CS230

Pet Breed Classification on Altered Images

Anthony Xie
anthonyx@stanford.edu

Jake Lee
lakejee@stanford.edu

Kantapong Kotchum
kkotchum@stanford.edu

Abstract

Our project attempts to classify cats and dogs by breed in unusual settings: given corrupted images, can our model still infer the breed which the animal in the scene belongs to? We approach this task with four models: k-nearest neighbor, CNN, CNN with residual network, and transfer learning. Our evaluation metric shows that k-nearest neighbor achieves the best accuracy score of 98.34% on validation dataset while CNN with ResNet achieves the best accuracy score of 35.4%. The poor accuracy score of CNN's can be attributed to a variety of factors within the data and model architecture. Nonetheless, transfer learning method seems to be the most promising method as it avoids overfitting problem found in other models.

1 Introduction

Image classification is one of the central problems in Computer Vision. The type of classification that we are focusing on in particular is the identification of the breed of a dog or a cat in a picture. Recent SOTA models such as EffNet-L2 [1], BiT-L [2], and EfficientNet-B7 [3] achieve an accuracy rate of over 95% on the Oxford-IIIT Pets dataset. This dataset, however, has a limitation in that all the images presented are fine-grained. We propose that instead of training the models on ideal datasets, we *intentionally* alter the images such that some areas are obfuscated, blurred, or gray-scaled. We hope to explore various non CNN models, experimenting our own CNN models, as well as implementing a transfer learning on some of these SOTA models.

2 Related Work

This project uses the Oxford-IIIT Pet Dataset [4] which was first used by Ekin D. Cubuk et al in their paper *AutoAugment: Learning Augmentation Policies from Data* [5]. Using a search algorithm to find the best policy such that the neural network yields the highest validation accuracy on a target dataset, they attained a Top-1 accuracy of 83.5%. Pierre Foret et al use Sharpness-Aware Minimization (SAM) to seek parameters that lie in neighborhoods having uniformly low loss. Their research shows that SAM improves model generalization across a variety of datasets, one of them being the Oxford-IIIT Pet Dataset. Since CNNs are commonly developed at a fixed resource budget, Mingxing Tan and

Quoc V. Le systematically study model scaling and identify that carefully balancing network depth, width, and resolution via a scaling method called EfficientNet, which is 8.4 times smaller and 6.1 times faster on inference than the best existing CNN.

3 Dataset and Preprocessing

3.1 Dataset

We used the Oxford-IIIT Pet Dataset (Omkar M Parkhi and Andrea Vedaldi and Andrew Zisserman and C. V. Jawahar) [4]. Our pet training dataset consists of 7349 images of varying dimensions, and each image has an associated ground truth annotation of breed, head ROI, and pixel level trimap segmentation. There are 37 categories (breeds) of cats and dogs, each breed having around 200 images. A more detailed breakdown is provided as Figure 4 in the Appendix Section.

3.2 Preprocessing

The objective of our project is image classification, so the associated ground truth labels, head ROI, and trimap segmentations are disregarded. The 7349 images have a large variation in scale, pose and lighting, and not pre-split into training/validating/testing sets. For standardization, we set the height and width of each image to be 256 pixels and use the linear interpolation technique to compute the new RGB value for each pixel in our newly-generated images. We then run 3 different data augmentation methods to generate new data to help our model better generalize: grayscale, gaussian blur, and salt-and-pepper noise. An explanation on how each method works is provided in the Appendix. After preprocessing, we arrive at a total of 29536 images, with a small number of images being lost due to file corruption. Since there is no training, validating, testing label for the original dataset, we use 90:10 split ratio for training/validating.

