Machine Learning Engineer Nanodegree

Capstone Project

Kunal Gusain

8 April 2017

Cats vs Dogs Image Classification

1. Problem Overview

This problem belongs to the computer vision branch of the machine learning. This project or competition was first introduced on Kaggle in 2013 [1]. This is one of the most popular competitions ever organized on Kaggle. This is a type of binary image classification problem where we have to predict the class of an input image. The classes, in this case, are "cats" and "dogs", i.e the input image will be of either dog or cat. The dataset used in this project is taken from Kaggle. I have used CNN deep learning model to achieve the best classification result.

2. Problem Statement

The problem statement is to design an algorithm which is capable of finding out the correct animal (cat or dog) in new unseen images. For each image in the test set, we have to predict the class or label to which the new image belongs (cat or dog). I have used a scaled down version of VGG 16 (deep CNN model) to perform the classification.

3. Datasets

Asirra [3] (Animal Species Image Recognition for Restricting Access) is a Human Interactive Proof 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!

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. Kaggle is fortunate to offer a subset of this data for fun and research.

The dataset contains thousands of images of cats and dogs. There are about 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. I divided the 25,000 images into training and test set with a ratio of 75:25. All the models used in this project have used training set of 18750 images and final accuracy is tested on the testing set of 6250 images.

4. Metrics

The evaluation metrics which used is accuracy score and the loss function which I have used is log loss metric or binary cross entropy. The evaluation metric which was used in the Kaggle competition was log loss metric. Log-loss, or logarithmic loss, gets into the finer details of a classifier. In particular, if the raw output of the classifier is a numeric probability instead of a class label, then log-loss can be used. The probability essentially serves as a gauge of confidence. If the true label is "0" but the classifier thinks it belongs to class "1" with probability 0.51, then the classifier would be making a mistake. But it's a near miss because the probability is very close to the decision boundary of 0.5. Log-loss is a "soft" measurement of accuracy that incorporates this idea of probabilistic confidence.

Mathematically, log-loss or binary cross entropy for a binary classifier looks like this:

$$logloss = \sum_{i=1}^{N} y_i log(p_i) + (1 - y_i) log(1 - p_i)$$

Here, p_i is the probability that the i-th data point belongs to class "1", as judged by the classifier. y_i is the true label and is either "0" or "1". The beautiful thing about this definition is that it is intimately tied to information theory: Intuitively, log-loss measures the unpredictability of the "extra noise" that comes from using a predictor as opposed to the true labels. By minimizing the cross entropy, we maximize the accuracy of the classifier.

5. Dataset Exploration

The dataset is provided by Kaggle. This dataset was provided by Asirra [3] to Kaggle. This dataset contains 25,000 images of dogs and cats. I have used 18,750 images as training data and rest i.e 6250 as testing data. Almost all the images are colored images i.e we can read the image as RGB image. The dataset is provided in the form of images. One of the problem with this dataset is that all the images are not of same resolution, we have to bring all the images to the same resolution during preprocessing.

6. Exploratory Visualization

The number of dog images and cat images in the dataset are equal. Few of the random examples from dataset are:

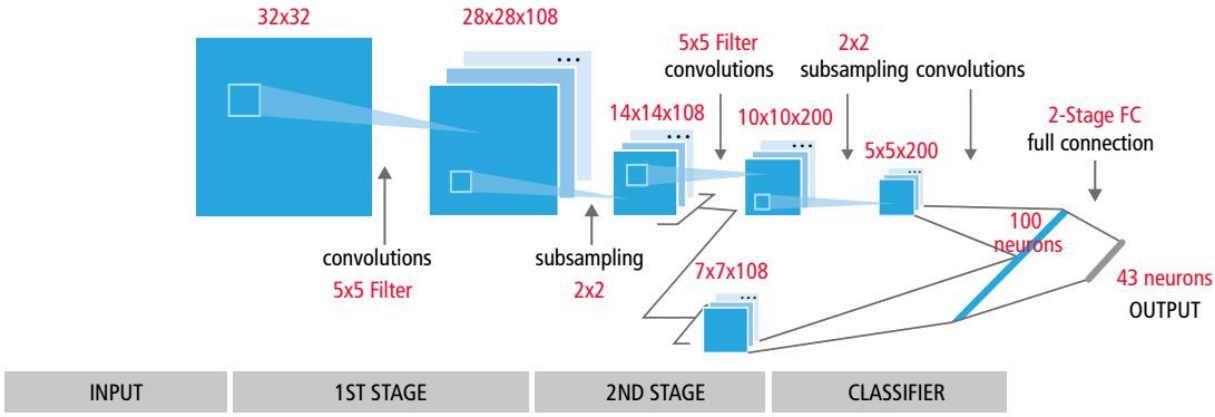
As we can see from the above images that it is easy to identify class for first two images, but in last two images it is slightly difficult to identify the class. The resolution of all the training images is varied, below is the scatterplot of the resolution of images.

The scatterplot clearly shows that most of the images do not have same resolution and we have to resize the images to bring them to same level.

7. Algorithms and Techniques

I have used Convolutional neural networks (CNNs) for this project. Convolutional neural networks (CNNs) are widely used in pattern- and image-recognition problems as they have a number of advantages compared to other techniques. A CNN is a special case of the neural network. A CNN consists of one or more convolutional layers, often with a subsampling layer, which are followed by one or more fully connected layers as in a standard neural network. The design of a CNN is motivated by the discovery of a visual mechanism, the visual cortex, in the brain. The visual cortex contains a lot of cells that are responsible for detecting light in small, overlapping sub-regions of the visual field, which are called receptive fields. These cells act as local filters over the input space, and the more complex cells have larger receptive fields. The convolution layer in a CNN performs the function that is performed by the cells in the visual cortex. In a CNN, convolution layers play the role of feature extractor. But they are not hand designed. Convolution filter kernel weights are decided on as part of the training process. Convolutional layers are able

to extract the local features because they restrict the receptive fields of the hidden layers to be local.



