DEEP LEARNING FOR COMPUTER VISION AND NLP PROJECT DOG BREED CLASSIFICATION WITH LIMITED SAMPLES

PROJECT REPORT

Jayanth Prakash Kulkarni[*]
School of Computer Science
University of Windsor
401 Sunset Ave, Windsor, ON N9B 3P4
kulka112@uwindsor.ca

Yicheng Lu[*]
School of Computer Science
University of Windsor
401 Sunset Ave, Windsor, ON N9B 3P4
lu16a@uwindsor.ca

Sandeep Saradhi[*]
School of Computer Science
University of Windsor
401 Sunset Ave, Windsor, ON N9B 3P4
saradhi@uwindsor.ca

December 12, 2018

ABSTRACT

The goal of the project is to build a Dog breed classifier With limited training samples(Less than 100 for each class) using convolutional neural networks and other deep learning methods. We aim to achieve better performance of our model by combining different state-of-the-art models and by tuning the hyperparameters of the deep learning architecture.

***K*eywords** Deep Learning, Convolutional Neural Network, Transfer learning, Hyperparameter Tuning, Regularization

1 Problem Statement

The dataset contatins a training set and a test set of images of dogs. The dataset comprises 120 breeds of dogs. The goal of the competition is to create a classifier capable of determining a dog's breed from a image.[1] The train data and test data contains around 10k images.The challenging part is that there are around 100 labeled images per class which is very less for a neural network to

[*]All Authors have contributed equally

train from scratch. We first propose our own neural network model and improve the accuracy of the model by tuning the hyperparameters and resorting to proven methods like data augmentation, batch normalization and regularization. We further try to incorporate transfer learning and as a result of which we achieve.

So the final research question is" Can we train a model with decent accuracy with very few training samples?"

2 Vanilla Convolutional neural network

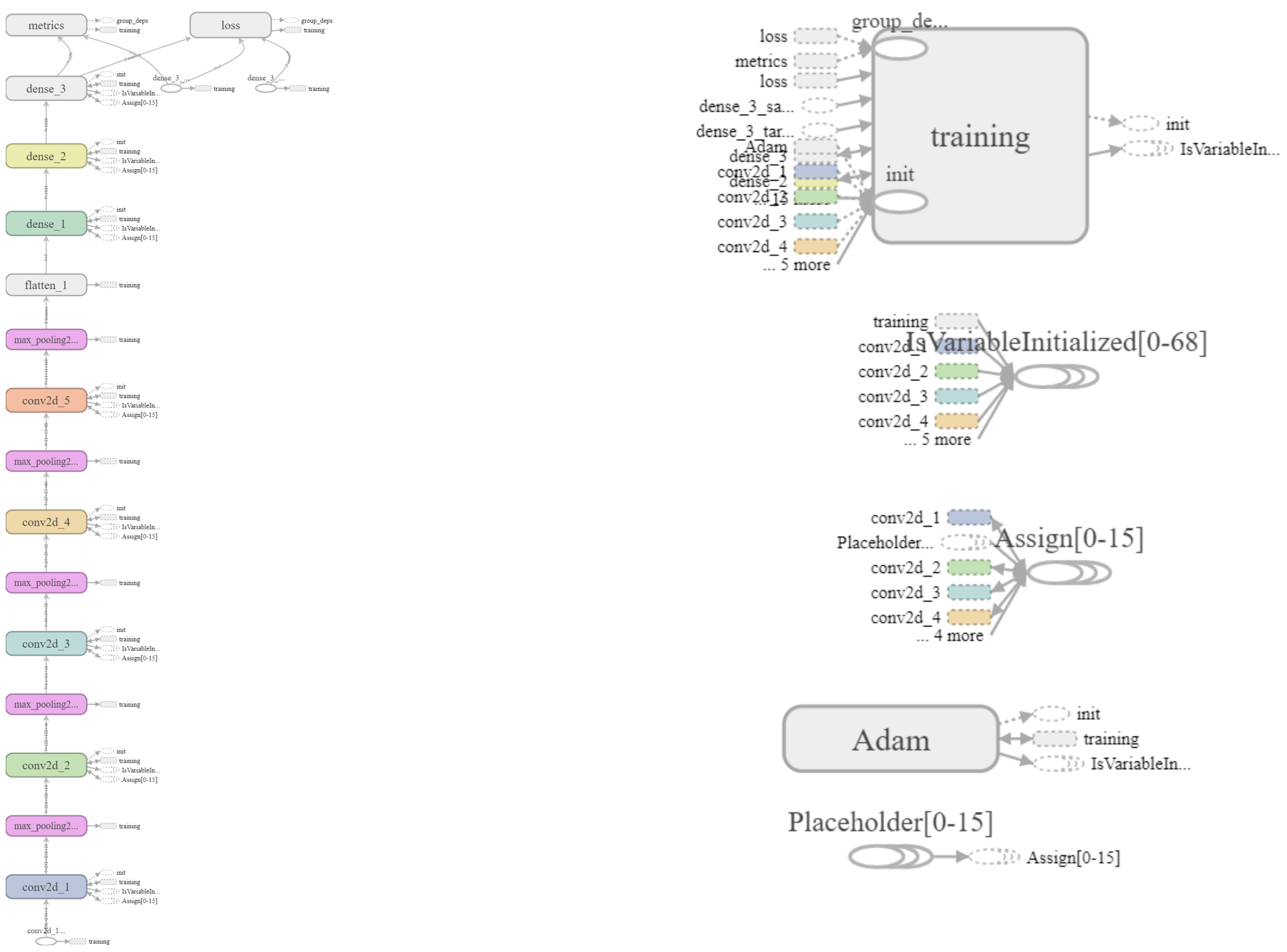
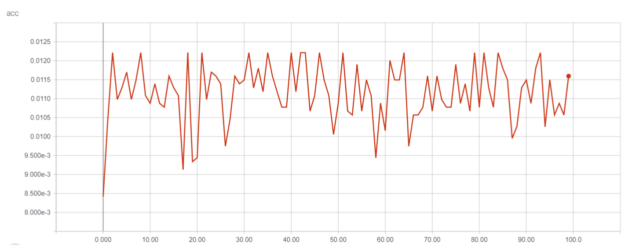
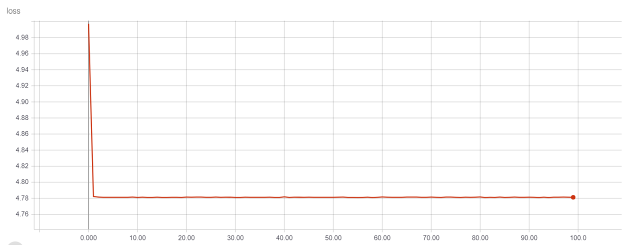


