNet. We found ResNet-18 to perform even better, with a mean validation accuracy of 83%. We then decided to experiment with different depths of this network, so we tried ResNet-50 and ResNet-152 as well, and found ResNet-50 to have the least variance, with a validation accuracy of 90%. All of our transfer learning models had training accuracies of 97-99%, which show that the bias was already minimized in these models, it was just a matter of reducing variance as much as possible, which we found our fine-tuned ResNet-50 model to achieve quite well.

C. Models Trained with CNN Extracted Features

As mentioned before, we would like to use our fine-tuned ResNet-50 model as a feature extractor and try training an SVM and a K-means clustering model using these features. To compute these features, we passed our training images to our CNN model, and copied the feature embeddings after the avgpool layer, and built a numpy array to store these embeddings. We first tried using these features to train an SVM, which we tried all the same variations as we did with our SIFT features, and it turned out that the default rbf kernel and a one-vs-rest classifier worked best yet again, with a mean accuracy of 94%, which was much higher than our fine-tuned ResNet-50 model. This shocked us as it went against our hypothesis that the CNN with the fully connected layers would perform better than either of the other models with the CNN extracted features.

We also used the ResNet-50 extracted features to train a k-means clustering model. This gave us significantly better results than our k-means model trained with SIFT features. We got a best mean accuracy of 90%, which was similar to the mean accuracy of the CNN with fully connected layers. However because of the post-processing needed for our k-means models, there are some classes which the model missed completely, causing the precision and recall to be worse, and hence a lower f1-score.

D. *Scores and Comparisons*

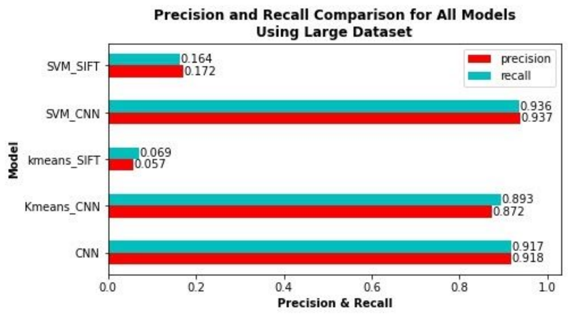


Fig. 2. Precision and Recall for all models trained on large dataset

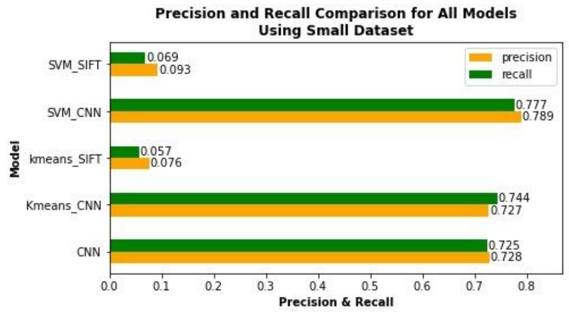


Fig. 3. Precision and Recall for all models trained on small dataset

An interesting observation in the first plot (Fig. 2) is that the SVM models perform better and have a higher precision with both feature sets, while the k-means models have a higher recall.

We see a similar trend in the models trained with smaller datasets (Fig. 3), except here the K-means model trained with SIFT features had a higher precision. SVM_CNN had the highest precision and recall values, and KMEANS_SIFT had the lowest values in both large and small testsets.

TABLE I

F1 Scores

	SVM SIFT	**SVM CNN**	**K MEANS SIFT**	**K MEANS CNN**	**CNN**
Large set	0.17	0.94	0.06	0.88	0.92
Small set	0.08	0.78	0.07	0.74	0.73

This table illustrates the F1 scores of all models by calculating the harmonic mean of precision and recall values.

There are some interesting observations in the f1-scores of the models. We noticed that the fully connected layers of the CNN's do not perform as well as expected, with the SVM performing better with the same features from the large dataset, and both SVM and K-means performing better with the same features from the smaller dataset. Another point is that SVM's perform better with more data, regardless of the features. We noticed that K-means does better with smaller datasets. As expected according to our hypothesis, all of our models perform better on CNN extracted features than they do on SIFT features.

Training times on the K-means models were around the same regardless of dataset size, while SVM's took a fair bit longer on the large dataset even though they trained quite fast on the small dataset. Predicting on K-means models were fastest, as they gave almost instantaneous predictions.

E. *Misclassifications*

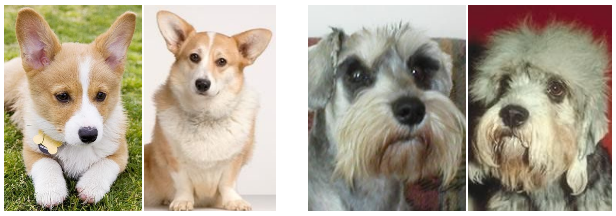
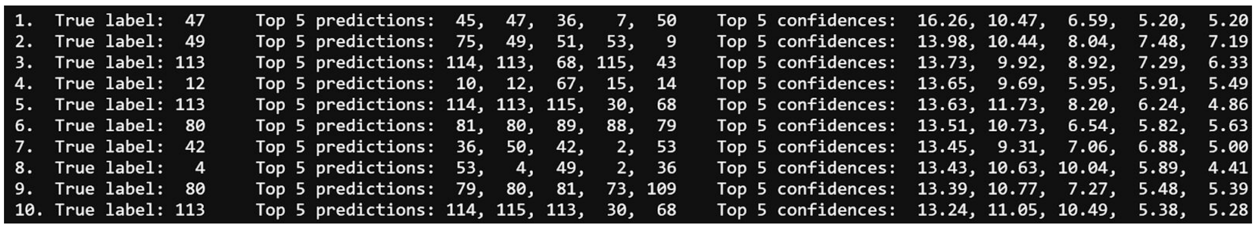


