CS63 Spring 2017
Animal Photo Classification with Neural Nets

Leah Brumgard and Zoe Kyaw

May 8th, 2017

1 Introduction

The problem that we tackled for this project was the expired competition ”Dogs vs Cats” from Kaggle (https://www.kaggle.com/c/dogs-vs-cats). The goal of this competition was to train a machine learning algorithm that takes in a photo and classifies it as either a cat or a dog. We decided to use neural networks through Keras as our machine learning algorithm. We used convolutional networks to train a model for image classification.

Neural networks are typically used for image classification, because deep neural networks with many layers have proven to do well with large data sets. Additionally, using neural networks allowed us to use convolutional layers, which fundamentally serve as feature detectors for image classification.

2 Method and Details

The data set that we used consisted of 25,000 jpg images of cats and dogs, from which we used 20,024 images for training. We preprocessed the images by padding smaller images with a black border so that all of the images were a consistent size. All of the final preprocessed images were 768 pixels by 1050 pixels and had all 3 channels of RGB colors. We also normalized the RGB colors, assuming that the minimum value was 0 and the maximum value was 255. After preprocessing all of the images, we wrote them into 40 separate outfiles containing the shuffled photos, along with one file containing the corresponding labels for each photo.

We saved the preprocessed images into 40 separate files to avoid technical issues, since we are working with such a large data set, and also to simplify the splitting of our data for cross validation. We decided to test our algorithm using the cross validation method. Originally, we tested using 5 folds, which resulted in fairly low accuracy, likely because with roughly 20000 images and 5 folds, our cross validation algorithm was training on 16000 images and testing on 4000 images. Therefore, the algorithm was being tested against a fairly large amount of unseen images, resulting in a low accuracy. In our final rounds of testing, we increased the number of folds to 10, significantly improving our accuracy, because with 10 folds and approximately 20000 images, the algorithm was being trained on 18000 images and tested on only 2000 images. This seems to be a more reasonable number of unseen images to test the neural net algorithm on, because with 5 folds our average accuracy was less than 50 percent, and with the increase to 10 folds our accuracy increased to around 90 percent.

For our neural network structure, we used layer pairs that we expected to work well together after doing research on image classification with neural networks. We decided to use pairs of convolutional layers, which act as a sort of feature detector for image classification, along with pooling layers, which compress the data using either the average or maximum value.

We found that using an average pooling layer for our very first layer worked best, because it compressed the data from the images by averaging the pixel values. We then used 3 consecutive pairs of convolutional layers along with maximum pooling layers.

After the pairs of convolutional and pooling layers, we used a flatten layer to put all of the data into a one-dimensional layer. We then used a dense layer followed by a dropout layer which dropped out 50 percent of the nodes, which is essentially a type of ensemble learning, because it means that the neural network is seeing different combinations of the nodes from the previous dense layer every time.

The final layer of our network was a dense layer with two output nodes and a sigmoid activation function, so that the final output of our network matched the expected label format of a two-number tuple. The sigmoid activation function was used so that the output numbers ranged between 0 and 1, and we then chose the maximum value from the tuple as the final image label.

3 Results

While experimenting with our neural net structure, we ended up using a RELU activation function for only non-output layers, because using RELU on the output layer dropped our accuracy significantly. We found that a sigmoid activation function made more sense for our output layer, and increased accuracy significantly. This makes sense because the output labels for our categorization were [0,1] for a dog and [1,0] for a cat, and a sigmoid activation function on the output layer allowed our neural net to output floats between 0 and 1 for each number in the output tuple. The final classification label is then dog if the second number in the tuple is the maximum, and cat if the first number in the tuple is the maximum.

We had several experiments:

- 1. 1000 images with 50 epochs and 5 folds. (nohupSample100.out)
- 2. 40 files (500 images per file) with 15 epochs and 5 folds. (final.out)
- 3. 20 files (500 images per file) with 10 epochs and 10 folds. (20FilesWith10Epochs.out)
- 4. 10 files (500 images per file) with 10 epochs and 10 folds. (10Folds10Files10Epochs.out)
- 5. 3 files (500 images per file) with 3 epochs and 3 folds and different numbers of convolutional layers. (3Files3Folds2Conv.out, 3Files3Folds3Conv.out, 3Files3Folds4Conv.out)

- 6. 3 files (500 images per file) with 3 epochs and 3 folds and different numbers of dropout layers. (3Files3Folds0Drop.out, 3Files3Folds1Drop.out, 3Files3Folds2Drop.out, 3Files3Folds3Drop.out)
- 7. 10 files (500 images per file) with 5 epochs and different numbers of folds. (10Files10Folds5Epochs.out, 10Files5Folds5Epochs.out, 10Files2Folds5Epochs.out)

We initially wanted to run our neural network with 1000 images, partly to make sure that our cross validation and neural network code didn't break but also to conduct a short experiment on a small dataset. We made it run for 50 epochs, because we were only giving it a small amount of data. For later experiments, we would be adding more input data and training on fewer epochs. This experiment was really successful, because the accuracy after testing was really high (average accuracy over folds 0.994).