4 Evaluation Metrics

Our objective is to classify an image into 37 classes. Suppose there are N data samples where the i-th sample is $(x^{(i)}, y^{(i)})$ such that $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \{1, 2, ..., 37\}$, and our model predicts the class the sample belongs as $\hat{y}^{(i)}$. Our evaluation metric is validation accuracy which, for our k-nearest neighbor model and CNN-based models, is computed as follows

For k-nearest neighbor, $\hat{y}^{(i)} \in \{1, 2, ..., 37\}$. The accuracy in training and testing will be computed based on how many data points are correctly assigned to the label of their clusters. That is,

$$Accuracy = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}[\hat{y}^{(i)} = y^{(i)}]$$

For CNN-based models, $\hat{y}^{(i)} \in [0, 1]^{37}$ where each entry in the vector represents the probability that $x^{(i)}$ belongs to that index (1-indexing). The accuracy in training and testing will be computed based on how many data points are correctly assigned to their true label, where the assignment is the index with the highest probability. That is,

$$Accuracy = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}[argmax(\hat{y}^{(i)}) = y^{(i)}]$$

Our model also computes cross-entropy loss, which is

$$Loss = -\sum_{i=1}^{N} \sum_{j=1}^{37} \log(\hat{y}^{(i)}[y^{(i)}])$$

We note that $\hat{y}^{(i)}[y^{(i)}] = \mathbb{P}(x^{(i)}, y^{(i)})$ is the probability that our model predicts $x^{(i)}$ belongs to the actual class $y^{(i)}$.

5 Methodology and Result

5.1 Initial test: K-nearest neighbor

While our focus within this project is centered around CNN's, we decided to experiment with K-nearest neighbor, which is a parametric and unsupervised machine learning algorithm. Each image (256 * 256 * 3) is flattened into a 196608 dimensional numpy array, and we use Euclidean Distance as a metric. We vary the number of neighbors k from $k = 2, 5, 10, 25, 100, 200, 500$. The table of training and validating accuracy is provided here:

k (number of neighbors)	Training Accuracy	Validating Accuracy
2	99.75	98.34
5	98.27	87.54
10	84.04	52.91
25	44.12	27.83
100	20.01	16.22
200	15.85	13.27
500	12.05	10.39

Figure 1: Accuracy of k Nearest Neighbor

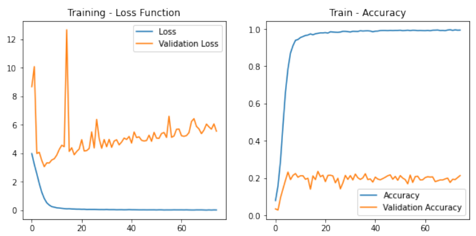
The training/validating accuracy is extremely high for small number of neighbors, with $k = 2$ achieving 99.75% training accuracy and 98.34% validating accuracy.

5.2 Baseline Model: Multi-layer CNN Model

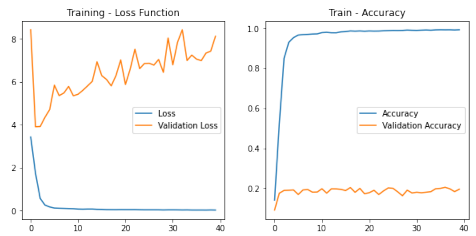
We began by coding a CNN model using Keras as our baseline model. Our current model is designed such that it consists of three main layers. In each layer, the input is sequentially

- fed into a 2D convolution layer with filter size 3 x 3, stride 1, and no padding
- fed into a batch normalization layer
- fed into a max pooling layer with filter size 2 x 2
- fed into a dropout layer

We use ReLU activation for all of the layers. The input to the first layer is the image that we want to train or test. The output from a dropout layer in the third main layer is then flattened and fed into a softmax activation layer with 37 classes. We train our model on only the original dataset (7349 images) and a full-sized dataset (29536 images).



(a) Data Subset



(b) Full Dataset

Figure 2: Loss and Accuracy of Multi-layered CNN

In a smaller dataset, we achieved a 99.4% training accuracy and 21.4% validating accuracy. When running on our full dataset, we had similar results: 99.2% training accuracy and a maximum validating accuracy of 20.4%.

These graphs showcase clearly that our model has incredibly high bias (high training accuracy but low validation accuracy), and is overfit to the training data. We can assume that the cause was also most likely our model architecture as opposed to the size/construction of the dataset, since increasing the dataset had little to no effect on the end validation accuracy result.

