Machine Learning Engineer Nanodegree

Capstone Project

Sarah Hossam Elmowafy
November 25, 2018

I. Definition

Project Overview

In this project i will be solving an image classification problem by building a convolutional neural network that will be trained on few thousand images of cats and dogs, where our goal will be to tell which class the input image belongs to.
The way i am going to achieve it is by training an artificial neural network on few thousand images of cats and dogs and make the NN(Neural Network) learn to predict which class the image belongs to, next time it sees an image having a cat or dog in it.

Simply Dog and Cat Classification project takes an image of a cat or a dog and Predict what is in the image, is it a cat or a dog.

Problem Statement

For this project, Our goal is to create a classifier that trains the computer to extract these distinctive features and correctly classify animals, namely cats and dogs. There are various features to create our classifier around, such as shape of the faces, fur details, the angles created from the faces, etc. After serious consideration, The model we will choose then will be able to correctly classify the image if it's cat or dog, then we could use it in any researches.

For this project I'm proposing a binary classification problem, and I am going to use the Kaggle dataset, which is related to the problem of classification of Dogs vs Cats

I will be using a Convolution Neural Network (CNN), which is the best at capturing the features from images .

The problem is to predict whether an image contains a dog or a cat. This is a binary classification challenge. The goal is to predict is_cat is ‘0’ or ‘1’.

If ‘0’ it is a cat else ‘1’, which is a dog.

The strategy to solve this problem is as follows.

1. Download the train and test data from Kaggle.
2. Data loading and Preprocessing.
3. Plotting one batch of the training data to see how does it look like and giving them labels of [0,1] for cats and [1,0] for dogs.
4. Build a Classical Convolutional Neural Network with three Blocks of conv layers followed by fully connecte layer.
5. Compile the model with ‘binary_crossentropy’ loss, ‘rmsprop’ optimizer and ‘accuracy’ metric.
6. Train the model choosing the best number of epochs then.
7. Predict the 0/1 ‘as we convertes [0,1] for a cat to 0 and [1,0] for a dog to 1’ on the validation data then evaluate the accuracy.
8. Data Augmentation to to reduce overfitting and better generalization for the network, also this is increase number of data, this will include things like Flipping the images either vertically or horizontaly, Rotating the image, Zooming in or out on the image, cropping the image.
9. Fitting the model on the new augmented data and evaluate the accuracy.
10. Showing a sample for the augmentation to an image from the taining images to see how does it look like.
11. Fine-tuning using ‘VGG16’ to reach to the best accuracy to the performance of the classifier by using the same archticture and just modify the last layer in it wich is Dense layer with 1000 node to 2 nodes which are our categories Cat and Dog.
12. Compile the model with learning rate of .0001 and categorical crossentropy and accuracy metric.
13. Fit this model to the training data and evaluate on the validation set.
14. Prediction for the final model and this what will be used for classification of images of Dogs and Cats

Metrics

The evaluation metric used in accuracy.

$$Accuracy = \frac{TP + TN}{TN + FN + TP + FP}$$

The simple accuracy metrics used as the training dataset has more than 50.0% "Cat" and less than 50% "Dog". The classes are close to balanced.

If the training data classes are imbalanced, then f1-score metric is better but, this dataset is almost balanced, so using Accuracy as the metrics.

Note: As for Kaggle submissions probabilities of iceberg required, so, for Kaggle submissions log loss is the correct metrics.

II. Analysis

Data Exploration

Kaggle dataset link: https://www.kaggle.com/c/dogs-vs-cats/data these datasets are labeled, here's example:

We are going to save the directories relative paths of training ,validation and test data in these variables

train_data_dir = 'data/train'

validation_data_dir = 'data/validation'

test_data_dir = 'data/test'

after that we will

after that we will generate them as batches from corresponding directories using 'ImageDataGenerator()' Keras object

this will generate data as tensor image data and this is the format that the actual images need to be in to be read by the Keras model.

This is a batch from the training set.