Because of this, we decided to run another test with all files for 15 epochs and 5 folds. If you look at the results in final.out, the accuracy was really high during the training (high 90's) but dropped a lot during the testing period (low 30's). We think this was partly due to the fact that we had so many epochs that the network just memorized everything and couldn't generalize to the unseen data. The cross validation results suggested that over-fitting was occurring, so in our next experiment, we decreased the number of epochs to 10. But we also increased the folds to 10, because we were worried that the network was not given enough training data but instead was given too much testing data.

We reran our program with 20 files (half of the dataset) with 10 epochs and 10 folds. We got through the first fold before the power outage. The accuracy over the entire first fold is 99% (0.993999999523). Although we only got through the first fold, this was a huge improvement from the first experiment, because we were able to attain a much higher accuracy.

Table 1: Experiments with different numbers of convolutional layers. 3 Folds 3 Files 3 Epochs.

	Number of Convolutional Layers		
	2	3	4
Fold 0 Acc	63.4	61.8	61.8
Fold 1 Acc	95.2	63.4	69.0
Fold 2 Acc	99.4	75.2	77.2
Average Acc	86.0	66.8	69.3

In this experiment, we ran 3 different trials using 10 data files (each with 500 images), consistently using 3 folds and 3 epochs. We changed the number of convolutional layers, using 2, 3, and 4 convolutional layers each immediately followed by max-pooling layers. All convolutional layers used a RELU activation function.

Table 2: Experiments with different numbers of dropout layers. 3 Folds 3 Files 3 Epochs.

	Number of Dropout Layers			
	0	1	2	3
Fold 0 Acc	62.4	64.4	61.8	61.8
Fold 1 Acc	83.6	86.6	61.8	61.8
Fold 2 Acc	97.8	95.6	62.6	61.8
Average Acc	81.3	82.2	62.1	61.8

In this experiment, we ran 3 different trials using 10 data files (each with 500 images), consistently using 3 folds and 3 epochs. We changed the number of dropout layers, using 0, 1, 2, and 3 dropout layers each immediately followed by a dense layer. Each dropout layer had a dropout percentage of 50% and all dense layers used a RELU activation function, except for the final dense output layer, which used a sigmoid activation function.

Table 3: Experiments with different numbers of folds. 5 Epochs 10 Files.

	Number of Folds		
	2	5	10
Fold 0 Acc	81.0	94.6	85.8
Fold 1 Acc	96.0	97.4	99.8
Fold 2 Acc	-	97.2	100.0
Fold 3 Acc	-	96.8	100.0
Fold 4 Acc	-	97.6	
Fold 5 Acc	-	-	
Fold 6 Acc	-	-	
Fold 7 Acc	-	-	
Fold 8 Acc	-	-	
Fold 9 Acc	-	-	
Average Acc	88.5	96.7	

In this experiment, we ran 3 different trials using 10 data files (each with 500 images), consistently using 5 epochs, 3 convolutional layers, and 1 dropout layer. We changed the number of folds used in our cross-validation training/testing method, using 2, 5, and 10 folds. We expected to see a difference in the accuracies yielded from these trials, because with less folds, the network will be trained on less images and tested on more unseen images. In these trials, each convolutional layer used a RELU activation function, and the dropout layer used a dropout percentage of 50%.

4 Conclusions

We would like to continue experimenting with the dropout layers' inputs. We currently have 0.5 as the dropout percent, but we would've liked to see how changing the dropout percentage affects the classification.

It would also have been interesting to find out how different methods of pre-processing affect how well classification is. We used padding, but perhaps resizing the images would have yielded better results faster.

Due to time constraints, we also could not run our experiment on all 20,024 images, but instead only ran the program on 1/4 or 1/2 of the data set. In the future, we would like to use the entire data set to create a stronger neural network.