5.3 Single-layer CNN Model

Since we understood that our model architecture was too complex and overfit to the training data, we attempted to reduce the complexity of our architecture by reducing our CNN model from three main layers to just a single layer, and trained it again on our full dataset. Going from 3 Conv2D layers to just 1, we were able to reduce the number of trainable parameters from around 7.4M to 2.6M. We found however that this still did not reduce the issue of high bias, and this model was even less effective, as after training the model showed a 84.8% training accuracy and a maximum validating accuracy of 12.1%. While it is possible that retraining the model with additional epochs may have alleviated some part of the low accuracies, we believe that regardless of the epoch number the primary issue of low validation accuracy would not have been fixed. In addition, we tested other hyperparameter values for batch size, optimizer, and dropout rate, and found that very little changed.

5.4 CNN with Residual Blocks

Again in an effort to reduce the high bias of our model, we decided to implement residual blocks into our CNN model. Residual blocks allow for learned weights in previous layers to be reused in future layers. In this case we will do this by adding the result of our previous layer to the result of our next layer, and performing the activation on it. See Figure 5 in the Appendix for a diagram of a unit of residual block.

For our CNN model we will use a series of a single convolutional block followed by multiple identity blocks. Within each block there are 3 Conv2D layers, in addition to a skip connection. The identity block only does batch norm within the skip connection, keeping the dimensions the exact same, whereas the convolutional block does an additional Conv2D filter in addition to batch norm in the skip connection. Using these principles we constructed a CNN which after passing the image through an initial Conv2D layer, passes it through 3 stages of convolution and identity block sets, with an increasing number of filters at each stage. Overall, the model totals 5.9M trainable parameters (See Figure 6 in Appendix Section for full architecture)

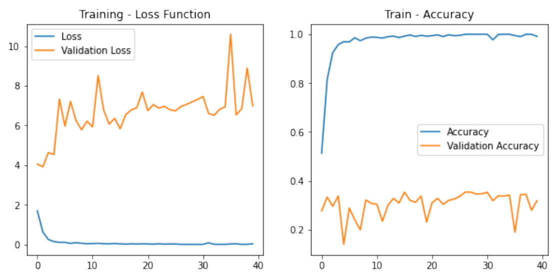


Figure 3: (Loss and Accuracy of CNN Model w/ Residual Blocks)

After training this residual block CNN model, we found our overfitting issues to still be present, with the training accuracy far exceeding the validation accuracy. However, the validation accuracy in this model was able to outperform the previous two models that we had coded, with a maximum validation accuracy of 35.4%.

5.5 CNN w/ Transfer Learning

The final model that we utilized was to leverage existing CNN ResNet models in order to bolster the accuracy of our own. We use ResNet50 model with pretrained weights from ImageNet and freeze all but the last 15 layers. We then add 3 main layers (each main layer consists of a dense layer, a dropout layer, and a batch normalization layer) and a final softmax activation layer with 37 classes. While due to technical issues we were unable to finish running the training for this model, we could see that through each epochs both the training and validation accuracy increased at around the same rate and the overfitting issue did not seem to be present. That is, at epoch 10 we saw the training accuracy to be around 17% and validation accuracy of around 14%. This generalizes significantly better than our other models, all of which had a difference of >50% between training and validation accuracy before the 10th epoch.

6 Analysis and Future Work

Given the results of the different models that we have tested and trained, we believe we can identify a few key causes the performace of KNN and CNN-based models.

For KNN, a small number of neighbors results in an extremely large accuracy for both training and validating dataset because of the way we augment the data. That is, our generated images resemble the original images, only varying slightly for some pixels (See Appendix, Figure 7).

With small number of neighbors, it is very likely that any image will be mapped to its augmented version, hence high accuracy score. As the number of neighbor increases, KNN performs poorer than the CNN-based model, which is explainable due to the high dimension of the input and how KNN does not learn the relationship among nearby pixels (the model only computes Euclidean distance **directly**, pixel-by-pixel).