By stacking multiple and different layers in a CNN, complex architectures are built for classification problems. Four types of layers are most common: convolution layers, pooling/subsampling layers, non-linear layers, and fully connected layers.

I have intially trained a baseline model with just just four convolution layers to set a benhmark model. I have analysed the effect of different inputs (grayscale and rgb input) to the CNN. I have trained 10 layer and 13 layer CNN on color data (3 channel input) and 10 layer CNN model on grayscale images (1 channel input). I have also trained models using two different optimization techniques. I have used Adam optimizer and RMSProp optimizer.

8. Benchmark Model

The benchmark model which I have used has four convolution layers and 2 fully connected layers. The input to the benchmark model is a image input of 224*224*1 i.e the input has height and width of 224 pixel and has 1 channel. The number of kernels or filters used in each layer are 32,32,64,64. I used Adam optimizer in benchmark model. The accuracy achieved in the benchmark model is ~84.5%. Below is the image which gives the summary architecture of baseline model.

Layer (type)	Output Shape	Param #	Connected to
convolution2d_1 (Convolution2D)	(None, 224, 224, 32)	320	convolution2d_input_1[0][0]
convolution2d_2 (Convolution2D)	(None, 224, 224, 32)	9248	convolution2d_1[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 112, 112, 32)	0	convolution2d_2[0][0]
convolution2d_3 (Convolution2D)	(None, 112, 112, 64)	18496	maxpooling2d_1[0][0]
convolution2d_4 (Convolution2D)	(None, 112, 112, 64)	36928	convolution2d_3[0][0]
maxpooling2d_2 (MaxPooling2D)	(None, 56, 56, 64)	0	convolution2d_4[0][0]
flatten_1 (Flatten)	(None, 200704)	0	maxpooling2d_2[0][0]
dense_1 (Dense)	(None, 64)	12845120	flatten_1[0][0]
dropout_1 (Dropout)	(None, 64)	0	dense_1[0][0]
dense_2 (Dense)	(None, 64)	4160	dropout_1[0][0]
dropout_2 (Dropout)	(None, 64)	0	dense_2[0][0]
dense_3 (Dense)	(None, 1)	65	dropout_2[0][0]
activation_1 (Activation)	(None, 1)	0	dense_3[0][0]

Total params: 12,914,337
Trainable params: 12,914,337
Non-trainable params: 0

Fig. Benchmark Model Architecture

9. Data Preprocessing

Following steps were followed during data preprocessing:

1. All the images were first resized to 224*224 pixel.
2. Converted all the images to two different sets GrayScale format as well as RGB format for later analyses which of these performs better.
3. Normalized all the images in both the sets of GrayScale format as well as RGB format.
4. Next all the images were divided into two sets for each of GrayScale format as well as RGB format. One set of 18750 images called training set and rest set of 6250 images called testing set. Finally we have 4 sets, 2 training and 2 testing set, one for each Grayscale and RGB format respectively.

10. Implementation

I analyzed the effect on accuracy from different optimizer function, different inputs (Grayscale, RGB) as well as the effect on accuracy because of increasing numbers of layers of CNN. The learning rate for Adam optimizer was 1e-3 and learning rate for RMSprop optimizer was 1e-4.

Below are the list of models I implemented or trained:

1. CNN Model with Grayscale (1 channel) input, with 7 convolution layers with Adam optimizer. Below is the architecture:

Layer (type)	Output Shape	Param #	Connected to
convolution2d_1 (Convolution2D)	(None, 224, 224, 32)	320	convolution2d_input_2[0][0]
convolution2d_2 (Convolution2D)	(None, 224, 224, 32)	9248	convolution2d_1[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 112, 112, 32)	0	convolution2d_2[0][0]
convolution2d_3 (Convolution2D)	(None, 112, 112, 64)	18496	maxpooling2d_1[0][0]
convolution2d_4 (Convolution2D)	(None, 112, 112, 64)	36928	convolution2d_3[0][0]
maxpooling2d_2 (MaxPooling2D)	(None, 56, 56, 64)	0	convolution2d_4[0][0]
convolution2d_5 (Convolution2D)	(None, 56, 56, 128)	73856	maxpooling2d_2[0][0]
convolution2d_6 (Convolution2D)	(None, 56, 56, 128)	147584	convolution2d_5[0][0]
convolution2d_7 (Convolution2D)	(None, 56, 56, 128)	147584	convolution2d_6[0][0]
maxpooling2d_3 (MaxPooling2D)	(None, 28, 28, 128)	0	convolution2d_7[0][0]
flatten_1 (Flatten)	(None, 100352)	0	maxpooling2d_3[0][0]
dense_1 (Dense)	(None, 128)	12845184	flatten_1[0][0]
dropout_1 (Dropout)	(None, 128)	0	dense_1[0][0]
dense_2 (Dense)	(None, 128)	16512	dropout_1[0][0]
dropout_2 (Dropout)	(None, 128)	0	dense_2[0][0]
dense_3 (Dense)	(None, 1)	129	dropout_2[0][0]
activation_1 (Activation)	(None, 1)	0	dense_3[0][0]

