MACHINE LEARNING ENGINEER NANODEGREE – CAPSTONE PROJECT

SANGAME KRISHNAMANI
MAR 29, 2018

DEFINITION

PROJECT OVERVIEW

The problem domain for this exercise is computer vision and deep learning. Image classification is one of the reasons why I got interested in machine learning course. Deep learning Dog Breed recognition project made me interested in pursuing further in improving the algorithm. But instead of staying in the realms of the same idea and improving on top of it, I have made it more challenging. The idea is to recognize missing pets and by security camera footage/images posted by people online of homeless pets.
The American Humane Association estimates

- over 10 million dogs and cats are lost or stolen in the U.S. every year. One in three pets will become lost at some point during their life.
- One in three pets will become lost at some point during their life
- Only 22 percent of lost dogs and 2 percent of lost cats that entered the animal shelters were reunited with their families

To make this more focused and achievable the aim of this project is to build a convolutional neural network that classifies whether image contains dogs vs cats.
It is easy for humans to recognize a dog vs cat, but is a challenge for computers because of the variety of breeds, large diversity of photo noise (wide variety of backgrounds, angles, poses, lighting, etc). This makes accurate classification quite difficult to achieve.
In an informal poll conducted many years ago, computer vision experts posited that a classifier with better than 60% accuracy would be difficult without a major advance in the state of the art. For reference, a 60% classifier improves the guessing probability of a 12-image HIP from 1/4096 to 1/459.
Advent in Machine learning has made this problem solvable with decent accuracy and what excites me the most is the fact that ML has opened the door to unlimited possibilities.
This base idea can be expanded to other species and even humans for face recognition, very similar to how Facebook provides suggestions for tagging a face.

Data Source: https://www.kaggle.com/c/dogs-vs-cats/data
Academic Paper:
https://www.robots.ox.ac.uk/~vgg/publications/2012/parkhi12a/parkhi12a.pdf

https://www.mathworks.com/company/newsletters/articles/deep-learning-for-computer-vision-with-matlab.html

PROBLEM STATEMENT

The goal is to build a face recognition system for Dogs and Cats and classify them appropriately.

The dataset images were manually classified and labeled by Asirra. We have a dataset of 25000 images for cats and dogs with labels to accomplish this. Once the computer is trained, we have a test dataset of 12,500 with no labels (cat or dog). Assumption would be that image would contain either a dog or cat but not both. The images might also contain other objects.

The images are not all of the same size and there are other objects that act as noise on the image. This is a supervised learning problem because each image acts as an input and a desired output value (dog or cat) is expected, i.e. a binary classification. My approach here is to use a convolutional neural network (CNN).

The steps used in the project:

1. Get data from Kaggle
2. Preprocess the data
   a) Divide images to training and validation set
   b) Resize images in training folder
3. Define the architecture of the CNN from scratch
4. Train and Optimize for accuracy.
5. Use the classifier to recognize dogs vs cats.

METRICS

For this project we have two datasets:

- Train dataset of 25,000 images with labels
- Test dataset of 12,500 images. No labels provided.

I am dividing the train dataset into 3 different sets –

1. Training dataset of 10400 images that will be used to train the model,
2. Validation dataset of 2600 images for testing the performance of the CNN
3. A test set of 5000 images that will be used to do a final evaluation before submission.

The goal of creating this test set is that as the CNN is repeatedly trained and tested on validation set, it will eventually "see" the validation set. This test set will be a more objective evaluation of the accuracy of CNN on an out of sample. I will be using two metrics for measuring the success of the project:

Either Log Loss or Accuracy Metrics:

Log Loss (categorical cross entropy) quantifies the accuracy of a classifier by penalizing false classifications. Minimizing the Log Loss is basically equivalent to maximizing the accuracy of the classifier, but there is a subtle twist which we'll get to in a moment.
In order to calculate Log Loss the classifier must assign a probability to each class rather than simply yielding the most likely class. Mathematically Log Loss is defined as

$$-\frac{1}{N}\sum_{i=1}^{N}\ \sum_{j=1}^{M} y_{ij} p_{ij}$$

where N is the number of samples or instances, M is the number of possible labels, y_{ij} is a binary indicator of whether or not label j is the correct classification for instance i, and p_{ij} is the model probability of assigning label j to instance i. A perfect classifier would have a Log Loss of precisely zero.

