Cats vs Dogs Image Classification

Arjun Srinivasan Ambalam, Praveen Menaka Sekar, Arun Kumar Dhandayuthabani

1 Introduction

The Dogs vs. Cats dataset is a standard computer vision dataset that involves classifying photos as either containing a dog or cat. The training dataset contains 25,000 images, including 12,500 images of dogs and 12,500 images of cats, while the test dataset contains 12,500 images. The average size for these images is around 350500. Our learning task is to learn a classification model to determine the decision boundary for the training dataset. As this classification problem is a binary classification, we have written a custom Convolutional Neural Network (CNN) architecture for this project as we think it is very much sufficient to obtain a good classification performance.

2 Data Preprocessing

The given images are of different sizes and will have to be reshaped prior to modeling so that all images have the same shape. This is often a small square image. We choose $200 \times 200 \times 3$ as our image shape for our network. The 25,000 images with 200x200x3 pixels each, or 3,000,000,000 32-bit pixel values. Hence to make data loading easy We load all of the images, reshape them, normalize them to [0, 1], and store them as a single NumPy array.

3 Convolutional Neural Network (CNN)

3.1 Introduction

1. CNN processes images using matrixes of weights called kernels/filters (features detectors) that detect specific features present in an image such as vertical edges, horizontal edges, etc.
2. Moreover, as the image progresses through each layer, the kernels/filters are able to recognize more complex features.
3. A CNN consists of input layer, convolutional layer, pooling layer, fully connected layer, and output layer.
4. Input layer takes a matrix of pixel values of shape (width, height, channels).
5. Convolutional layers contain many feature maps/filters/kernels, which are two dimensional hidden nodes. Every feature maps/filters/kernels is constructed using weight matrix, and different feature maps/filters/kernels owns different weight matrix.
6. Kernels do convolutional operation with every feature map in previous layer (layer j), then we sum them up and put it into an activation function.
7. With feature maps/filters/kernels, we can learn different features of data and the amount of parameters is not increased exponentially with the number of hidden nodes and layers.
8. Pooling layers do down sampling operation on feature maps. For every n pixels, we only retain the max value, so that the size of feature maps will be half of the original size.

9. Down sampling reduces the resolution of feature maps and reduces the sensitivity of the output to shifts and distortions, so that the model will be more robust.

10. In a fully connected layer, we flatten the output of the last convolution layer and connect every node of the current layer with the other node of the next layer. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks and work in a similar way.

11. The output layer of CNN will compute the class probability scores.

3.2 Model Architecture 1

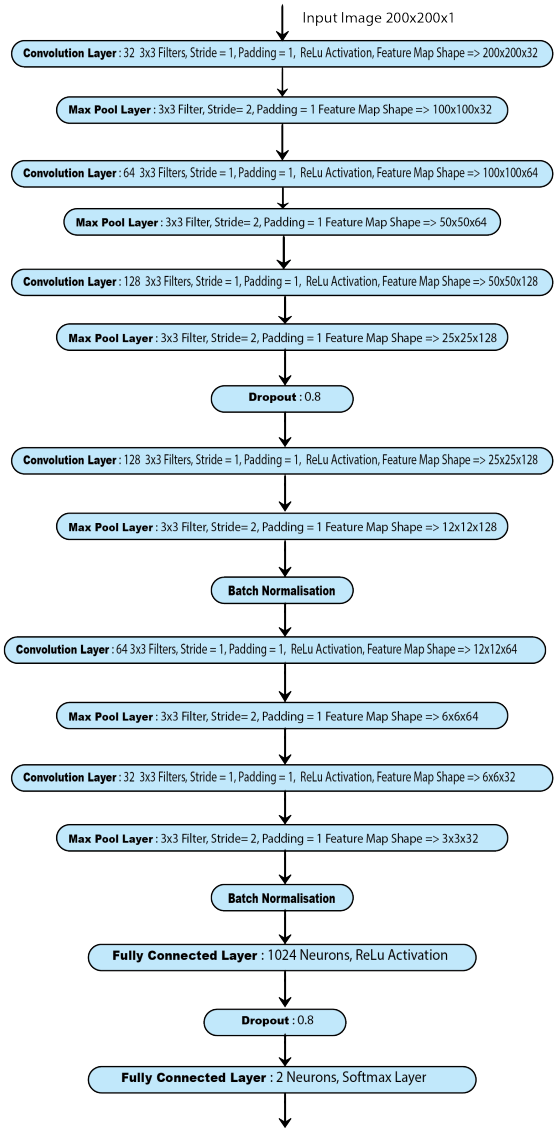

Figure 1: Convolutional Neural Network Model Architecture

The above given diagram describes the model architecture that we used for our classification task. Our CNN architecture comprises of following layers,