Figure 1: Tensorboard Graph for the vanilla architecture

This is the convolutional neural network which we trained and costructed from scratch. It is a basic convolutional neural network without any regularization and batch normalization. The model is designed as shown above with properly tuned hyperparameters.The 10k images from our dataset were split into training and validation in a ratio of 9:1 and the neural network was trained and validated on the same. The following results were obtained.

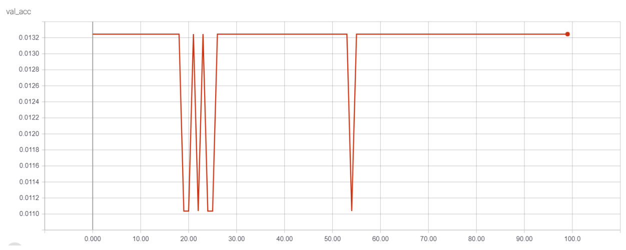


(a) Accuracy Plot

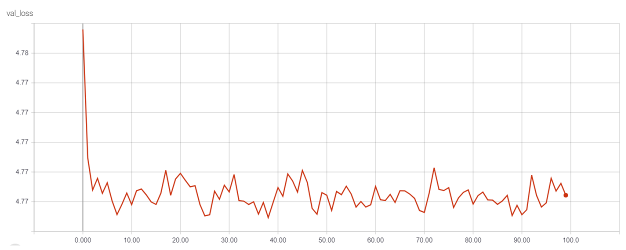


(b) Loss Plot

Figure 2: Training accuracy and loss on training split of original data



(a) Accuracy Plot



(b) Loss Plot

Figure 3: Training accuracy and loss on validation split of original data

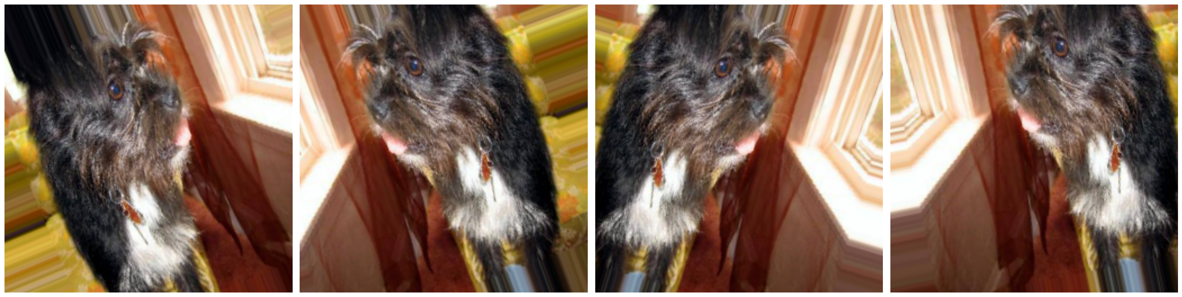
As we observe the the above figures that the accuracy on both the training and validation set is very poor. it was able to attain only 1% accuracy after 100 epochs of training.

3 Approaches with Results

As our training accuracy is hopelessly low, we have to try out various approaches to improve it. The methods which we decided to use are the following.

3.1 Data Augmentation

As the data contains very less number of samples for each class to train on, there was a need to create variance in the dataset by introducing some augmentations to the original data.



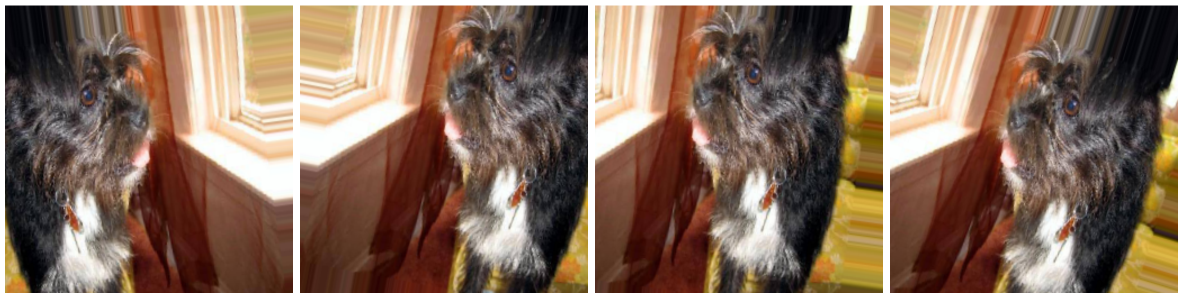
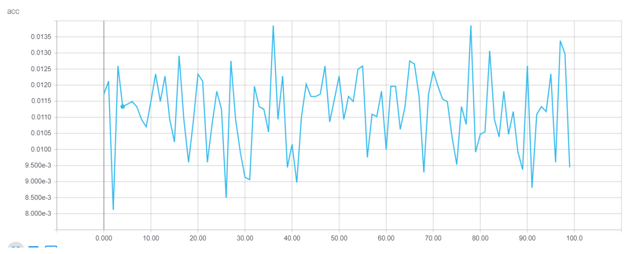
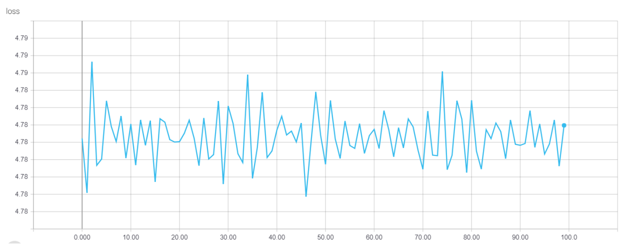


Figure 5: 8 variations on the original dog image

The neural network was then trained on this augmented data set with the following results.