Accuracy is the ratio of correct predictions to total number of events. In this case it would be the percentage labels correctly classified.

$$\frac{\mathrm{TN}+\mathrm{TP}}{\mathrm{TN}+\mathrm{FP}+\mathrm{FN}+\mathrm{TP}}$$

Where, TN – number of True Negative cases, FP – False Positives, FN – False Negatives, TP – True Positives.

The reason I am considering Accuracy Metrics and log loss is because of the simplicity and effectiveness of these metrics when validating for an image classification problem. The accuracy can be easily validated on the test data based on quick manual validation of the images and using the above formula to come up with the accuracy percentage. And when we look at it, log loss and accuracy is inversely proportional and goes hand in hand. So I mainly went with using accuracy throughout the project with few instances where I used log loss.

ANALYSIS

DATASETS EXPLORATION

A subset of **Asirra dataset** will be used for this project. CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) or HIP (Human Interactive Proof) are challenges that are easy for people to solve but not for computers. HIPs are used for many purposes, such as to reduce email and blog spam and prevent brute-force attacks on web site passwords.

Asirra (Animal Species Image Recognition for Restricting Access) is a HIP that works by asking users to identify photographs of cats and dogs. This task is difficult for computers, but studies have shown that people can accomplish it quickly and accurately. Many even think it's fun! Here is an example of the Asirra interface:

Asirra is unique because of its partnership with Petfinder.com, the world's largest site devoted to finding homes for homeless pets. They've provided Microsoft Research with over three million images of cats and dogs, manually classified by people at thousands of animal shelters across the United States.

Each image will have either a dog or cat in the image and can contain other objects and human. The train folder contains 25,000 images of dogs and cats. Each image in this folder has the label as part of the filename. The test folder contains 12,500 images, named according to a numeric id.
Few challenging images are shown here:
Some are blurry, other have more than one cat/dog, some have humans and other objects, the image sizes are different and the animals are facing in different angles.
Image format: .jpg file
Number of Images:

Training
Cat images: 12500
Dog images: 12500
Each image in this folder has the label as part of the filename

Testing
Cat/Dog images: 12500

EXPLORATORY VISUALIZATION

Size and description of images

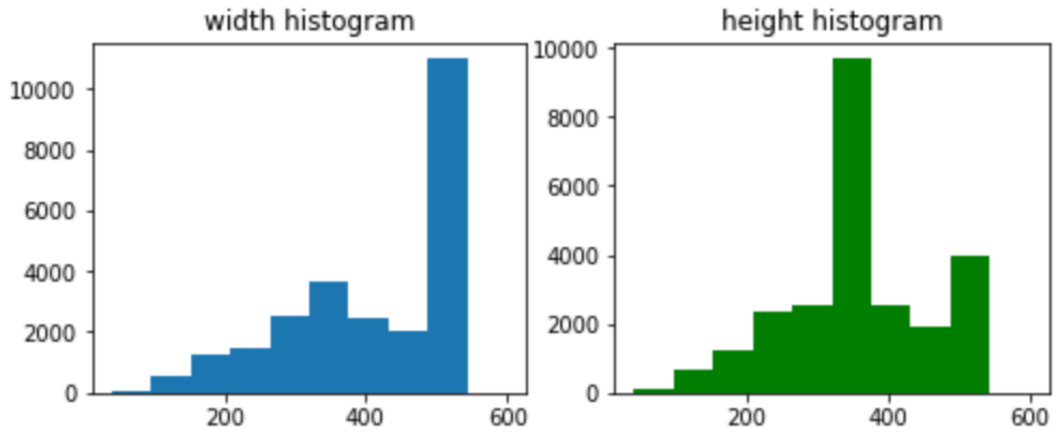
All images are different in sizes.

- The images have varying levels of lightening
- there are other objects and humans in the dataset along with cats or dogs
- The face of the animal is angled at different positions and is not always front facing
- The animal is not always present in the center of the frame.

Data Source: https://www.kaggle.com/c/dogs-vs-cats/data

Random Training sample examples:

Width and Height histogram to represent the variation of the width and height of images:



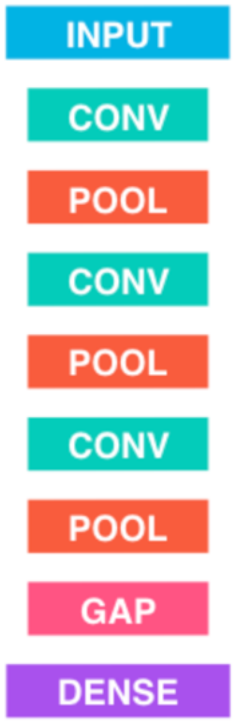
```
Mean width : 404.09904
Mean height : 360.47808
```

ALGORITHMS AND TECHNIQUES

Deep learning techniques are most effective on image classification and the same approach would be used to solve this problem. Data augmentation and transfer learning will be used to train a Convolutional Neural Network (CNN) to classify the images as dogs vs cats.