Fig. 4. The first pair of images are of a Pembroke (left 1) and a Cardigan (left 2), the second pair contains images of a Miniature Schnauzer (right 1) and a Dandie Dinmont (right 2)



```
1.  True label:  47     Top 5 predictions:  45,  47,  36,   7,  50     Top 5 confidences:  16.26, 10.47,  6.59,  5.20,  5.20
2.  True label:  49     Top 5 predictions:  75,  49,  51,  53,   9     Top 5 confidences:  13.98, 10.44,  8.04,  7.48,  7.19
3.  True label: 113     Top 5 predictions: 114, 113,  68, 115,  43     Top 5 confidences:  13.73,  9.92,  8.92,  7.29,  6.33
4.  True label:  12     Top 5 predictions:  10,  12,  67,  15,  14     Top 5 confidences:  13.65,  9.69,  5.95,  5.91,  5.49
5.  True label: 113     Top 5 predictions: 114, 113, 115,  30,  68     Top 5 confidences:  13.63, 11.73,  8.20,  6.24,  4.86
6.  True label:  80     Top 5 predictions:  81,  80,  89,  88,  79     Top 5 confidences:  13.51, 10.73,  6.54,  5.82,  5.63
7.  True label:  42     Top 5 predictions:  36,  50,  42,   2,  53     Top 5 confidences:  13.45,  9.31,  7.06,  6.88,  5.00
8.  True label:   4     Top 5 predictions:  53,   4,  49,   2,  36     Top 5 confidences:  13.43, 10.63, 10.04,  5.89,  4.41
9.  True label:  80     Top 5 predictions:  79,  80,  81,  73, 109     Top 5 confidences:  13.39, 10.77,  7.27,  5.48,  5.39
10. True label: 113     Top 5 predictions: 114, 115, 113,  30,  68     Top 5 confidences:  13.24, 11.05, 10.49,  5.38,  5.28
```

Fig. 5. Top 10 most confident incorrect predictions with their true label and top 5 predicted classes/confidences

We also wanted to analyze the misclassifications of our fine-tuned-ResNet-50 model. We decided to find the most confident incorrect predictions, with their top 5 incorrectly predicted classes and confidences. There are some very interesting observations here, we found that the most confident incorrect prediction in the whole testing set was predicting an image of a Miniature Schnauzer as a Dandie Dinmont (Fig. 4). Another pair the ResNet-50 model seemed to confuse were Pembrokes as Cardigans (Fig. 4). The model was very confident when giving this prediction, it appeared 3 times in the top 10 incorrect predictions (Fig. 5). Although the most confident predictions for each of these images were incorrect, the true labels did appear within the top 3 class predictions.

F. *Possible Next Steps*

If we were to continue this project, we would like to try building our own CNN to get the same or better results as our transfer learning approach. We would also like to look into why the SVM performs better than the CNN fully connected layers when trained with the feature extracted from the CNN and perhaps try to improve our transfer learning model. This could involve hyperparameter tuning, since we were not focused on making our models the absolute best, but rather to compare them in different cases. Though it would be nice to see how we can improve our current best results.

We would also like to compare some other algorithms, but due to limited time we had to choose 3 only which was already quite a bit of work to make each of them perform as best as they could. If we were to compare some other models, we would focus on supervised learning algorithms, perhaps a bag-of-words model, logistic regression, or K-nearest-neighbors, with each of our two feature sets.

Another option would be trying some other feature sets. In another paper we came across during our research, they used another dataset of dog images (Columbia Dogs Dataset), which contained 8350 images of 133 different breeds [1]. This dataset also contained annotations with labels, bounding boxes, as well as facial keypoints, which we did not have in our dataset. With this data, they were able to train a CNN to extract facial keypoints from other dogs, and create SIFT descriptors on these keypoints. We think this would be an interesting experiment, especially if we combine this approach with our current dataset which had much more data. This would be a semi-supervised learning process since the keypoint detector CNN could only be trained using the data with keypoint annotations. But we think this could perform well when training another CNN with the predicted keypoint SIFT descriptors of the images in our current dataset.

V. References

[1] W. LaRow, B. Mittl, and V. Singh, "Dog Breed Identification," *Stanford University*, 2016. [Online]. Available: https://web.stanford.edu/class/cs231a/prev_projects_2016/output%20(1).pdf.

[2] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao and Li Fei-Fei. Novel dataset for Fine-Grained Image Categorization. *First Workshop on Fine-Grained Visual Categorization (FGVC), IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[3] Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.

[4] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in PyTorch. In *NIPS-W*.

[5] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., & others (2011). Scikit-learn: Machine learning in Python*Journal of machine learning research, 12*(Oct), 2825–2830.