Binary Image Classification Using Deep Learning

Michael Griese
mjgriese@email.wm.edu
College of William and Mary
Williamsburg, Virginia

ABSTRACT

Image classification, the most raw form of computer vision, is a highly desirable capability in a number of modern fields ranging from autonomous vehicles to satellite image analysis. Image classification is also a prime candidate for deep learning as there exists abstract divide between different classes of images with no mathematically deterministic way to separate them by hand. In this paper, a neural network will be used to classify images of household pets into the two classes of either "cat" or "dog".

CCS CONCEPTS

• **Theory of computation → Machine learning theory**.

KEYWORDS

neural networks, binary classification, deep learning,

ACM Reference Format:
Michael Griese. 2021. Binary Image Classification Using Deep Learning. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The strength of deep learning comes from the implicit results that follow from the architecture that is created. This model will make use of several convolutional layers, as well as a standard densely connected layer to classify images as either a cat or a dog.

The model presented is generally based on the AlexNet image classifier which also makes use of several convolutional layers before a densely connected layer completes the task of classification. In order to achieve accurate classification, the model presented will also make use of image manipulation to increase the range of training data, dynamic learning rate to allow for fine detailed learning, as well as a modified implementation of gradient descent that can provide improvements to training times.

It will be shown that the resulting network is capable of high accuracy identification of the two output classes after relatively minimal training. Additionally, it is unlikely that this network has been pushed to its limits entirely and could still see further improvements, as will be shown later.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 MODEL OVERVIEW

This model was created to complete the binary classification task of separating images of cats and dogs. The model consists of several sequential layers. The first three layers are convolutional layers which exist to identify broader structures and features within the image. After these three layers, there is a fully connected layer which interprets these convolutional layers and acts as the main source of classification. Finally the output layer returns the probability of the two classes, the higher of which is taken to be the final result.

The model is implemented using the keras python deep learning API. Keras allows the user to define a neural network with preexisting features, allowing the user to focus more on fine tuning the model rather than the exact implementation of the model. This allowed the model in this paper to use features that would otherwise require greater time and expertise to implement.

2.1 Convolutional Layers

The purpose of the three sequential convolutional layers is to allow the network to learn how to extract defining features of the images to assist in classification. Each convolutional layer follows the same procedure with differing kernel sizes and output dimensions. First, the data is sent into the convolutional layer where the filter is applied. All convolutional layers used the rectified linear unit "relu" activation function. After the filter is applied, the layer is normalized before max pooling is performed. The max pooling allows dominant features to become more visible and removed less important features. After the max pooling, a dropout stage is performed before the data moves to the next layer. The dropout forces the network to become less dependent on any single node value, which encourages the model to view the features on a larger scale rather than limiting itself to a particular region of the image.

Each successive convolutional layer is given a greater number of filters for two reasons. The first being that each layer is searching for more minute features. If the number of filters were to be kept constant, the number of nuanced features that could be recognised would be reduced. Additionally, after each successive round of max pooling the dimensionality of the data is halved. This allows the number of filters to be increased with minimal performance loss, which is of great importance with respect to the amount of time required to train the model for large numbers of epochs.

In the implementation, the image flows through three convolutional layers before classification occurs. The first convolutional layer consists of 32 filters. The size of this filter and the filter for all of the convolutional layers is a square 3 pixels wide and tall. These filters serve to find the largest features in the image. After normalization and max pooling with square pools of size 2, the data is passed to the second convolutional layer. This layer consists

of 64 filters. Finally, after a second round of normalization and max pooling, the data is passed to the third convolutional layer. This layer consists of 128 filters and is the final major step before classification.

2.2 Densely Connected Layer

The final layer of the model is a densely connected layer consisting of 500 nodes. As input, this layer receives the final results of the third convolutional layer after normalization and max pooling. However, this data must first be flattened into a single vector before it is passed to the dense layer. The output of each convolutional layer is a 3 dimensional tensor with dimensions $[l/2^n, l/2^n, f]$ where l is the length of the input image, 128 in this case. n is the number of convolutional layers and therefore max pooling layers that have been passed through. Finally f is the number of filters of the most recent convolutional layer. Following this pattern, the data before the flattening process is of dimension [16, 16, 128]. Following the flattening process, the input to the final dense layer is a single vector, x, of dimension 32,768. The dense layer functions as a traditional layer whose output is dictated by (1) where w represents the weights of the dense layer.

$$\hat{y} = x \cdot w \tag{1}$$

2.3 Final Layer

The final layer of the model is a single densely connected layer consisting of 2 nodes. These nodes represent the two possible output classes. Unlike the other layers, this layer uses a softmax activation function.

$$\sigma(\mathbb{Z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } 1, \ldots, K \text{ and } \mathbb{Z} \in (z_1, \ldots, z_k) \in \mathbb{R}^k \tag{2}$$

The softmax function, as defined in (2), serves to convert the generated output of the previous layers into a probabilistic ratio or "confidence" rating. The softmax function ensures the sum of all elements in the vector equals 1, which is what allows the value of individual dimensions to be considered as part of a probability distribution.