Total params: 13,295,841
Trainable params: 13,295,841
Non-trainable params: 0

2. CNN Model with Grayscale (1 channel) input, with 13 convolution layers with Adam optimizer.

Layer (type)	Output Shape	Param #	Connected to
convolution2d_8 (Convolution2D)	(None, 224, 224, 32)	320	convolution2d_input_3[0][0]
convolution2d_9 (Convolution2D)	(None, 224, 224, 32)	9248	convolution2d_8[0][0]
maxpooling2d_4 (MaxPooling2D)	(None, 112, 112, 32)	0	convolution2d_9[0][0]
convolution2d_10 (Convolution2D)	(None, 112, 112, 64)	18496	maxpooling2d_4[0][0]
convolution2d_11 (Convolution2D)	(None, 112, 112, 64)	36928	convolution2d_10[0][0]
maxpooling2d_5 (MaxPooling2D)	(None, 56, 56, 64)	0	convolution2d_11[0][0]
convolution2d_12 (Convolution2D)	(None, 56, 56, 128)	73856	maxpooling2d_5[0][0]
convolution2d_13 (Convolution2D)	(None, 56, 56, 128)	147584	convolution2d_12[0][0]
convolution2d_14 (Convolution2D)	(None, 56, 56, 128)	147584	convolution2d_13[0][0]
maxpooling2d_6 (MaxPooling2D)	(None, 28, 28, 128)	0	convolution2d_14[0][0]
convolution2d_15 (Convolution2D)	(None, 28, 28, 256)	295168	maxpooling2d_6[0][0]
convolution2d_16 (Convolution2D)	(None, 28, 28, 256)	590080	convolution2d_15[0][0]
convolution2d_17 (Convolution2D)	(None, 28, 28, 256)	590080	convolution2d_16[0][0]
maxpooling2d_7 (MaxPooling2D)	(None, 14, 14, 256)	0	convolution2d_17[0][0]
flatten_2 (Flatten)	(None, 50176)	0	maxpooling2d_7[0][0]
dense_4 (Dense)	(None, 256)	12845312	flatten_2[0][0]
dropout_3 (Dropout)	(None, 256)	0	dense_4[0][0]
dense_5 (Dense)	(None, 256)	65792	dropout_3[0][0]
dropout_4 (Dropout)	(None, 256)	0	dense_5[0][0]
dense_6 (Dense)	(None, 1)	257	dropout_4[0][0]
activation_2 (Activation)	(None, 1)	0	dense_6[0][0]

Total params: 14,820,705
Trainable params: 14,820,705
Non-trainable params: 0

3. CNN Model with RGB (3 channel) input, with 10 convolution layers with Adam optimizer.

Layer (type)	Output Shape	Param #	Connected to
convolution2d_1 (Convolution2D)	(None, 224, 224, 32)	896	convolution2d_input_4[0][0]
convolution2d_2 (Convolution2D)	(None, 224, 224, 32)	9248	convolution2d_1[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 112, 112, 32)	0	convolution2d_2[0][0]
convolution2d_3 (Convolution2D)	(None, 112, 112, 64)	18496	maxpooling2d_1[0][0]
convolution2d_4 (Convolution2D)	(None, 112, 112, 64)	36928	convolution2d_3[0][0]
maxpooling2d_2 (MaxPooling2D)	(None, 56, 56, 64)	0	convolution2d_4[0][0]
convolution2d_5 (Convolution2D)	(None, 56, 56, 128)	73856	maxpooling2d_2[0][0]
convolution2d_6 (Convolution2D)	(None, 56, 56, 128)	147584	convolution2d_5[0][0]
convolution2d_7 (Convolution2D)	(None, 56, 56, 128)	147584	convolution2d_6[0][0]
maxpooling2d_3 (MaxPooling2D)	(None, 28, 28, 128)	0	convolution2d_7[0][0]
flatten_1 (Flatten)	(None, 100352)	0	maxpooling2d_3[0][0]
dense_1 (Dense)	(None, 128)	12845184	flatten_1[0][0]
dropout_1 (Dropout)	(None, 128)	0	dense_1[0][0]
dense_2 (Dense)	(None, 128)	16512	dropout_1[0][0]
dropout_2 (Dropout)	(None, 128)	0	dense_2[0][0]
dense_3 (Dense)	(None, 1)	129	dropout_2[0][0]
activation_1 (Activation)	(None, 1)	0	dense_3[0][0]

Total params: 13,296,417
Trainable params: 13,296,417
Non-trainable params: 0

4. CNN Model with RGB (3 channel) input, with 13 convolution layers with Adam optimizer.

Layer (type)	Output Shape	Param #	Connected to
convolution2d_8 (Convolution2D)	(None, 224, 224, 32)	896	convolution2d_input_6[0][0]
convolution2d_9 (Convolution2D)	(None, 224, 224, 32)	9248	convolution2d_8[0][0]
maxpooling2d_4 (MaxPooling2D)	(None, 112, 112, 32)	0	convolution2d_9[0][0]
convolution2d_10 (Convolution2D)	(None, 112, 112, 64)	18496	maxpooling2d_4[0][0]
convolution2d_11 (Convolution2D)	(None, 112, 112, 64)	36928	convolution2d_10[0][0]
maxpooling2d_5 (MaxPooling2D)	(None, 56, 56, 64)	0	convolution2d_11[0][0]
convolution2d_12 (Convolution2D)	(None, 56, 56, 128)	73856	maxpooling2d_5[0][0]
convolution2d_13 (Convolution2D)	(None, 56, 56, 128)	147584	convolution2d_12[0][0]
convolution2d_14 (Convolution2D)	(None, 56, 56, 128)	147584	convolution2d_13[0][0]
maxpooling2d_6 (MaxPooling2D)	(None, 28, 28, 128)	0	convolution2d_14[0][0]
convolution2d_15 (Convolution2D)	(None, 28, 28, 256)	295168	maxpooling2d_6[0][0]
convolution2d_16 (Convolution2D)	(None, 28, 28, 256)	590080	convolution2d_15[0][0]
convolution2d_17 (Convolution2D)	(None, 28, 28, 256)	590080	convolution2d_16[0][0]
maxpooling2d_7 (MaxPooling2D)	(None, 14, 14, 256)	0	convolution2d_17[0][0]
flatten_2 (Flatten)	(None, 50176)	0	maxpooling2d_7[0][0]
dense_4 (Dense)	(None, 256)	12845312	flatten_2[0][0]
dropout_3 (Dropout)	(None, 256)	0	dense_4[0][0]
dense_5 (Dense)	(None, 256)	65792	dropout_3[0][0]
dropout_4 (Dropout)	(None, 256)	0	dense_5[0][0]
dense_6 (Dense)	(None, 1)	257	dropout_4[0][0]
activation_2 (Activation)	(None, 1)	0	dense_6[0][0]

Total params: 14,821,281
Trainable params: 14,821,281
Non-trainable params: 0

5. CNN Model with Grayscale (1 channel) input, with 10 convolution layers with RMSProp optimizer. The architecture was same which was used in 10 layer grayscale model with adam optimizer, the only difference being the optimizer.