Data directories' paths includes :

1. Training data :
    - Cats = 2000
    - Dogs = 2000

2. Validation data :
    - Cats = 500
    - Dogs = 500

3. test data :
    - Cat = 100
    - Dog = 100

We will use 150x 150 images, with three layers RGB.

used to rescale the pixel values from [0, 255] to [0, 1] interval

Exploratory Visualization

CNN model chooses the features explicitly from the data by passing filters through the images and extract these features in every convolutional layer.

We have 4000 images of cats and dogs 50% cats, 50% dogs for training

And 1000 images for validation, 50 % cats and 50% dogs.

Also 200 images for testing 50% cats , 50% dogs

Algorithms and Techniques

The classifier here is Convolutional Neural Network , which is the state-of-the-art algorithm for most image processing tasks, including classification. It needs a large amount of training data compared to other approaches; fortunately, Kaggle datasets are big enough. The algorithm outputs 0 for cat and 1 for dog.

Basic Architecture of CNN

The basics of a CNN architecture consist of 3 components. A convolution, pooling, and fully connected layer. These components work together to learn a dense feature representation of an input.

1. Convolution: -

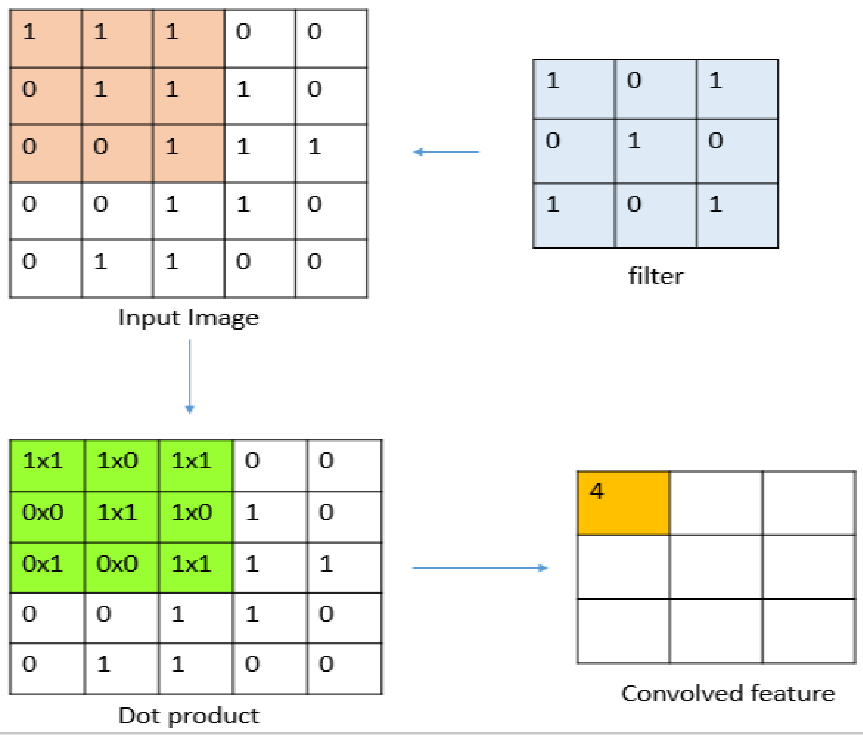
A convolution consists of a kernel, also called filter, that is applied in a sliding window fashion to extract features from the input. This filter is shifted after each operation across the input by an amount called strides. At each operation, a matrix multiply (dot product) of the kernel and current region of input is calculated. Filters can be stacked to create high-dimensional representations of the input.

2. Strides:

After we choose the filter size, we also choose the stride.

Stride controls how the filter convolves around the input volume. The filter convolves around the input volume by shifting one unit at a time. The amount by which the filter shifts is the stride.

Convolution example: The below example is a convolution layer, where a 3x3 filter is applied to a 5x5 image and stride 1.



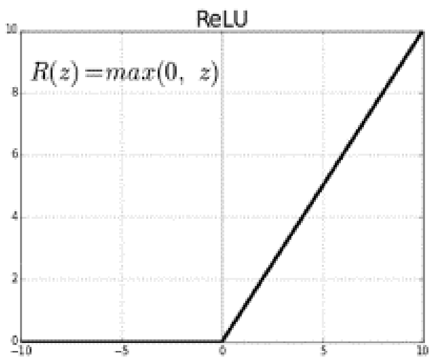
3. ReLU layer:

The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers.