The final choice of the network is the class which achieves the highest value after the softmax is applied.

3 DATA SET

The dataset that was used when training this model was acquired from a Kaggle competition that is several years old. The dataset consisted of 25,000 images that followed a naming convention that allowed the class of the image to be extracted. All images that were of cats began with the keyword "cat" and likewise for dogs.

The dataset was partitioned into three separate datasets consisting of the training, validation, and testing sets. The training set consisted of 16,000 images split evenly with 8,000 images of each class. The validation set consisted of 4,000 images which were likewise evenly split. Finally, the testing set contained 5,000 images evenly split between the classes.

The raw images within each dataset are of various dimensions. Each image was resized to 128 pixels wide by 128 pixels tall before being fed into the network. This resulted in 49,152 input values into the network after accommodating the 3 channels for the RGB values of each pixel.

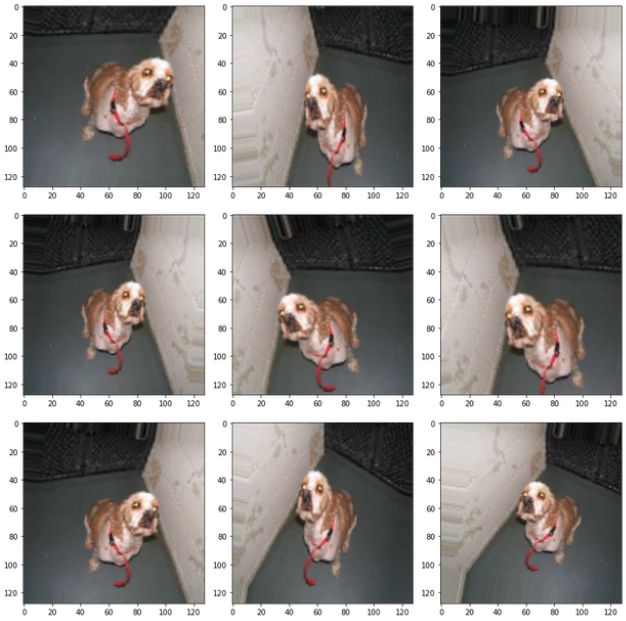

Figure 1: Example image deformations within the training set.

3.1 Training Deformations

A common method of artificially increasing the size of the training data for image classification involves the modification of the training images to create "new" images of the same class. The effects of this are twofold. Firstly, this simply increases the number of images upon which the model can be trained. Secondly, the distortions in the images allow the convolutional layers to, over time, more accurately extract features from images as the images they are exposed to are more diverse than the base training set.

The deformations used while training this model included image rotation, shearing, zooming or cropping, horizontal mirroring, and vertical and horizontal shifting. With these seven additional parameters, the number of trainable images is increased dramatically.

4 TRAINING PROCESS

The model was trained using an Intel i7-6700HQ and the final training session lasted approximately 90 minutes.

4.1 Dynamic Hyper-parameters

By using the Keras API, there are a number of callback functions that will trigger at different points during the training regiment, allowing the hyper-parameters to be changed during the training loop based on the training progress that has been so far achieved. The model used two such callbacks the "EarlyStopping" and "ReduceLROnPlateau" callbacks.

The EarlyStopping callback is created with a simple integer threshold. If the threshold number of epochs passes without the

validation error improving, the network terminates and skips the remaining epochs to avoid overfitting to the training data. A threshold of 10 epochs was used for this model. The training occurred over a maximum defined period of 50 epochs. Each epoch consisted of 1066 training samples, an arbitrary number that was chosen as a fraction of the main training set. During the training of the model, there was no instance where the training continued past 35 epochs due to the EarlyStopping callback. This implies that the model reaches near total accuracy by epoch 25, as there must be no improvement for 10 epochs to terminate early.

The ReduceLROnPlateau callback serves to modify the learning rate as the model begins to converge on a solution. The model begins with a learning rate of 0.001, a highly conservative value due to the high number of max epochs. This callback is triggered when the validation accuracy does not increase for 2 epochs with the assumption that smaller shifts can still result in minor improvements to classification. Each time the callback is triggered, the learning rate is halved to a minimum of 1×10^{-5}.

4.2 Loss Function

The model uses cross-entropy error between the generated result and the desired classification. This loss function was chosen because of its convex nature, allowing gradient descent techniques to smoothly reach a minimum. Additionally, it will punish misclassified training examples more harshly.