6. CNN Model with RGB (3 channel) input, with 10 convolution layers with RMSProp optimizer. The architecture was same which was used in 10 layer RGB (3 channel) model with adam optimizer, the only difference being the optimizer.

7. CNN Model with RGB (3 channel) input, with 13 convolution layers with RMSProp optimizer. The architecture was same which was used in 13 layer RGB (3 channel) model with adam optimizer, the only difference being the optimizer.

In general a deep convolution neural networks were used with either the Adam optimizer or RMSProp optimizer whose results were analyzed. The input to these models was either a Grayscale image (1 channel) or RGB image (3 channel). The number of filters used in successive layers of 10 layer models are 32,32,64,64,128,128,128. Similarly, the number

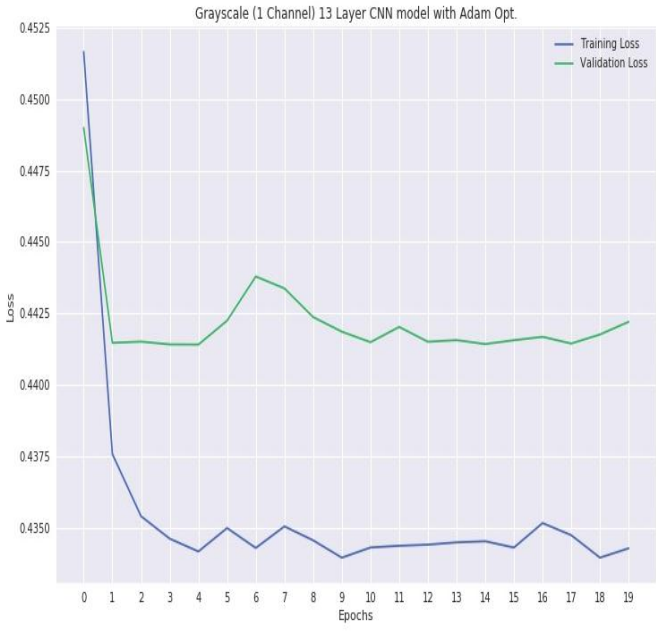
of filters used in successive layers of 13 layer models are 32,32,64,64,128,128,128,256,256,256. All the filters which were used had dimension of 3*3.

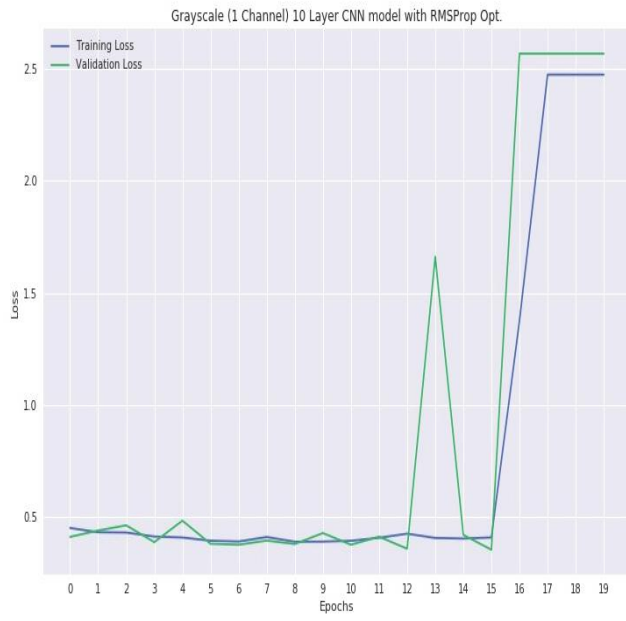
11. Refinement

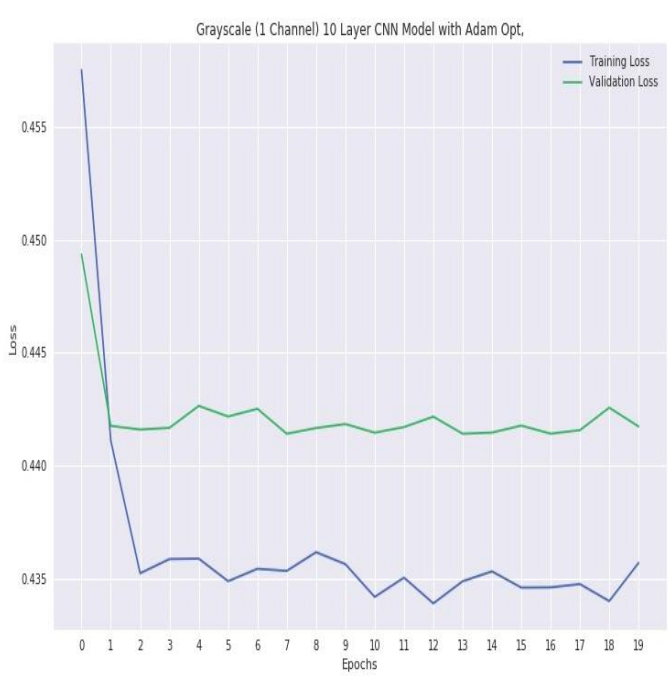
There were few steps which I followed to improve my model for better accuracy. I increased the number of layers in my CNN model with increased number of filters or kernels. I used different optimizer i.e RMSProp which improved the accuracy of my model. There was a significant improvement in accuracy in our final model which had 13 layers and used RMS prop with RGB image as input. Yes the initial solution was found and clearly reported. The accuracy of our benchmark model was ~84.4% but after above refinement the accuracy of our final model was ~94.4%.