In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that ReLU layers work far better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference to the accuracy.

It also helps to remove the vanishing gradient problem, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers.

The ReLU layer applies the function R(z) = max(0, z) to all of the values in the input volume. In basic terms, this layer just changes all the negative activations to 0.



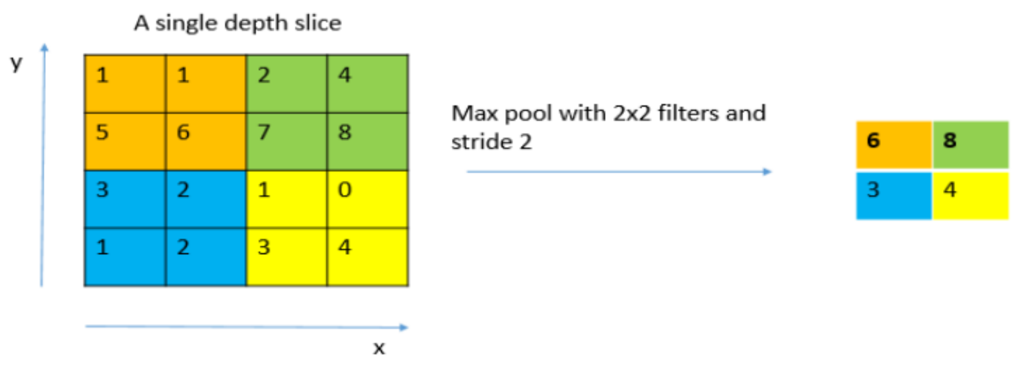
4. Pooling:

Pooling is an operation to reduce dimensionality. It applies a function summarizing neighboring information. Two common functions are max pooling and average pooling. By calculating the max of an input region, the output summarizes intensity of surrounding values.

Pooling layers also have a kernel, padding and are moved in strides. To calculate the output size of a pooling operation, you can use the formula,

(Input Width - kernel width + 2 * padding) / strides + 1.

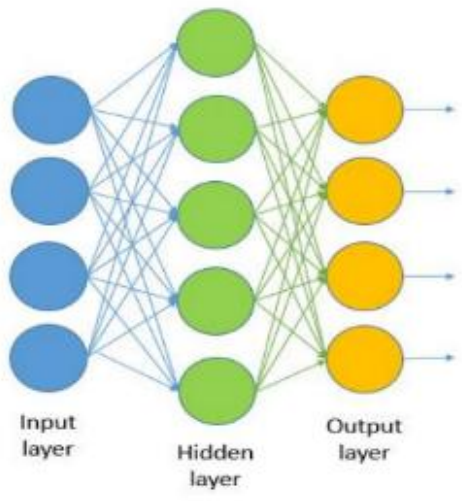
Example: Max pooling applied to below 4x4 image with stride 2 and 2x2 filters. So,(4 – 2 + 2 * 0)/2+1 = 2



5. Fully Connected layer: Fully connected layers are neural networks, where Each neuron in the input is connected to each neuron in the output; fully-connected. Due to this connectivity, each neuron in the output will be used at most one time.

In a CNN, the input is fed from the pooling layer into the fully connected layer.

In the below figure each neuron is fully connected the neurons in the next layers, except the output layer.