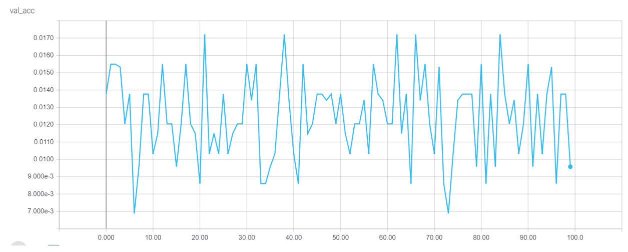


(a) Accuracy Plot

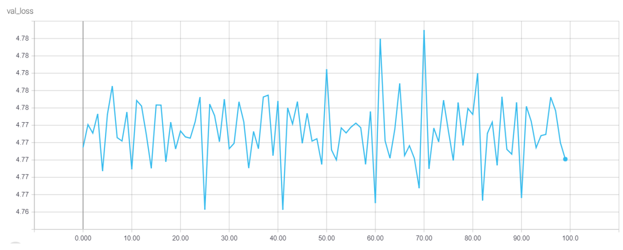


(b) Loss Plot

Figure 6: Training accuracy and loss on training split of original data



(a) Accuracy Plot



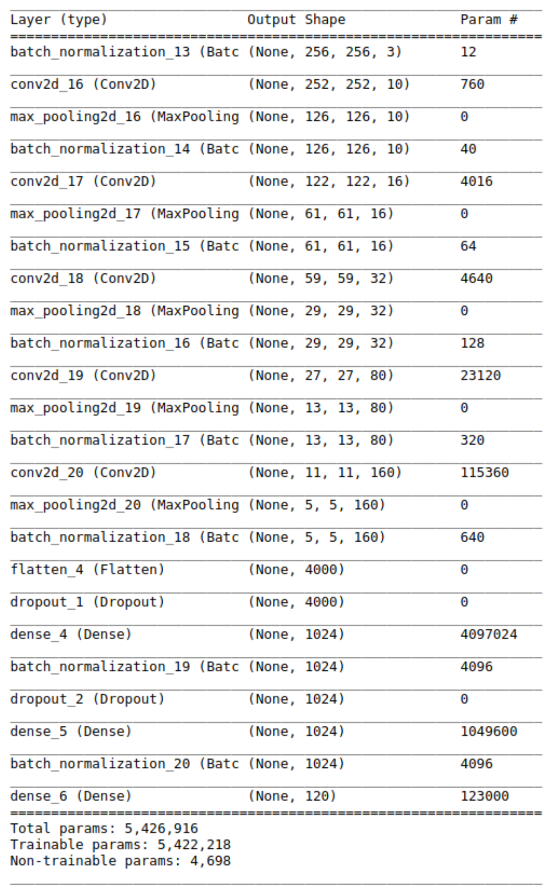
(b) Loss Plot

Figure 7: Training accuracy and loss on validation split of original data

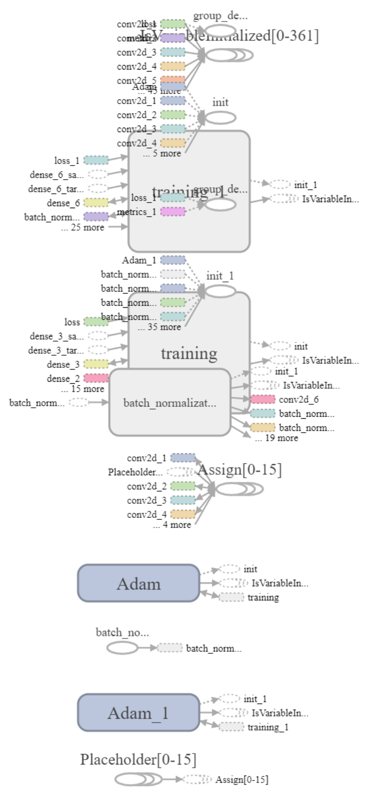
This augmentation helped the network to improve it's accuracy on the validation set, mainly because it was exposed larger variance of the data and had learnt the model on augmented versions of the training data.

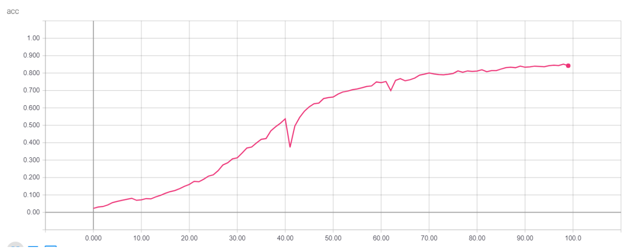
3.2 Advanced Structure

The neural network architecture lacked many of the state of the art techniques that enhance the performance of the algorithm. Batch normalization and regularization layers were added to improve the working of the model.