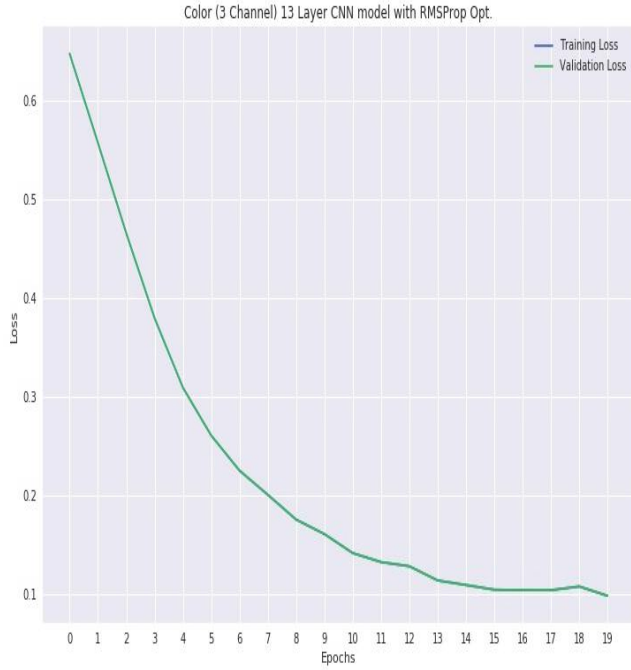
12. Model Evaluation and Validation

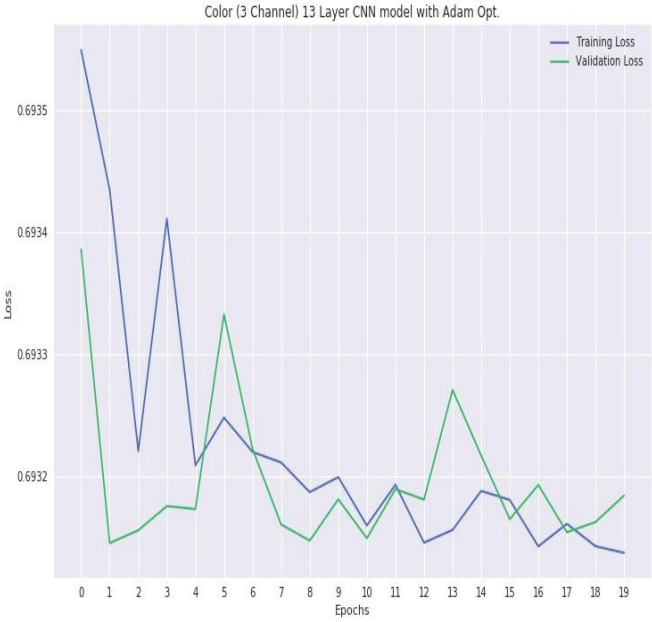
I created 7 models which were different from each other in terms of input data, optimizer used or the number of layers used in the CNN model. Below are the various plots which contains the loss (training and validation loss) for each epoch.

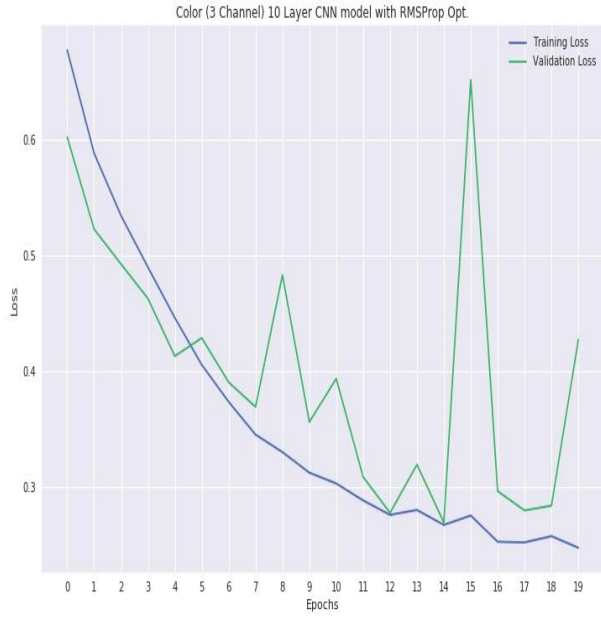




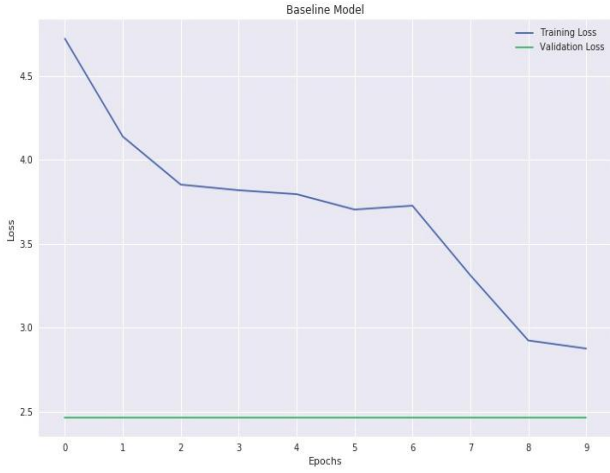

Grayscale (1 Channel) 10 Layer CNN Model with Adam Opt,
Training Loss
Validation Loss
Loss
Epochs