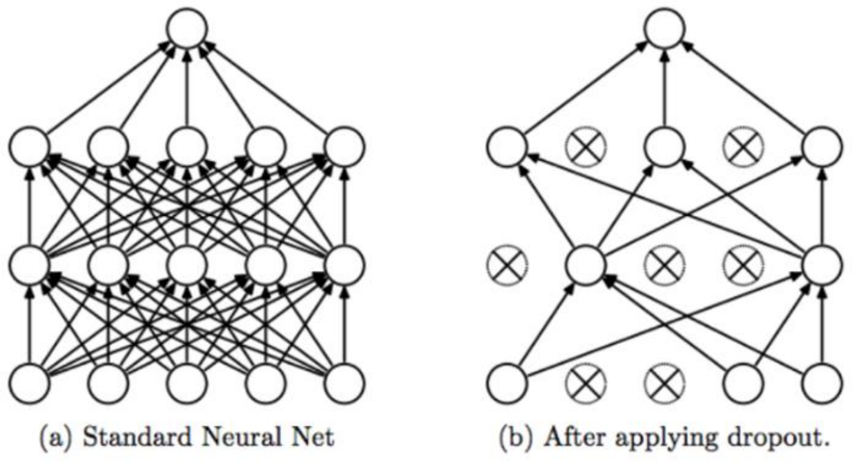


6. Dropout layer:

Dropout layer used to reduce overfitting.

Dropout prevents complex co-adaptation on the training data.

Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.



Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

Transfer Learning:

Transfer learning is the process of taking a pre-trained model (the weights and parameters of a network that has been trained on a large dataset by somebody else) and "fine-tuning" the model with your own dataset.

The idea is that this pre-trained model will act as a feature extractor. Remove the last layer of the network and replace it with your own classifier (depending on what your problem space is). You then freeze the weights of all the other layers and train the network normally (Freezing the layers means not changing the weights during gradient descent/optimization).

The VGG16 pre-trained model used for transfer learning.

Data Augmentation Techniques:

when a computer takes an image as an input, it will take in an array of pixel values.

Let's say that the whole image is shifted left by 1 pixel. To you and me, this change is imperceptible. However, to a computer, this shift can be significant as the classification or label of the image doesn't change, while the array does.

Approaches that alter the training data in ways that change the array representation while keeping the label the same are known as data augmentation techniques.

The model will be trained using transfer learning, which uses the popular pretrained model vgg-16 architecture. The model validation done using k-fold cross validation.

The following parameters can be tuned to optimize the classifier:

1. Training parameters:

• Stochastic gradient descent parameters:

- lr: float >= 0. Learning rate.
- momentum: float >= 0. Parameter updates momentum.
- decay: float >= 0. Learning rate decay over each update.
- nesterov: boolean. Whether to apply Nesterov momentum.

• Keras model compile parameters:

- optimizer: String (name of optimizer) or optimizer object.
- loss: String (name of objective function) or objective function.
- metrics: List of metrics to be evaluated by the model during training and testing.

• Keras fit_generator parameters:

- steps_per_epoch: Total number of steps (batches of samples) to yield from generator before declaring one epoch finished and starting the next epoch.
- epochs: Integer, total number of iterations on the data.
- shuffle: Whether to shuffle the order of the batches at the beginning of each epoch.
- verbose: Verbosity mode, 0, 1, or 2.
- validation_data: A generator for the validation data -> A tuple (inputs, targets)
- callbacks: List of callbacks to be called during training.

2. Neural network architecture

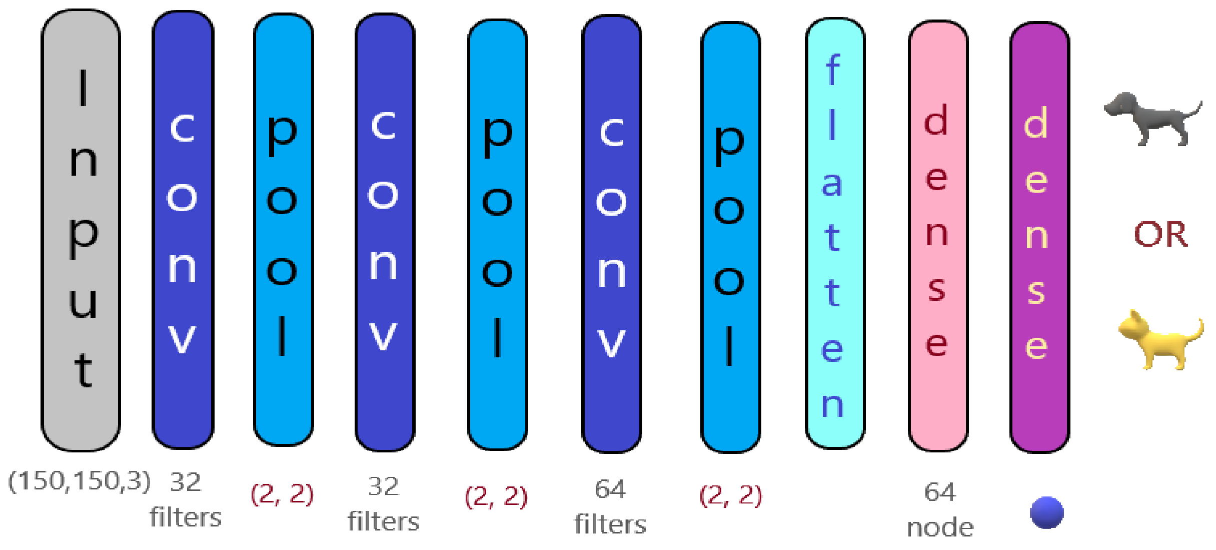
- Number of layers
- Layer types (convolutional, fully-connected or pooling, dense, dropout)
- Layer parameters