Option 1:

The CNN architecture is known for best image classification. As mentioned in the workflow section below, a CNN will be trained from scratch. Using an architecture similar to the one shown below, I would attempt to train a CNN from scratch. The architecture has 3 convolution layers with a ReLu activation function, followed by max-pooling layers. Before the fully-connected layer, because of the large amount of data a dropout layer can considered to reduce overfitting.



OR/AND

Option 2:

Transfer learning or inductive transfer is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks. Many such networks pre-trained on imagenet challenge are available today – RESNET, Inception-V3, VGG-16, VGG-19, etc.

The CNN will be trained using transfer layer, one of the pre-trained network will be used for the same.

Comparison (optional)

If the previous option of training the CNN from scratch is attempted Option 1 can be compared with 2.

Model architecture techniques and terms explained

“In machine learning, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery.

Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli

Pooling - Convolutional networks may include local or global pooling layers, which combine the outputs of neuron clusters at one layer into a single neuron in the next layer. For example, max pooling uses the maximum value from each of a cluster of neurons at the prior layer.

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network

Global average pooling (GAP) layers are used to minimize overfitting by reducing the total number of parameters in the model. Similar to max pooling layers, GAP layers are used to reduce the spatial dimensions of a three-dimensional tensor. However, GAP layers perform a more extreme type of dimensionality reduction, where a tensor with dimensions $h \times w \times d$ is reduced in size to have dimensions $1 \times 1 \times d$. GAP layers reduce each $h \times w$ feature map to a single number by simply taking the average of all hw values.”

BENCHMARK

“The security of ASIRRA is based on the presumed difficulty of classifying images of cats and dogs automatically. As reported in [6], evidence from the 2006 PASCAL Visual Object Classes Challenge suggests that cats and dogs are particularly difficult to tell apart algorithmically. A classifier based on color features, described in [6], is only 56.9% accurate.”
Ref: https://eprint.iacr.org/2008/126.pdf

The benchmark implementation with no optimization yielded an accuracy around 50% and validation loss around 81%.

```
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_8 (Conv2D)            (None, 223, 223, 16)      208       
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 111, 111, 16)      0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 197136)            0         
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 394274    
=================================================================
Total params: 394,482
Trainable params: 394,482
Non-trainable params: 0
_________________________________________________________________
```

```
Train on 10000 samples, validate on 2500 samples
Epoch 1/4
 9980/10000 [============================>.] - ETA: 0s - loss: 8.0285 - acc: 0.5002Epoch 00001: val_loss improved from inf to 7.99458, saving model to saved_models/weights.be
st.benchmark_from_scratch.hdf5
10000/10000 [==============================] - 108s 11ms/step - loss: 8.0318 - acc: 0.5000 - val_loss: 7.9946 - val_acc: 0.5040
Epoch 2/4
 9980/10000 [============================>.] - ETA: 0s - loss: 8.0639 - acc: 0.4997Epoch 00002: val_loss did not improve
10000/10000 [==============================] - 105s 10ms/step - loss: 8.0639 - acc: 0.4997 - val_loss: 7.9946 - val_acc: 0.5040
Epoch 3/4
 9980/10000 [============================>.] - ETA: 0s - loss: 8.0704 - acc: 0.4993Epoch 00003: val_loss did not improve
10000/10000 [==============================] - 104s 10ms/step - loss: 8.0639 - acc: 0.4997 - val_loss: 7.9946 - val_acc: 0.5040
Epoch 4/4
 9980/10000 [============================>.] - ETA: 0s - loss: 8.0671 - acc: 0.4995Epoch 00004: val_loss did not improve
10000/10000 [==============================] - 103s 10ms/step - loss: 8.0639 - acc: 0.4997 - val_loss: 7.9946 - val_acc: 0.5040
```

METHODOLOGY

DATA PREPROCESSING

“These are few of the data preprocessing steps that were taken after analyzing the data manually and performing exploratory visualization of image properties:

When using TensorFlow as backend, Keras CNNs require a 4D array (which we'll also refer to as a 4D tensor) as input, with shape

(samples, rows, columns, channels)

where samples corresponds to number of images and number of rows, columns and channels follow.