1. Convolution Layer with 32 kernels of size 3×3 and ReLu activation.
2. Max Pool Layer with kernel of size 3×3.
3. Convolution Layer with 64 kernels of size 3×3 and ReLu activation.
4. Max Pool Layer with kernel of size 3×3.
5. Convolution Layer with 128 kernels of size 3×3 and ReLu activation.
6. Max Pool Layer with kernel of size 3×3.
7. Dropout Layer with 0.8 dropout parameter.
8. Convolution Layer with 128 kernels of size 3×3 and ReLu activation.
9. Max Pool Layer with kernel of size 3×3.
10. Batch Normalization Layer
11. Convolution Layer with 64 kernels of size 3×3 and ReLu activation.
12. Max Pool Layer with kernel of size 3×3.
13. Convolution Layer with 32 kernels of size 3×3 and ReLu activation.
14. Max Pool Layer with kernel of size 3×3.
15. Batch Normalization Layer.
16. Fully Connected Layer with 1024 Neurons and ReLu activation.
17. Dropout Layer with 0.8 dropout parameter.
18. Fully Connected Layer with 2 Neurons Softmax Layer

3.3 Model Accuracy and Loss

- Number of Epochs: 15
- Learning rate: 0.0001
- Training Accuracy: 92.85 %
- Validation Accuracy: 86 %

```
In [4]: model = validate_train_set(c_net,train_set)

Training Step: 5279  | total loss: 0.22351 | time: 327.883s
| Adam | epoch: 015 | loss: 0.22351 - acc: 0.9257 -- iter: 22464/22500
Training Step: 5280  | total loss: 0.21549 | time: 338.628s
| Adam | epoch: 015 | loss: 0.21549 - acc: 0.9285 | val_loss: 0.35602 - val_acc: 0.8600 -- iter: 22500/22500
--
```

Figure 2: Training, Validation Loss and Accuracy vs Epochs

Though the above model resulted in a good classification performance, we wanted to improve our performance further. Hence, we incorporated data augmentation into our model architecture, and trained our dataset using this model in gpu to obtain better performance.

3.4 Model Architecture 2

This is the COLAB link for Notebook: Model2-GPU.
This is the drive link for restructured data: Data.

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_28 (Conv2D)           (None, 67, 67, 32)        896
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 22, 22, 32)        0
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 8, 8, 64)          18496
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
dropout_4 (Dropout)          (None, 2, 2, 64)          0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 1, 1, 64)          36928
_________________________________________________________________
flatten_2 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 1024)              66560
_________________________________________________________________
dropout_5 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_7 (Dense)              (None, 256)               262400
_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 257
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0
=================================================================
Total params: 385,537
Trainable params: 385,537
Non-trainable params: 0
```

Figure 3: Convolutional Neural Network Model Architecture

3.5 Model Accuracy and Loss

- Number of Epochs: 1
- Learning rate: 0.0001
- Training Accuracy: 90.52 %

4 Observation

1. With the increase in the resolution of the Image, the number of parameters becomes huge. With that many parameters, it's difficult to get enough data to prevent a neural network from overfitting, thereby leading to implementation of a Convolutional Neural Network.
2. Before we normalized the inputs, the weights associated with these inputs would vary a lot. To accommodate this range difference between the features some weights would have to be large and then some have to be small. If we have larger weights then the updates associated with the back-propagation

would also be large and vice versa. Because of this uneven distribution of weights for the inputs, the learning algorithm keeps oscillating in the plateau region before it finds the global minima. Therefore, we normalize the input features such that all the features would be on the same scale and the weights associated with them would also be on the same scale thus helping the network to train faster.

3. Data augmentation enables us to increase the diversity of data available for training models, without actually collecting new data, thereby helping the model to learn all possible scenarios and enhancing it's performance.

4. Each layer in out CNN architecture display hierarchical nature of the features in the network. Layer 1 learns some basic edges and colors. Layer 2 responds to corners and other edge/color conjunctions. Layer 3 capture complex textures or patterns. Layer 4 shows class specific variations such as dog and cat faces. Layer 5 tries to discover pose variations present and so on.

5. Dropout regularization is a computationally cheap way to regularize a deep neural network and prevent overfitting. Dropout works by probabilistically removing, or "dropping out," inputs to a layer, which may be input variables in the data sample or activations from a previous layer. It has the effect of simulating a large number of networks with very different network structures and, in turn, making nodes in the network generally more robust to the inputs.

6. Batch Normalization is done in order to bring all the activation layer values to the same scale, we normalize them such that the hidden representation doesn't vary drastically and also helps us to get improvement in the training speed, also preventing from overfitting.