During training, both the training and the validation sets are loaded into the RAM. After that random batches are selected to be loaded into the GPU memory for processing.

The following parameters can be tuned to optimize the classifier:

- Training parameters
  - Training length (number of epochs)
  - Batch size (how many images to look at once during a single training step)
  - Solver type (what algorithm to use for learning)
  - Learning rate (how fast to learn; this can be dynamic)
- Neural network architecture
  - Number of layers
  - Layer types (convolutional , fully-connected , or pooling)
  - Layer parameters (see links above)
- Preprocessing parameters (see the Data Preprocessing section)

This is the architecture of the model i have created from scratch



Benchmark

The Benchmark model got an accuracy of 75% for classification

These are the predictions

```
array([[1.],[1.],[0.],[1.],[1.],[1.], [1.], [1.], [1.], [1.]],dtype=float32)
```

-the results are unsatisfying

III. Methodology

Data Preprocessing

We are going to save the directories relative paths of training ,validation and test data in these variables

- train_data_dir = 'data/train'
- validation_data_dir = 'data/validation'
- test_data_dir = 'data/test'

after that we will

after that we will generate them as batches from corresponding directories using 'ImageDataGenerator()' Keras object

this will generate data as tensor image data and this is the format that the actual images need to be in to be read by the Keras model

Data directories' paths includes :

1. Training data :

- Cats = 2000
- Dogs = 2000

2. Validation data :

- Cats = 500
- Dogs = 500

3. test data :

- Cat = 100
- Dog = 100

We will use 150x 150 images, with three layers RGB.

used to rescale the pixel values from [0, 255] to [0, 1] interval

Implementation

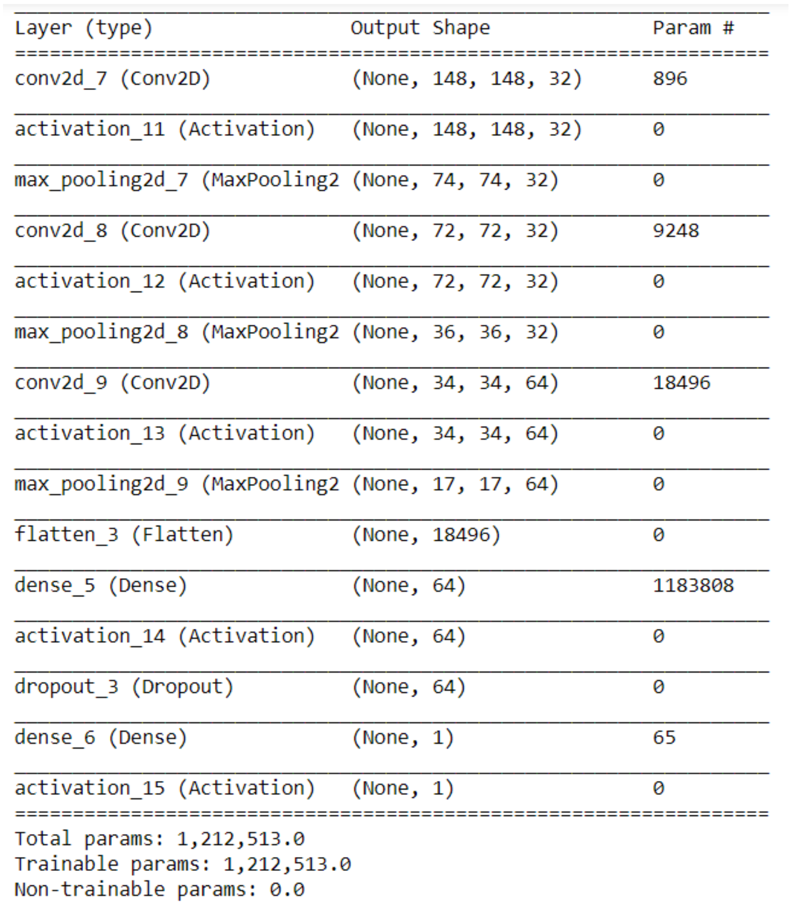
after loading data and preprocessing images.