4.3 Weight Optimization

The model used a variation of standard stochastic gradient descent called RMSprop. RMSprop, root mean square propigation, exists to more effectively apply the effects of gradient descent. The method is still based around the calculation of the gradient and then moving in weight space in the calculated direction, though the magnitude is not only dependent on the learning rate. RMSprop uses the gradient for direction but adjusts the magnitude of an update step individually for each weight. This allows RMSprop to more quickly find minimums when compared with standard gradient descent. Within Keras, RMSProp maintains a moving average of the square of previous gradients. The gradient at any given step is then divided by the square root of this moving average gradient. The resulting gradient is one that is equal in direction but with an augmented magnitude to either increase the rate of update when near a minimum or decrease it in the event that the model begins to wander. The final net result of this weight optimization style is a weight optimizer that is not following the static learning rate blindly for every weight. By untethering each individual weight from the global learning rate, the weight optimization is allowed to adjust each weight by a more effective amount each step which accelerates the learning process.

5 RESULTS

A model was trained and achieved over 90% accuracy over the full testing set, placing it well above any possibility of accidental chance.

5.1 Training Results

From Keras a plot can be generated which graphs the performance of the model over the training duration. On each plot, the blue line represents the performance within the training data set and the red line represents the performance within the validation set.

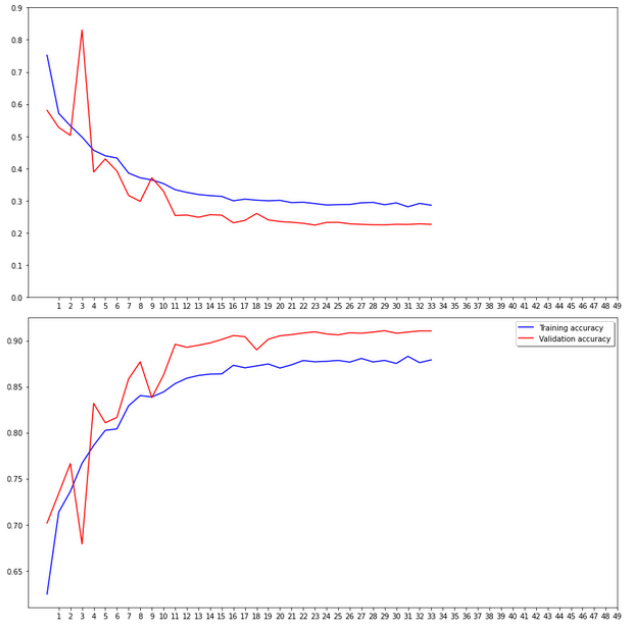

Figure 2: Loss over epochs and Accuracy over epochs

The top plot is the plot of loss against epochs. The loss begins at roughly .55 as the network is initialized with random weights. It is interesting to note that the loss during the third epoch drastically increases within the validation set to as high as .8; this is likely due to the network only guessing one of the two possible classes after over correcting from one of the previous two epochs. The early stages of learning are quite volatile but by the 11^{th} epoch the loss in both the training and validation sets becomes quite stable. From this point onward, the model optimizes only slightly further- for approximately 12 more epochs until epoch 23. At this point, the loss is nearly constant and the early stopping callback threshold is reached after which training terminates after only 33 epochs.

The second plot is a plot of the accuracy over epochs. In almost all ways it is merely a mirrored version of the loss plot. One observation is that during epoch 31 there was a sudden slight increase to the in sample training accuracy, as well as a slight increase to the validation accuracy. This suggests that it may be possible to further train the network with greater patience in the EarlyStopping callback to prevent the early termination. This could result in a few more images being properly classified in the final test set.

5.2 Testing Results

The model was quite powerful even after training for only 33 epochs. This amount of training required 90 minutes and achieved a total accuracy of 91.37%, misclassifying only 431 out of the 5,000 test images. The network has indeed achieved the desired result of identifying the overarching structures of cats and dogs in 2d images.

Figure 3: Example images with their file name and the classification assigned by the network

Network Performance		
Incorrectly Classified	Correctly Classified	Percentage Correctly Classified
431	4568	91.37%

6 CONCLUSION

Using the Keras python API for deep learning, a model was created with the intent of classifying images of cats and dogs. The model relies on three convolutional layers of increasing filters, followed by a round of max pooling after each, in order to extract the features of the image that is passed to the network. From these features, a densely connected layer is used to classify the image into the final two binary classes. This structure was heavily influenced by that of AlexNet, as the structure used for this task was effectively a version of AlexNet with fewer layers of smaller dimension. Just like AlexNet, this model performed incredibly well at its task and was able to classify images with over 90% accuracy after just 30 epochs of training, requiring only around 90 minutes to train. The strong performance of this network shows the potential that exists within the field of deep learning in completing complex tasks with minimal explicit instruction.

ACKNOWLEDGMENTS

To Winston, for grading my PS0 even though it was late. To Professor White, for doing an excellent job of explaining these concepts during the lecture. Finally my mother, who provided me with many cookies to eat while completing this project.