As for CNN-based model and its low performance, we speculate that even with augmented data we still do not have enough images for each class. After preprocessing, each class still has less than 1000 images which helps explain why the model overfits within a few epochs of training. This is also compounded with the fact that for a majority of our models we were training our models from scratch with only our provided data rather than leveraging existing pre-trained weights, therefore our models will struggle to generalize well, especially considering that our augmented data retains many of the same elements as the original data and doesn't differentiate nearly as much as a new image.

Our suggestion for future work is to

1. leverage the power of transfer learning, since most SOTA models like ResNet50 were trained on a much larger sample size of dataset, resulting in better generalization
2. increase the number of data. One method is to find more image processing techniques to augment our dataset such as obfuscation, rotation, or color masking. Another method is to webscrape images of cats and dogs and manually label them.

7 Contributions

- **Anthony:** Data preprocessing/augmentation coding, coded + trained multi-layered, single-layered, and residual block CNNs, project writeup, video script
- **Jake:** Data preprocessing/augmentation coding, project writeup, poster creation, video script

- **Mark:** Data downloading, basic CNN and residual block CNN model research, wrote code for KNN, experimented with SVM (did not use due to O(N^3) runtime complexity), experiment with transfer learning (ResNet 50), project writeup

References

[1] Pierre Foret, et al. "Sharpness-Aware Minimization for Efficiently Improving Generalization." Google Research - Brain Team, ICLR 2021. `https://arxiv.org/pdf/2010.01412v3.pdf`

[2] Alexander Kolesnikov, et al. "Big Transfer (BiT): General Visual Representation Learning." Google Research - Brain Team, 2016, `https://arxiv.org/pdf/1912.11370.pdf`

[3] Mingxing Tan, Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." ICML 2019. `https://arxiv.org/pdf/1905.11946.pdf`

[4] Parkhi, Omkar M, et al. "The Oxford-IIIT Pet Dataset." Visual Geometry Group - University of Oxford, University of Oxford, 2012, `https://www.robots.ox.ac.uk/~vgg/data/pets/`.

[5] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, Quoc V. Le. "AutoAugment: Learning Augmentation Policies from Data." CVPR 2019, `https://arxiv.org/abs/1805.09501`

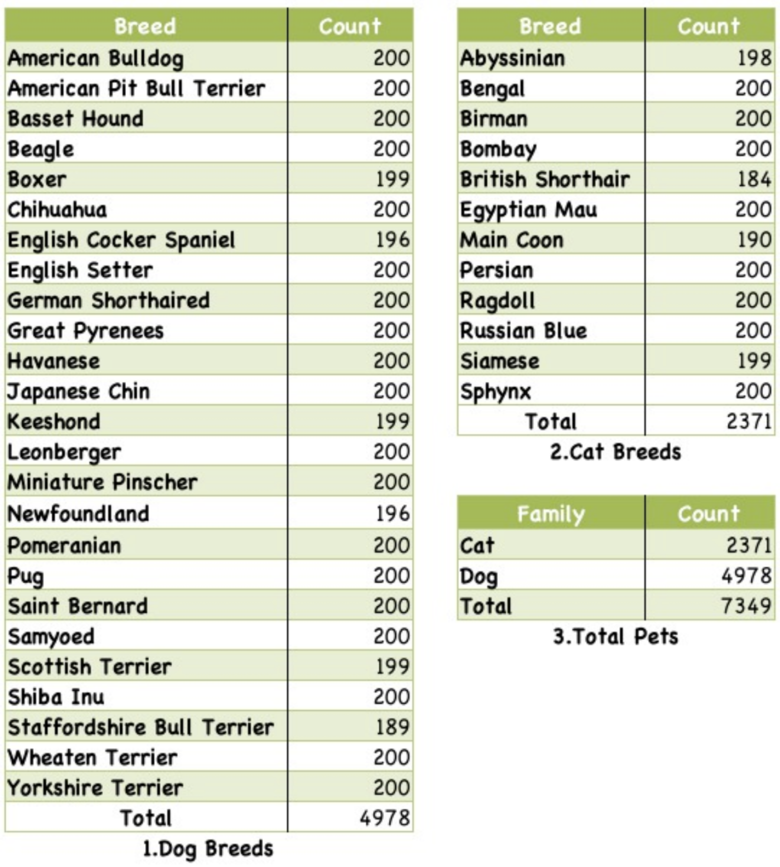
[6] scikit-learn, `https://scikit-learn.org/stable/`