we show a batch from the training data labeled with [0,1] for cats, [1,0] for dogs

we build a classical Convolution network which has three Blocks followed by a fully connected layer, so the three blocks layers' architecture looks like:

- Convolution2D: first layer is always convolutional layer,
- Activation 'relu' activation function to replace any negative values with zero.
- Maxpooling to rescale the data in smaller dimension.

with total number of params = 1,212,513.0



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
activation_11 (Activation)   (None, 148, 148, 32)      0
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 72, 72, 32)        9248
_________________________________________________________________
activation_12 (Activation)   (None, 72, 72, 32)        0
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 34, 34, 64)        18496
_________________________________________________________________
activation_13 (Activation)   (None, 34, 34, 64)        0
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 17, 17, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 18496)             0
_________________________________________________________________
dense_5 (Dense)              (None, 64)                1183808
_________________________________________________________________
activation_14 (Activation)   (None, 64)                0
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 65
_________________________________________________________________
activation_15 (Activation)   (None, 1)                 0
=================================================================
Total params: 1,212,513.0
Trainable params: 1,212,513.0
Non-trainable params: 0.0
```

After that we compile the model
We can figure the learning process by using the compile method where we'll define

- **rmsprop** as optimization function for gradient descent
- **loss** as 'binary_crossentropy' which is the prefered loss for binary classification problems.
- **metrics** 'accuracy' since this is a classification problem.

ModelTraining

first we define the number of training and validation data, also the number of epochs to run for each.
then write out our fit function to train the model giving it those parameters for training and validation data as well as the number of epochs.

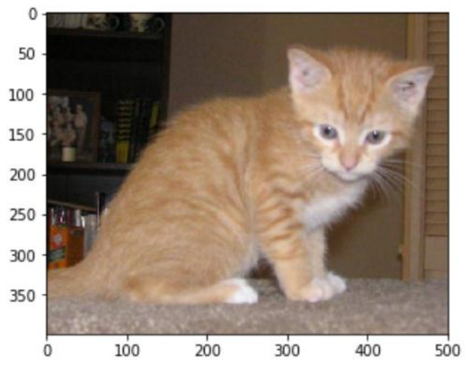
- nb_epoch = 18
- nb_train_samples = 4000
- nb_validation_samples = 1000

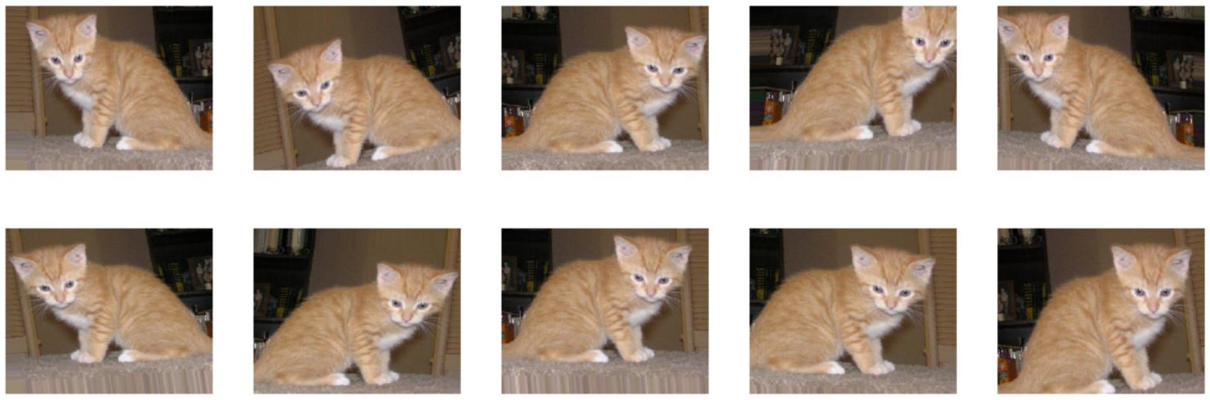
Data augmentation for improving the model

By applying random transformation to our train set, we artificially enhance our dataset with new unseen images.
This will hopefully reduce overfitting and allows better generalization capability for our network.

then we will fit the model again to the augmented data, but the accuracy still not good enough

this is a sample



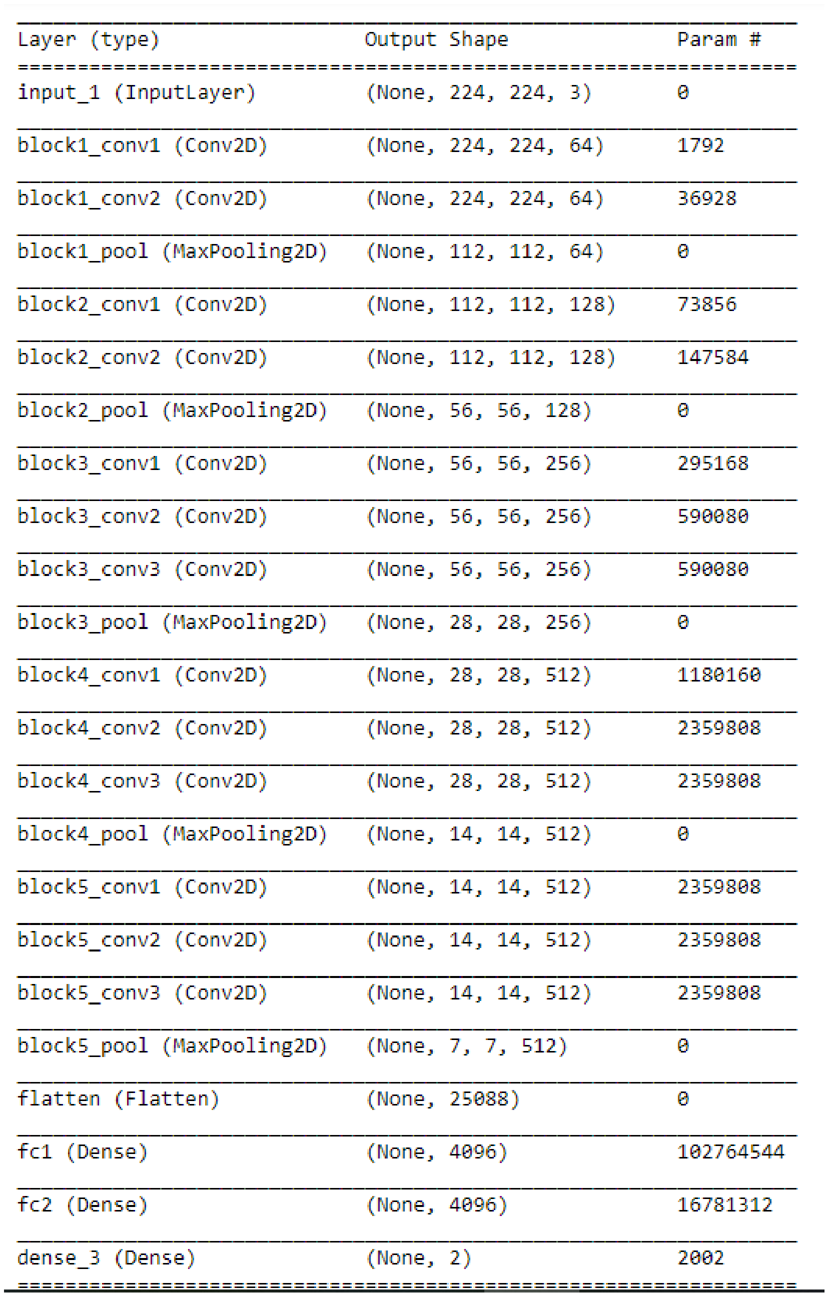


Fine Tuning

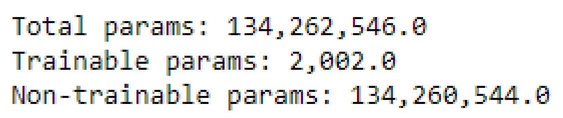
we use VGG16 modeling to improve the performance of our model

first : Build Fine-tuned VGG16 model

we will use the same layers of VGG16 in a sequential model but we replace the last dense to fit our model



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 2002
=================================================================
```



```
Total params: 134,262,546.0
Trainable params: 2,002.0
Non-trainable params: 134,260,544.0
```