The path_to_tensor function takes a string-valued file path to a color image as input and returns a 4D tensor suitable for supplying to a Keras CNN. The function first loads the image and resizes it to a square image that is pixels. Next, the image is converted to an array, which is then resized to a 4D tensor. In this case, since we are working with color images, each image has three channels.

The paths_to_tensor function takes a numpy array of string-valued image paths as input and returns a 4D tensor with shape

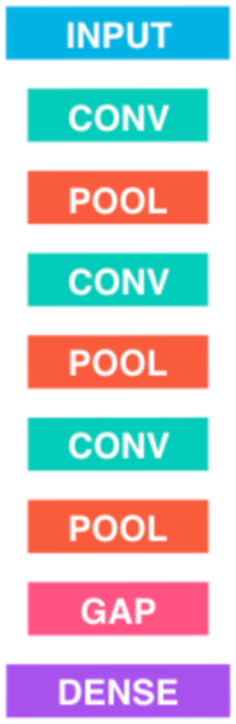
(samples, 224, 224, 3)

Here, samples is number of images, in the supplied array of image paths. It is best to think of samples as the number of 3D tensors (where each 3D tensor corresponds to a different image) in your dataset.

IMPLEMENTATION

Libraries: Scikit-learn, Keras, Tensorflow, NumPy, Pandas, matplotlib, OpenCv
Workflow:
Potentially these would be the steps I would follow –

- Resize the images, so they are of the same size.
- Train a CNN from scratch for comparison with transfer learning models (optional)



- Fine tuning the pre-trained network by choosing different optimizers.

Results for CNN built from scratch, the algorithm used is shown below:

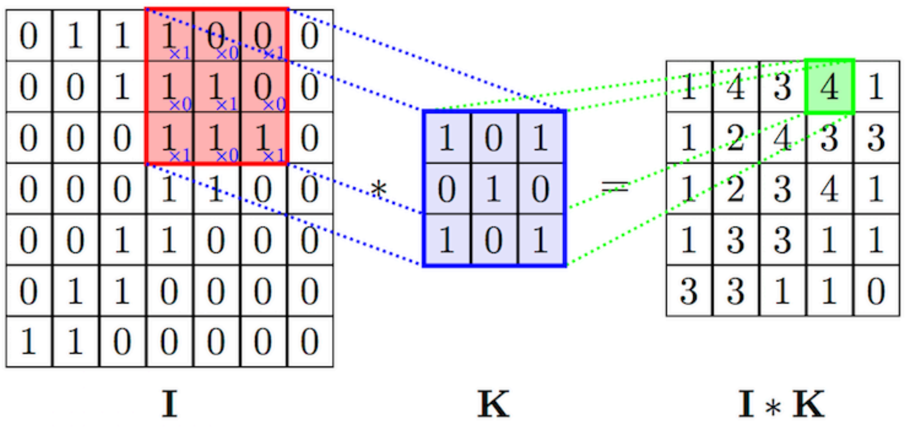
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 223, 223, 16)      208
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 111, 111, 16)      0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 110, 110, 32)      2080
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 55, 55, 32)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 54, 54, 64)        8256
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 27, 27, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 26, 26, 64)        16448
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 13, 13, 4)         2308
_________________________________________________________________
flatten_4 (Flatten)          (None, 676)               0
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 1354
=================================================================
Total params: 30,654
Trainable params: 30,654
Non-trainable params: 0
_________________________________________________________________
```

Step 1: Initializing the neural network as a sequential network from keras.models
Step 2: Import Conv2D from keras.layers to perform convolution operation on 2D images

"Enter the convolution operator. Given a two-dimensional image, I, and a small matrix, KK of size h×wh×w, (known as a convolution kernel), which we assume encodes a way of extracting an interesting image feature, we compute the convolved image, I∗KI∗K, by overlaying the kernel on top of the image in all possible ways, and recording the sum of elementwise products between the image and the kernel:

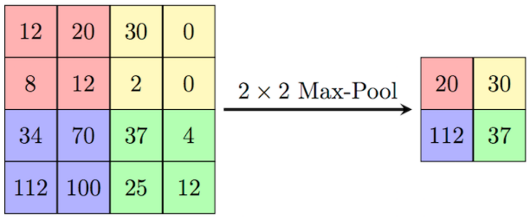
$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \cdot I_{x+i-1,y+j-1}$$

The images below show a diagrammatical overview of the above formula and the result of applying convolution (with two separate kernels) over an image, to act as an edge detector:"



Step 3: Import MaxPooling2D from keras.layers to get the maximum value pixel from region of interest.

"A very popular approach to downsampling is a pooling layer, which consumes small and (usually) disjoint chunks of the image (typically 2×22×2) and aggregates them into a single value. There are several possible schemes for the aggregation – the most popular being max-pooling, where the maximum pixel value within each chunk is taken. A diagrammatical illustration of 2×22×2 max-pooling is given below."



Step 4: Import Flatten from keras.layers to convert 2D arrays to liner vector.
Step 5: Import Dense from keras.layers to build the fully connected layer – Softmax gives the actual output class label probabilities.
Step 6: Compile the benchmark architecture with adam optimizer and cross entropy loss.
"Minimizing the cross-entropy loss has the effect of maximizing the model's confidence in the correct class, without being concerned for the probabilities for other classes – this makes it a more suitable choice for probabilistic tasks compared to, for example, the squared error loss."
Step 7: Train the classifier.

The outcome of the implementation are documented clearly in the below Results section.

REFINEMENT

Learnings from refinement:

- Image normalization and adding color channels did not have as much positive effect as I thought
- The depth of neural network affects accuracy. As the network gets deeper the ability to learn features improves
- The optimization function had the most impact on improving the accuracy

RESULTS

MODEL EVALUATION AND VALIDATION

I tried all the techniques I possibly could to refine the model. The final output seems reasonable in terms of parameters and layer structure. The incremental benefit of changes to the model was very small after a point. So theoretically refinement can continue but in my opinion the effect would be small and not worth the time.

Log loss is lower than the benchmark established in the beginning and the model accuracy score of 85.64% on validation set is reasonable.

The Log loss on the test dataset at Kaggle was 0.39. This is in the ballpark we had with the final model on our validation and test sample. As expected performance is slightly weaker on sample never seen before however the model seems robust.