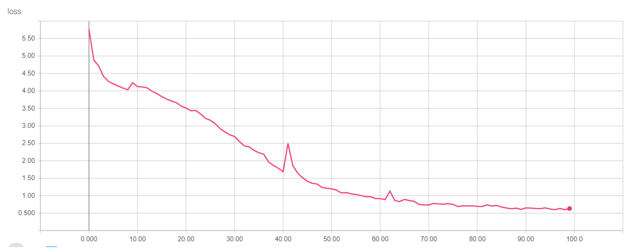


```
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_13 (Batc (None, 256, 256, 3)       12
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 252, 252, 10)      760
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 126, 126, 10)      0
_________________________________________________________________
batch_normalization_14 (Batc (None, 126, 126, 10)      40
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 122, 122, 16)      4016
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 61, 61, 16)        0
_________________________________________________________________
batch_normalization_15 (Batc (None, 61, 61, 16)        64
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 59, 59, 32)        4640
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 29, 29, 32)        0
_________________________________________________________________
batch_normalization_16 (Batc (None, 29, 29, 32)        128
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 27, 27, 80)        23120
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 13, 13, 80)        0
_________________________________________________________________
batch_normalization_17 (Batc (None, 13, 13, 80)        320
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 11, 11, 160)       115360
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 5, 5, 160)         0
_________________________________________________________________
batch_normalization_18 (Batc (None, 5, 5, 160)         640
_________________________________________________________________
flatten_4 (Flatten)          (None, 4000)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4000)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              4097024
_________________________________________________________________
batch_normalization_19 (Batc (None, 1024)              4096
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_5 (Dense)              (None, 1024)              1049600
_________________________________________________________________
batch_normalization_20 (Batc (None, 1024)              4096
_________________________________________________________________
dense_6 (Dense)              (None, 120)               123000
=================================================================
Total params: 5,426,916
Trainable params: 5,422,218
Non-trainable params: 4,698
_________________________________________________________________
```



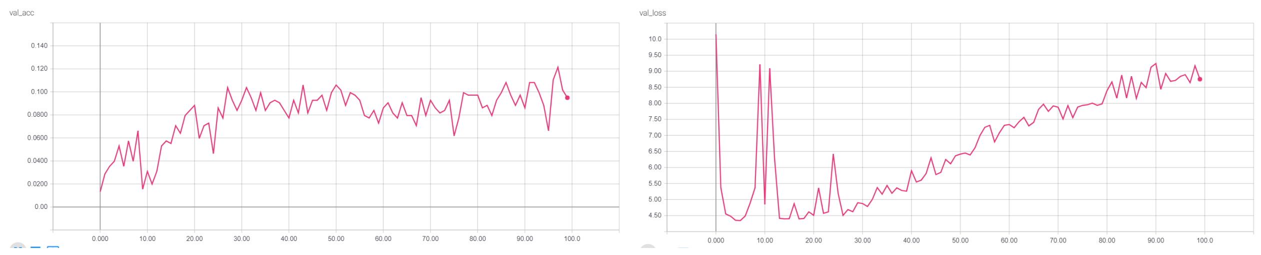


(a) Accuracy Plot



(b) Loss Plot

Figure 8: Training accuracy and loss on training split of original data with advanced architecture



(a) Accuracy Plot (b) Loss Plot

Figure 9: Training accuracy and loss on validation split of original data with advanced architecture

With the addition of bath normalization and dropout regularization, the training accuracy increased to 85%.

3.2.1 Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

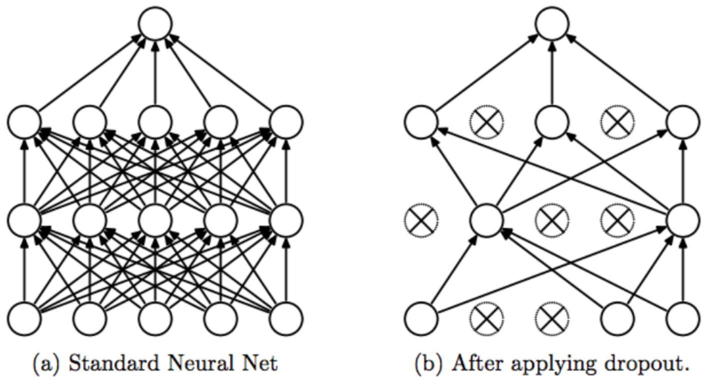
Batch normalization algorithm[2]

Batchnormalization scales the features such that there is no bias given to a particular feature. During the backpropogation stage the gradient will not oscillate and can converge to the optimality in faster time.

It also reduces the magnitude in which the future layers are affected by the gradients of the previous layers. This process takes care of the outliers in the dataset by normalizing them.

This step is crucial to our model as the features in some of the training images were too foreign with images including humans and other dogs in the same image.