After compiling and fitting the tuned model, we will make predictions again to find that it has higher value for accuracy and smaller for loss.

Refinement

at first I intended to use large amount of data, about 10000 for training and validation, after that I decided to use 4000 for training at 1000 for validation.

For the convolutional network I built I tried to change the number of epochs more than one time, but at the end found number of 15 epochs has the best accuracy evaluation.

Also I tried just one block of the conv layer with activation function and pooling and tried two but three gave me the best results.

Also for the same model on augmented data, I change the number of epochs till I reached to 20 epochs as the best accuracy of 83%

For the tuned model, it takes too long to run one epoch, so I chose just 5 epochs and it didn't achieve the best accuracy, but if we increased the number of epochs it may reach better accuracy.

For the number of batches, I tried to set the number of test batches to 5, but then change it to ten, to make the results clearer.

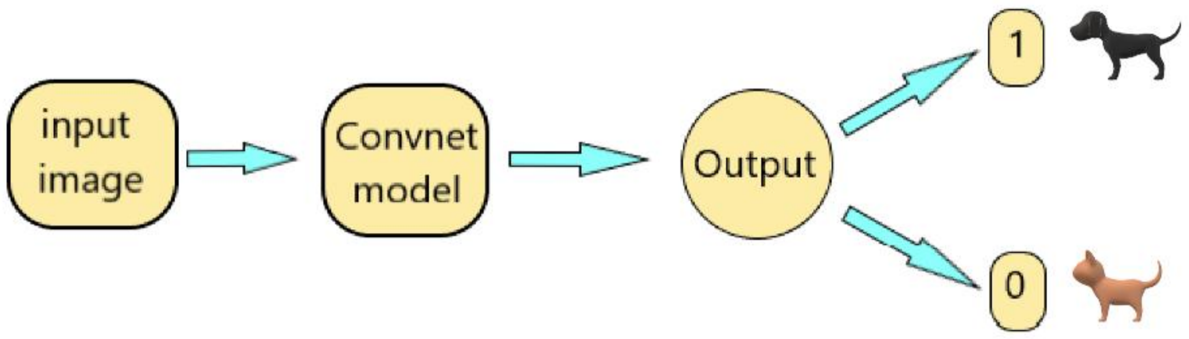
IV. Results

Model Evaluation and Validation

As the accuracy reached 97% after 5 epochs, if we increased number of epochs it may reach 100%.

V. Conclusion

Free-Form Visualization



Reflection

- Collecting data
- Building model
- Train the model
- Augmenting the   data
- Compile and fit the model
- Fine tuning to the model
- Prediction

The difficulties is that I have no GPU support and this takes more than 8 hours for one epoch in the tuned model, so it was really hard for me to try in this important part.

Improvement

gather more image dataset for better accuracy result.

Classification of all animals, not just cats and dogs.