```
Train on 10000 samples, validate on 2500 samples
Epoch 1/8
 9980/10000 [============================>.] - ETA: 0s - loss: 0.3219 - acc: 0.8600Epoch 00001: val_loss improved from inf to 0.35799, saving model to saved_mode
ls/weights.best.from_new_scratch.hdf5
10000/10000 [==============================] - 214s 21ms/step - loss: 0.3221 - acc: 0.8601 - val_loss: 0.3580 - val_acc: 0.8340
Epoch 2/8
 9980/10000 [============================>.] - ETA: 0s - loss: 0.2982 - acc: 0.8718Epoch 00002: val_loss improved from 0.35799 to 0.35754, saving model to saved_
models/weights.best.from_new_scratch.hdf5
10000/10000 [==============================] - 201s 20ms/step - loss: 0.2985 - acc: 0.8716 - val_loss: 0.3575 - val_acc: 0.8432
Epoch 3/8
 9980/10000 [============================>.] - ETA: 0s - loss: 0.2745 - acc: 0.8842Epoch 00003: val_loss improved from 0.35754 to 0.34683, saving model to saved_
models/weights.best.from_new_scratch.hdf5
10000/10000 [==============================] - 200s 20ms/step - loss: 0.2751 - acc: 0.8841 - val_loss: 0.3468 - val_acc: 0.8492
Epoch 4/8
 9980/10000 [============================>.] - ETA: 0s - loss: 0.2520 - acc: 0.8931Epoch 00004: val_loss did not improve
10000/10000 [==============================] - 204s 20ms/step - loss: 0.2520 - acc: 0.8929 - val_loss: 0.4002 - val_acc: 0.8292
Epoch 5/8
 9980/10000 [============================>.] - ETA: 0s - loss: 0.2341 - acc: 0.9064Epoch 00005: val_loss did not improve
10000/10000 [==============================] - 212s 21ms/step - loss: 0.2344 - acc: 0.9062 - val_loss: 0.4301 - val_acc: 0.8236
Epoch 6/8
 9980/10000 [============================>.] - ETA: 0s - loss: 0.2120 - acc: 0.9117Epoch 00006: val_loss did not improve
10000/10000 [==============================] - 212s 21ms/step - loss: 0.2118 - acc: 0.9117 - val_loss: 0.3675 - val_acc: 0.8492
Epoch 7/8
 9980/10000 [============================>.] - ETA: 0s - loss: 0.1962 - acc: 0.9161Epoch 00007: val_loss did not improve
10000/10000 [==============================] - 212s 21ms/step - loss: 0.1962 - acc: 0.9162 - val_loss: 0.3716 - val_acc: 0.8496
Epoch 8/8
 9980/10000 [============================>.] - ETA: 0s - loss: 0.1675 - acc: 0.9331Epoch 00008: val_loss did not improve
10000/10000 [==============================] - 250s 25ms/step - loss: 0.1673 - acc: 0.9331 - val_loss: 0.3983 - val_acc: 0.8564
```

The model has an 85.64% accuracy on the validation set and 93.31 % on the training set.

<u>Comparison with benchmark</u>:

When the benchmark model implementation was trained on 10,000 sample images and validated on 2500 samples. We got a validation set accuracy of 50% at best as opposed to 85.64% accuracy on validation set which is a significant improvement.

JUSTIFICATION

The final model for this problem is a deep convolution neural network with 5 convolution layers, 4 MaxPool layers, 1 dense layer and 1 flatten layers on colored images to predict probability of cat vs dog. After trying multiple combinations of parameters and optimization function to tune the model to its final setting there is minute incremental benefit of continuing this optimization after this point.

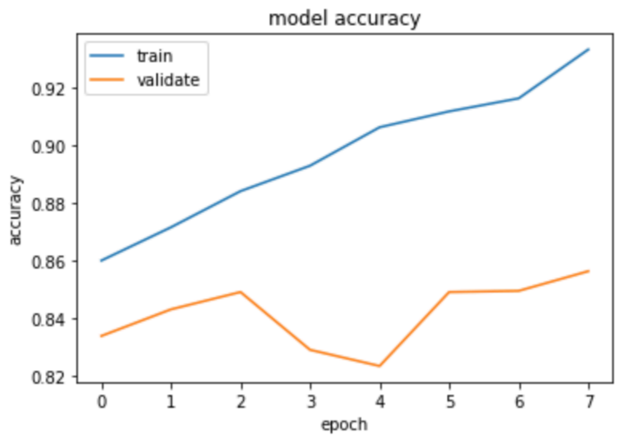
The model has an accuracy of 85.64% on validation set in its ability to correctly predict cat vs dog. This is still considerably low when compared with how well humans can categorize the images. I consider this as a representation of limitation on machine learning on images and also the fact that there is more factors to be considered for making this prediction even better.

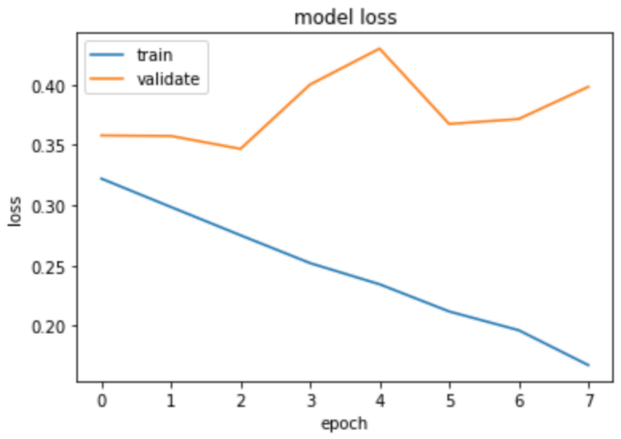
CONCLUSION

The problem I set out to solve was for classifying dogs and cats images. Convolution neural networks is known for analyzing visual imagery, I defined the metrics to measure, a benchmark model was established and the accuracy metrics was noted down. When implementing the actual model architecture, multiple layers were used and different optimization techniques were applied to get better and better results. When I hit a 85+% accuracy on the validation set when compared against the benchmark set at 50%, I had significantly improved the algorithm.

Another metric that can effectively help measure this problem is the model loss. Both these metrics are shown for the training and validation data set below on the final model.

FREE-FORM VISUALIZATION

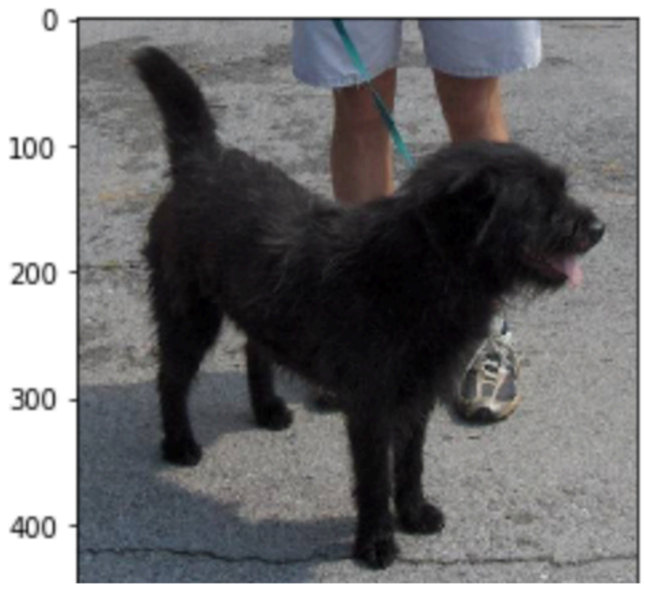




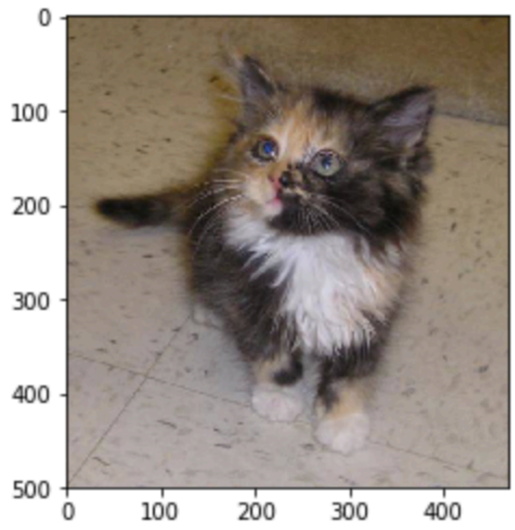
Finally a test data set which was never seen before by this algorithm is used to validate the robustness of the model. 50 images were tested in our case. The model is able to predict accurately on 43 images out of 50, which gives us an accuracy of 86%. Which is almost consistent with the predicted accuracy % of the validation set.

There were 5 images in the test set where predictions were incorrectly made. Looking at the images the ambiguity can be assessed, the dog/cat looks unique or the position in which the image was captured leads to this ambiguity. We still got an accuracy of 86% on test images.