[7] keras, `https://keras.io/`

8 Appendix

A description of each preprocessing techniques is provided below

1. Grayscaled operation: the value of each pixel is a single sample representing a certain amount of light. This is achieved by using torchvision's Grayscale function in transforms, converting the 3 channel RGB image into a 3 channel grayscale image.
2. Gaussian Blur operation: the image is convolved with a Gaussian filter instead of a box filter. In our implementation, we apply a Gaussian filter of size 5 x 5 and with a standard deviation of 2 along the x-axis and 3 along the y-axis.
3. Salt-and-Pepper noise operation: SNP is an impulse type of noise in images. An area contains 'salt' noise if there is a white dot in the dark region of an image and contains 'pepper' noise if there is a black dot in the bright region of an image. We generate salt-and-pepper noise with a probability of 0.05 to convert each pixel into a noised one.



Breed	Count
American Bulldog	200
American Pit Bull Terrier	200
Basset Hound	200
Beagle	200
Boxer	199
Chihuahua	200
English Cocker Spaniel	196
English Setter	200
German Shorthaired	200
Great Pyrenees	200
Havanese	200
Japanese Chin	200
Keeshond	199
Leonberger	200
Miniature Pinscher	200
Newfoundland	196
Pomeranian	200
Pug	200
Saint Bernard	200
Samyoed	200
Scottish Terrier	199
Shiba Inu	200
Staffordshire Bull Terrier	189
Wheaten Terrier	200
Yorkshire Terrier	200
Total	4978

1.Dog Breeds

Breed	Count
Abyssinian	198
Bengal	200
Birman	200
Bombay	200
British Shorthair	184
Egyptian Mau	200
Main Coon	190
Persian	200
Ragdoll	200
Russian Blue	200
Siamese	199
Sphynx	200
Total	2371

2.Cat Breeds

Family	Count
Cat	2371
Dog	4978
Total	7349

3.Total Pets

Figure 4: (1) Dog breeds, along with image counts, (2) Cat breeds, along with image counts, (3) Animal breed, along with image counts