3.2.2 Regularization



Dropout Regularization

Regularization is used to constrain network adaptation to the data at training time. It is mainly used to stop the model from overfitting to the data.

In our model we use a form of regularization called dropout. In Dropout we ignore and drop a part of the learned nodes in the previous layers. This is important to our specific model as some dog breeds have similar features and this process helps to avoid overfitting to a particular breed and leads to better interpretation the different breeds.

3.3 Hyperparameter Tuning

Hyperparamaters are the parameters which the model cannot learn and are personally set by the user. They have to be properly set according to the input size, model size and our specific needs.

The important hyperparameters in our model that have to be tuned are Learning rate, Filter size, Layer size and dropout. We had to run the model many times with various hyperparameter values before we achieved suitable values for them.

Since we have a lot of hyperparameters to tune, this is the main reason we use Adam optimizer for our model instead of a simple gradient descent optimizer as Adam uses a larger effective step size, and the algorithm will converge to this step size without fine tuning.

3.4 Transfer Learning

Transfer learning is a deep learning method in which the weights of a neural network is trained for a generic task like image classification, on a huge dataset and the model is stored. Then every layer of the neural network is frozen apart from the last few dense layers and the model is retrained for a new classification problem at hand.

More specifically, the gradients are not updated for any layer apart from the last few layers. This technique works well for image classification as the convolutional layers extract meaningful features like edges, lines and shapes. So this information can be transferred for different classification problems.This approach generally applied when there are less number of images to train a neural network from scratch.

In our specific model we have our very few training samples(Less than 100) for each class. It's very hard to train a model from scratch with these few samples, therefore we perform transfer learning to use pretrained weights from established models. We add our own dense layers on top of this network and use it to learn the features for the dog breed dataset. This greatly helps our accuracy as the new model can use what it's previously learnt to improve it's prediction rate on the dog breed dataset.

3.4.1 VGG16

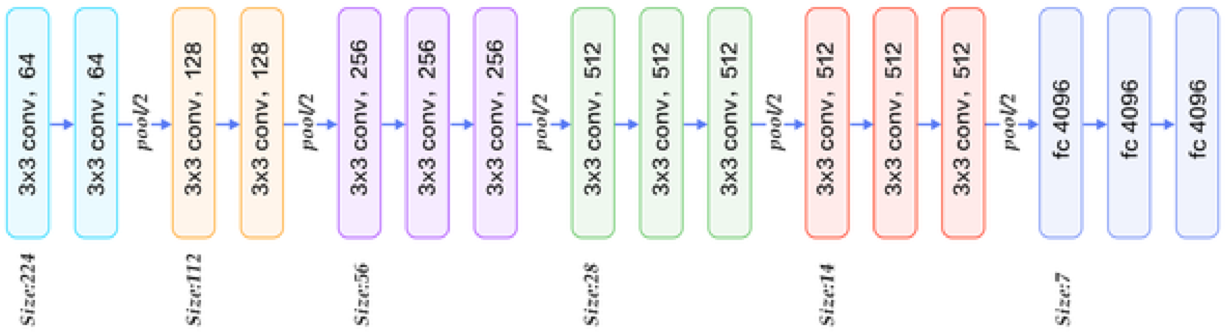


Figure 10: VGG16 architecture
[3]

VGG16 achieved the best ILSVRC in 2014. It scored first place on the image localization task and second place on the image classification task. We use this as our pre-trained model and retrained the dense layers for the dogbreed classification problem.