```
Cat Image:
```



```
Dog Image:
```



REFLECTION

After working on these projects these are my learnings and thoughts:

1. I have a better understanding of deep learning and CNN. Along with increased confidence in applying these algorithms to use by using python. I could get to this point by learning these concepts and techniques one at a time and putting them all together.

2. Dataset and image augmentation – Understanding the problem started with looking into the dataset/images and analyzing what type of preprocessing and normalization has to be done on the images before feeding it to a CNN (resizing, flattening, color contrasts, etc were considered).

3. Build a base model for benchmark and Optimization and Refinement – This step helped with getting a better understanding of the architecture and what each step/process meant. It was challenging and at the same time exciting to perform optimization to achieve higher accuracy with results. I also explored using EC2 instance to perform multiple iterations, as the speed of processing was slow on my laptop.

4. Documentation – Step by step documentation and reflection on everything that went into putting this project together (in a structured way), helped with giving me confidence and gaining a deeper understanding.

IMPROVEMENT

On comparing my results with the leaderboard in Kaggle, where I see log loss rate is 0.047, which is way better than what I have achieved here.

- Overfitting – Use techniques to reduce overfitting and thus providing more accurate predictions.
- Preprocessing of image – I still think I could do better with preprocessing of images. Where I could clear out the noise (other objects in the image) and improve the accuracy.

REFERENCES

https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

https://www.kaggle.com/c/dogs-vs-cats
https://www.petfinder.com/dogs/lost-and-found-dogs/why-microchip/
https://en.wikipedia.org/wiki/Transfer_learning
https://eprint.iacr.org/2008/126.pdf
https://github.com/sangamek/dog-project

https://shaoanlu.wordpress.com/2017/05/29/sgd-all-which-one-is-the-best-optimizer-dogs-vs-cats-toy-experiment/
https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
https://keras.io/layers/convolutional/
https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html
https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/
https://en.wikipedia.org/wiki/Convolutional_neural_network