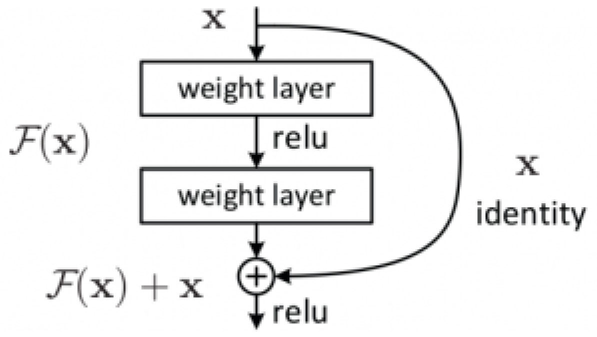

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
x
identity

Figure 5: Residual Block Diagram

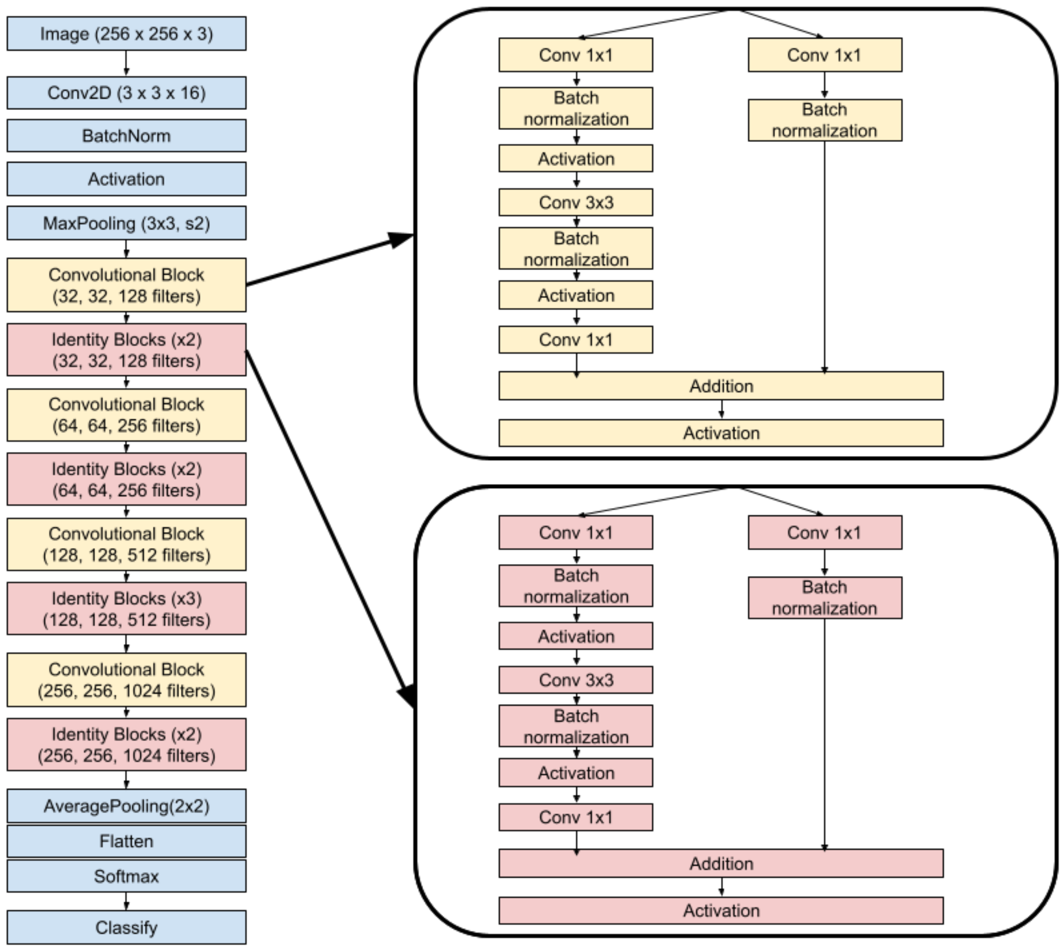


Figure 6: CNN with Residual Block Architecture

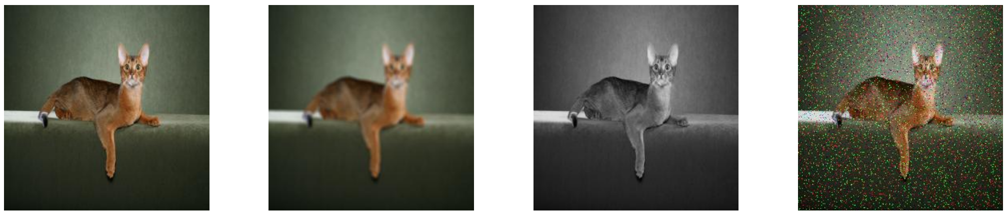

(a) Original (b) Blurred (c) Grayscaled (d) Salt and Pepper

Figure 7: Four Samples of an Abyssinian Cat

#	**Notable Experiment Details**	**Train., Val. Acc.**
1	K-nearest neighbor (k = 2)	99.75%, 98.34%
2	K-nearest neighbor (k = 25)	44.12%, 27.83%
3	K-nearest neighbor (k = 500)	12.05%, 10.39%
4	Multi-layer CNN Model; 3 layers;	99.2%, 20.4%
5	Single-layer CNN Model; Attempt to reduce overfitting and complexity	84.8%, 12.1%
6	CNN with Residual Blocks; Attempt to reduce high bias; Each block $\rightarrow$ 3 Conv2D Layers	99.9%, 35.4%
7	CNN with Transfer Learning; ResNet50 model with pretrained weight from ImageNet; Technical issues	17%, 14%

Figure 8: Certain Observations