3.4.2 Transfer learning architecture

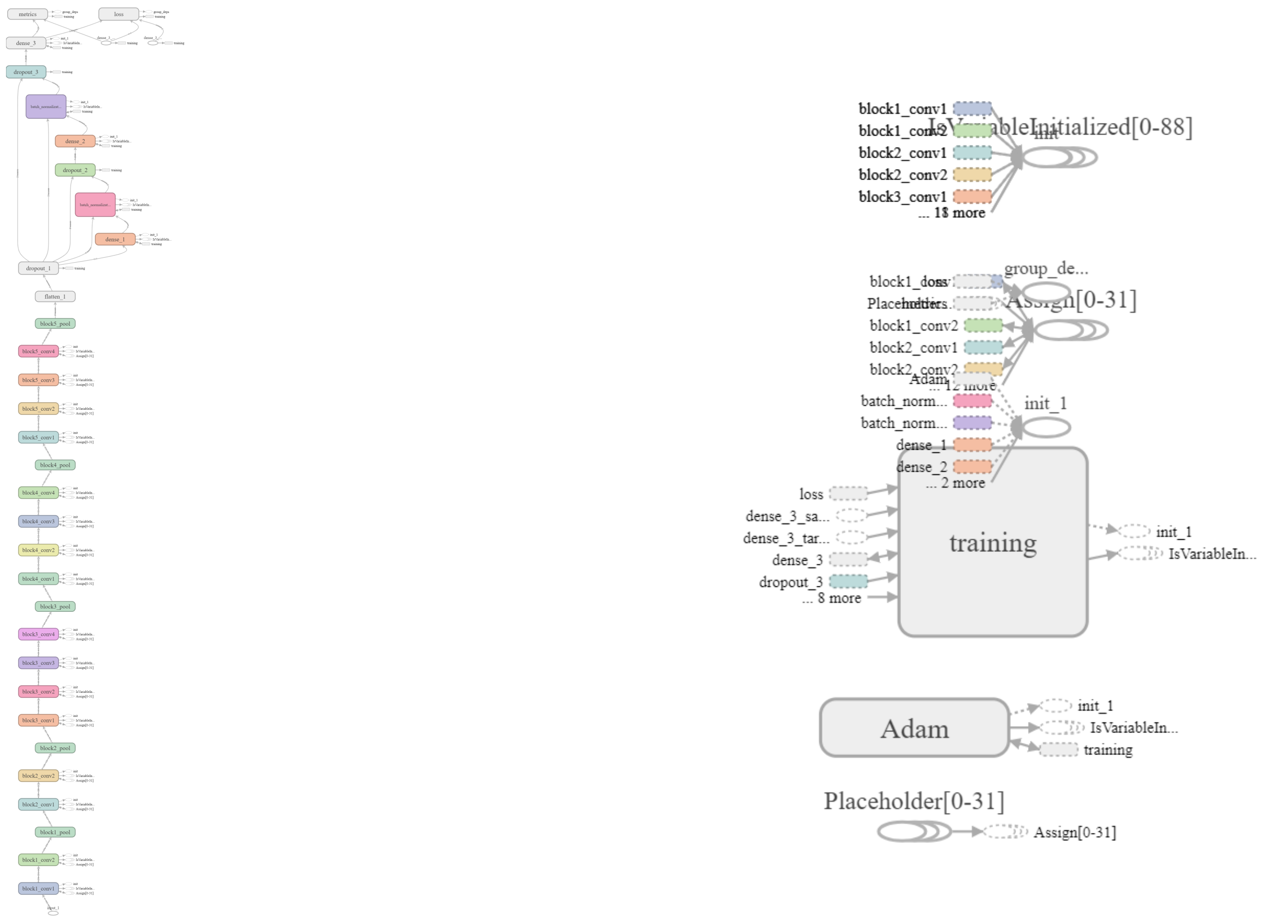
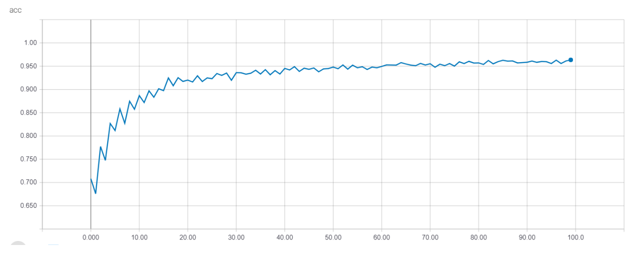
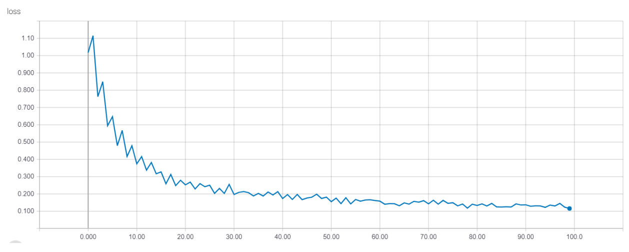


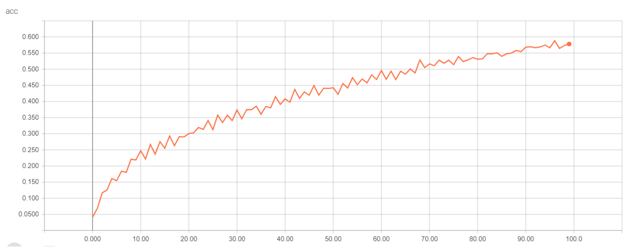
Figure 11: Final Transfer learning architecture