Color (3 Channel) 13 Layer CNN model with RMSProp Opt.
Training Loss
Validation Loss
Loss
Epochs


Color (3 Channel) 13 Layer CNN model with Adam Opt.
Training Loss
Validation Loss
Loss
Epochs


Color (3 Channel) 10 Layer CNN model with RMSProp Opt.
Training Loss
Validation Loss
Loss
Epochs


Color (3 Channel) 10 Layer CNN model with Adam Opt.
Training Loss
Validation Loss
Loss
Epochs


Baseline Model
Training Loss
Validation Loss
Loss
Epochs

As we can see from the above graphs that some of the model have very high loss values and will give poor results compared to benchmark model. Some model tends to overfit and will give poor result on unseen data.
Yes the final model reasonable and aligning with solution expectations. The final parameters of the model appropriate. The final model which I chose is a deep CNN with 13 layers. The input to this CNN model is a RGB image (3 channel input). The number of filters at each successive layers are 32,32,64,64,128,128,128,256,256,256. The number of neurons at the both fully connected layers are 256 neurons respectively. This model uses RMS prop as the optmizer with learning rate of 1e-4. This model gives accuracy of ~94.4% on unseen data which is 10% more than the baseline model. This model is very robust. This model was chosen after carefully optimizing number of layers and optimizer function. Yes the result found from the model can be trusted.

The accuracy plots for various models in comparison with baseline or benchmark model are given below:

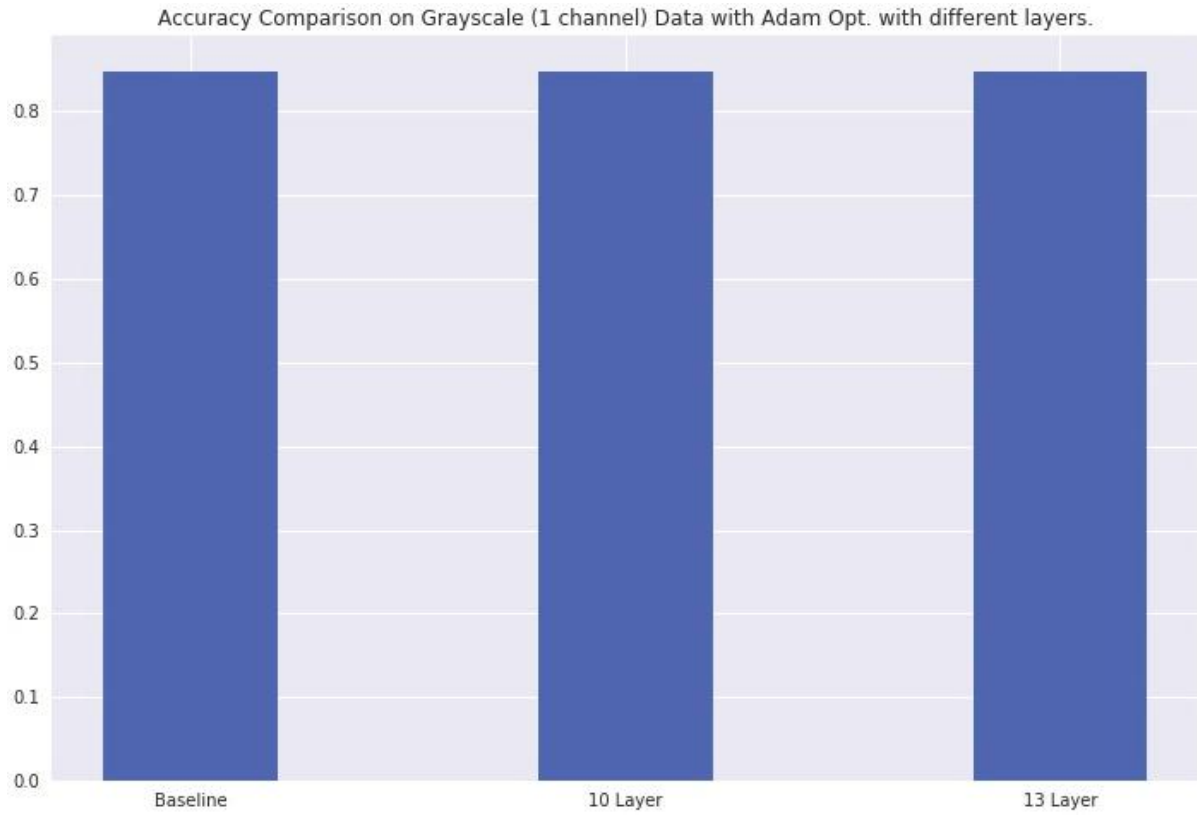


Fig. Accuracy comparison

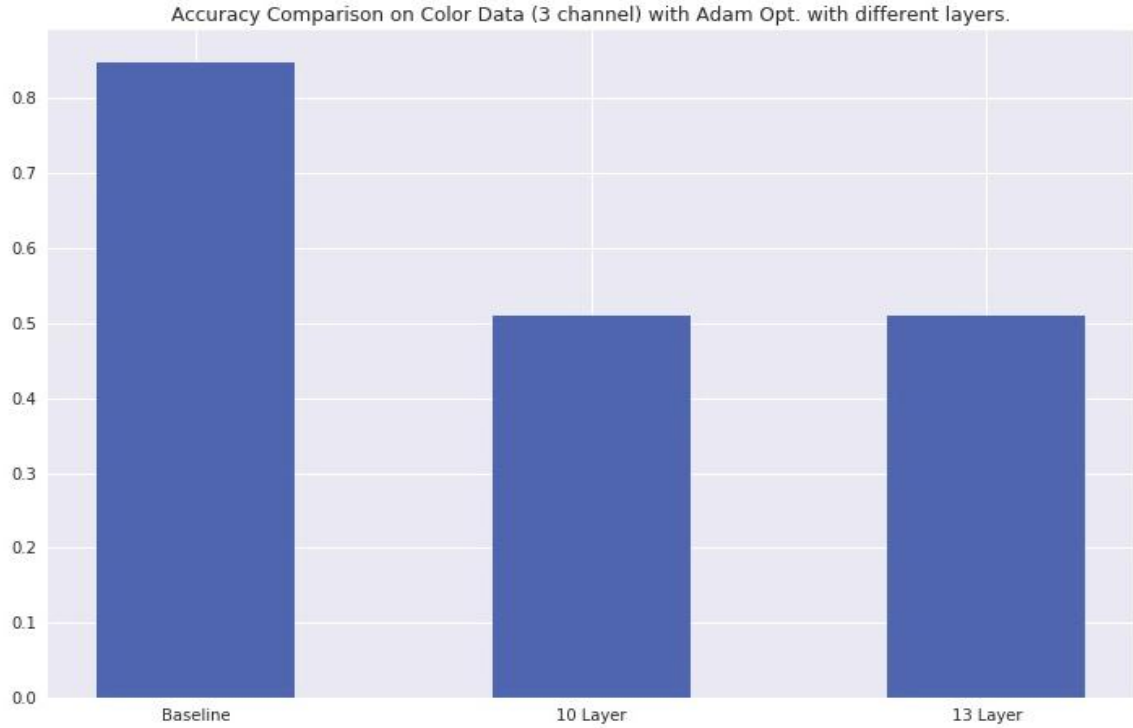


Fig. Accuracy Comparison

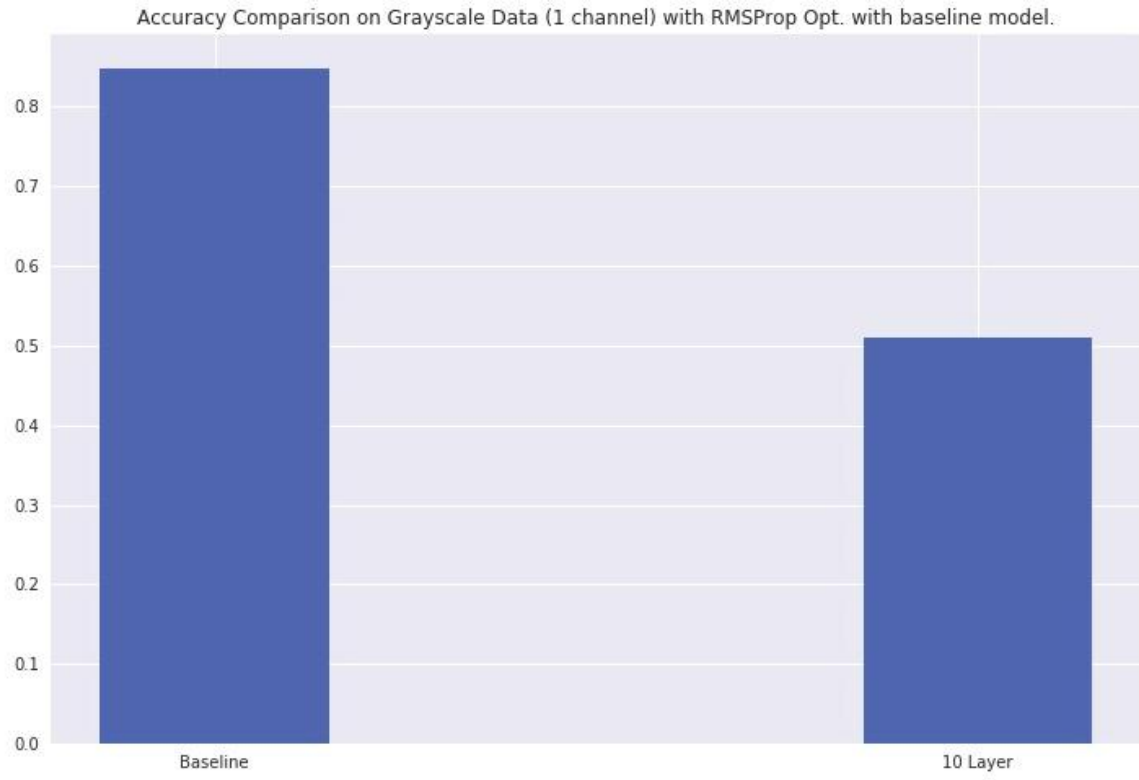


Fig. Accuracy Comparison

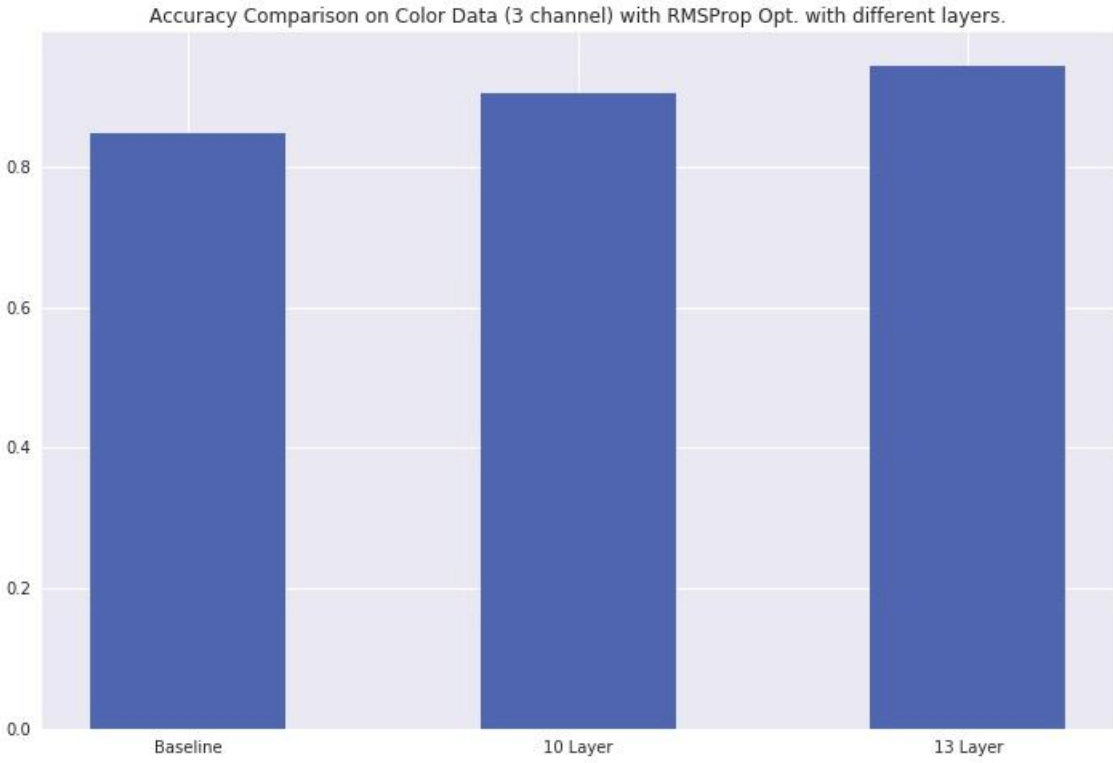


Fig. Final Model Accuracy Comparison

As we can see from the above plot which depicts final model accuracy comparison. My model performs better than baseline model. Th loss curve for the final model is given below:

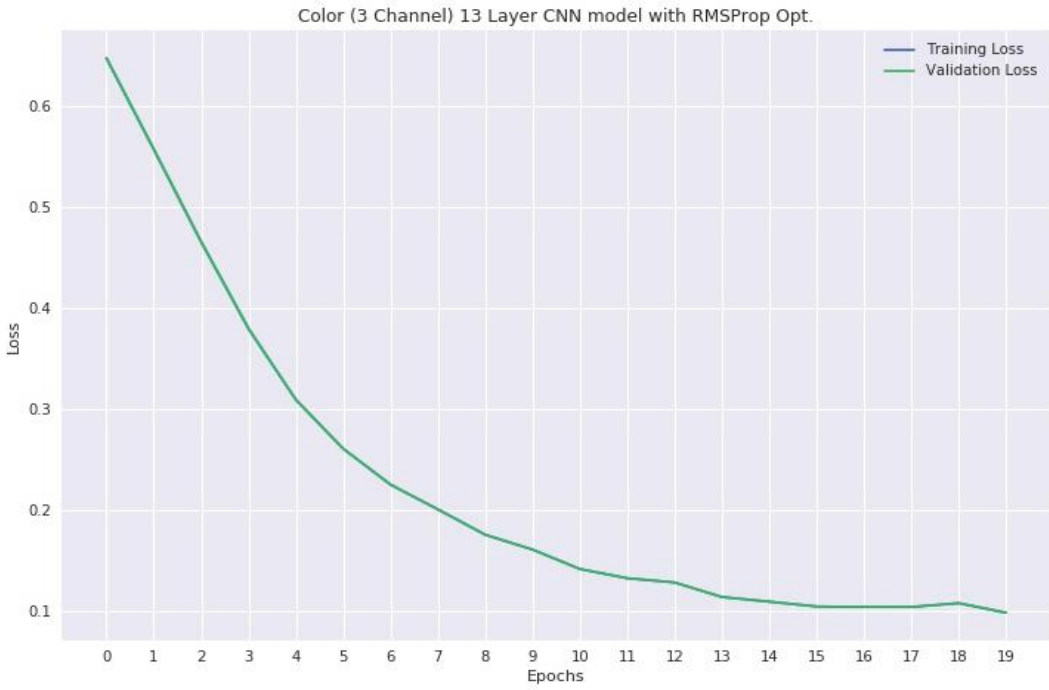


Fig. Final Model Loss Curve

13. Justification

The final results were found to be better than the benchmark result reported earlier. This is also evident from the below the plot.

Fig. Final Model Accuracy Comparison

Yes I have thoroughly analyzed and discussed the final solution. Yes the final solution has a significant improvement over the benchmark model. There is a improvement of around ~10% accuracy from this we can say that final solution is significant enough to solve the problem.

14. Conclusion

Free-Form Visualization

Visualization of the dataset and the model has been done widely. There are multiple plots like loss vs epoch plots for each model which helps us to understand each model better. Similarly the accuracy plot also shows us which model performs better than others.

Reflection

I have trained multiple models of CNN with varying input and varying depth. The input image of the final model had size of 224*224 and 3 channel. There were 13 layers in the final model which ad 2 fully connected layers. The accuracy of our final model is around ~94%. The most difficult aspect of this project was computational resources. As we grow our CNN deeper, we need more computational resources which is generally not easily available. Another interesting part of the project was even a very shallow CNN

(benchmark model) gave a accuracy of around 84% which is very good for a benchmark model, this shows us the power of CNN, which is able to extract latent features easily and gives us very good result from the beginning itself.

Improvement

Even though the results are promising, the accuracy score is far from the best model which is was made by Pierre Sermanet [2] which had an accuracy of ~98.9%. There are multiple things which we can do to improve our model like increasing the number of layers on our CNN model along with using more no of filters at each level. Increasing the number of layers requires much more computation which can only be done using high end GPU link Nvidia Pascal.

15. Refrences

[1] https://www.kaggle.com/c/dogs-vs-cats
[2] http://cs.nyu.edu/~sermanet/
[3] http://research.microsoft.com/en-us/um/redmond/projects/asirra
[4] https://en.wikipedia.org/wiki/Convolutional_neural_network