3.4.3 Transfer Learning Results

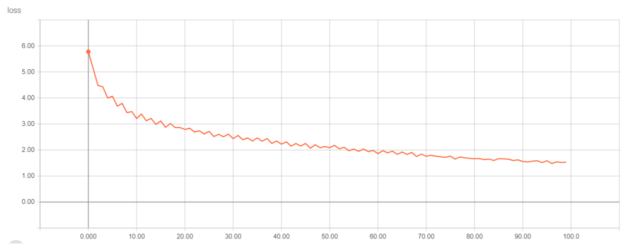



(a) Accuracy Plot (b) Loss Plot

Figure 12: Training accuracy and loss on training split of original data with Transfer learning from VGG16



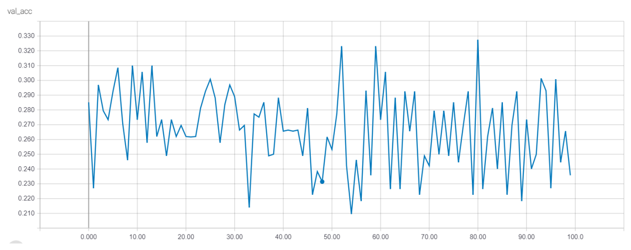
(a) Accuracy Plot



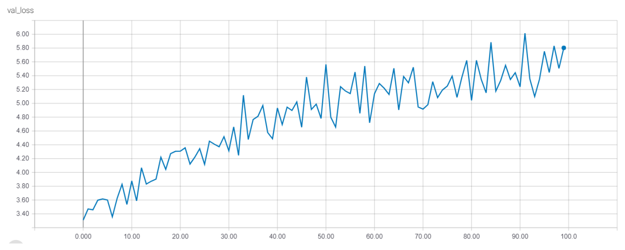
(b) Loss Plot

Figure 13: Validation accuracy and loss on validation split of original data with Transfer learning from VGG16

3.4.4 Transfer Learning Results with data augmentation

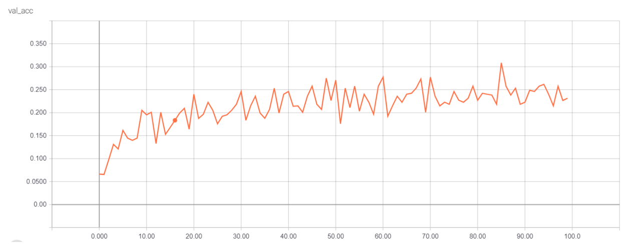


(a) Accuracy Plot

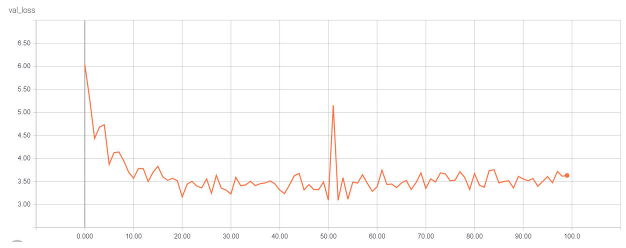


(b) Loss Plot

Figure 14: Training accuracy and loss on training split of augmented data with Transfer learning from VGG16



(a) Accuracy Plot



(b) Loss Plot

Figure 15: Validation accuracy and loss on validation split of augmented data with Transfer learning from VGG16

4 Conclusion

As observed, we have improved the accuracy of the model dramatically by performing image augmentation and transfer learning.

These methods have made up for the lack of total training samples available to the model. We also analyzed the positive effects of regularization and batch normalizaton on the model. The various hyperparameters like learning rate, Filter size and layer size have also been experimented on by trying out different values and tuning them to fit the model.

Our final accuracy on the training dataset is 97.2% and on the validation dataset is 33%

5 Future Work

We can further enhance the model by trying out transfer learning on various other established model like GoogleNet, AlexNet, etc.

We can also try to manually increasing the dataset by adding our own images to it.

References

[1] Kaggle. Dogbreed classification. In https://www.kaggle.com/c/dog-breed-identification/data.

[2] Karl N. Batch normalization. In https://gab41.lab41.org/batch-normalization-what-the-hey-d480039a9e3b,

[3] Karen Simonyan Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In https://arxiv.org/pdf/1409.1556.pdf.