Data Augmentation

Introduction

The purpose of this section was to test the impact of data augmentation on the CNN learning speed and eventual accuracy. Due to computational constraints, we only allowed 15 epochs. Note that the neural network used in this section was taken from here. This fits in with the general literature in that most papers are concerned with creating more intuitive and accurate frameworks for the CNN and not necessarily finding the ideal data augmentation to be fed into said networks, which is what we are looking at in this section.

Details and Methodology

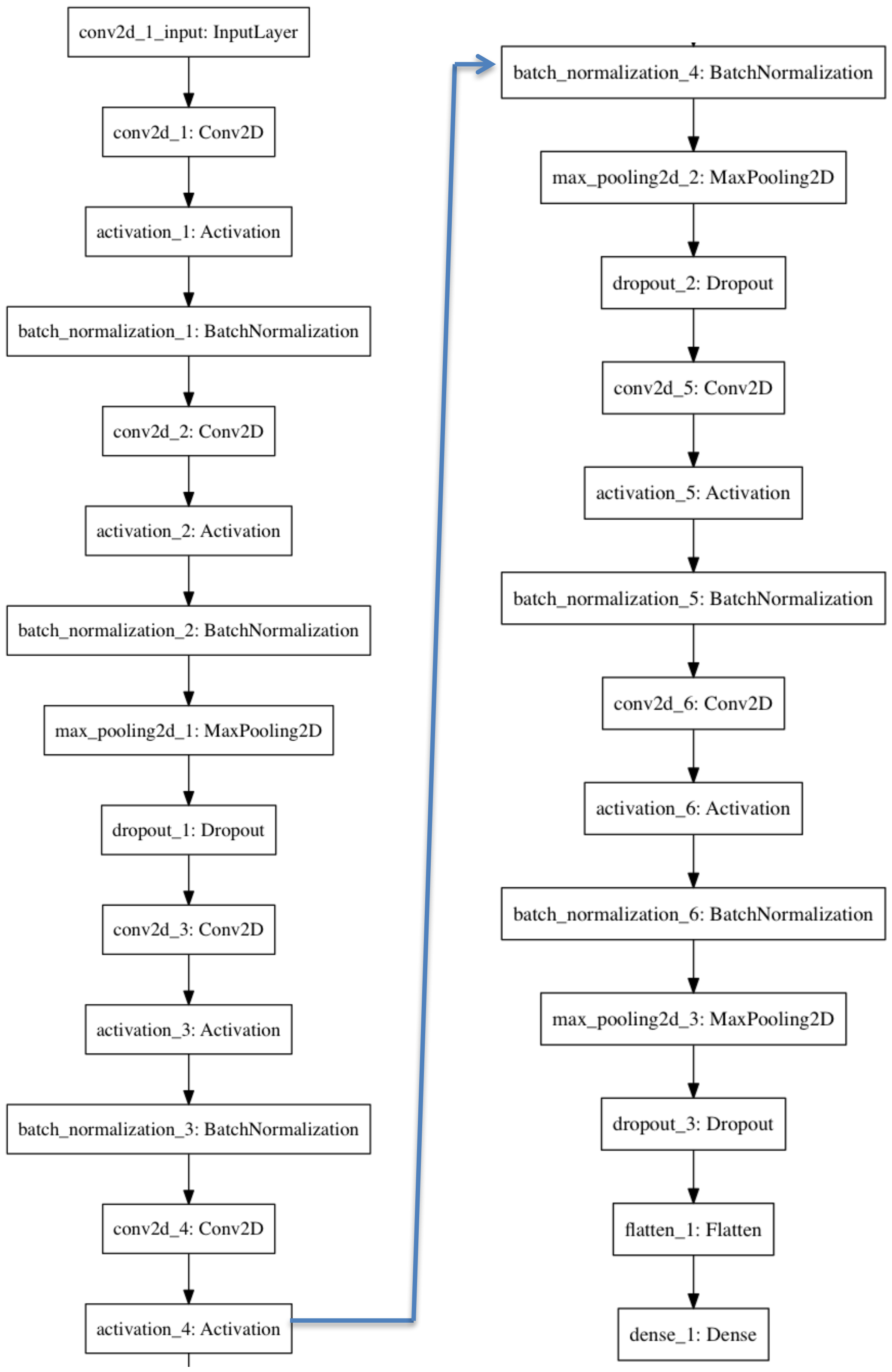
In terms of data augmentation we allowed randomization via the function ImageDataGenerator() in Keras with the following augmentation possibilities:

- Rotation_angle up to 50 degrees
- Width shift range up to 50% of the width range
- Height shift range up to 50% of the height range
- Horizontal flipping

Specifically this was done by allowing i to go from 0 to 10 and using this code:

```
datagen = ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=50/i if i != 0 else 0,
    width_shift_range=0.5/i if i != 0 else 0,
    height_shift_range=0.5/i if i != 0 else 0,
    horizontal_flip=True if i != 0 else False,
    vertical_flip=False
    )
datagen.fit(x_train)
```

And the model summary is below:


conv2d_1_input: InputLayer
conv2d_1: Conv2D
activation_1: Activation
batch_normalization_1: BatchNormalization
conv2d_2: Conv2D
activation_2: Activation
batch_normalization_2: BatchNormalization
max_pooling2d_1: MaxPooling2D
dropout_1: Dropout
conv2d_3: Conv2D
activation_3: Activation
batch_normalization_3: BatchNormalization
conv2d_4: Conv2D
activation_4: Activation
batch_normalization_4: BatchNormalization
max_pooling2d_2: MaxPooling2D
dropout_2: Dropout
conv2d_5: Conv2D
activation_5: Activation
batch_normalization_5: BatchNormalization
conv2d_6: Conv2D
activation_6: Activation
batch_normalization_6: BatchNormalization
max_pooling2d_3: MaxPooling2D
dropout_3: Dropout
flatten_1: Flatten
dense_1: Dense

Constraints

However, due to time constraints, we were unable to isolate the various augmentation parameters (i.e., if we had more computational power we would have run the augmentation parameters in every possible combination such as individually and with all possible combinations of the other augmentation parameters to truly determine what is the ideal augmentation for training). And so, it is possible that some things held constant in all augmentation runs such as horizontal flipping of images is hindering all results. Furthermore, we would've liked to run the model for more iterations (i.e. more than 15 epochs), but even 15 epochs took almost 2 hours to run each iteration.

Results

Training

In terms of results, from the training error (after each epoch) displayed below, we see that data augmentation actually decreases the rate at which the model learns and the final accuracy it reaches.

	Degree of Augmentation (i)										
Epoch #	10	9	8	7	6	5	4	3	2	1	0
1	26.1%	36.2%	38.7%	41.8%	42.6%	45.7%	45.2%	45.2%	45.0%	46.6%	49.4%
2	34.2%	46.1%	52.1%	57.2%	57.5%	60.8%	60.2%	62.3%	61.8%	62.3%	65.9%
3	37.5%	50.1%	58.1%	62.5%	63.3%	65.3%	65.4%	67.9%	66.7%	67.7%	71.6%
4	39.2%	53.4%	60.9%	66.1%	65.5%	67.2%	67.9%	70.8%	68.6%	70.1%	74.9%
5	40.6%	55.3%	63.2%	68.8%	67.9%	69.1%	70.8%	72.9%	70.6%	71.4%	76.7%
6	41.2%	58.2%	65.8%	69.9%	69.7%	71.0%	72.4%	74.3%	72.7%	73.0%	78.0%
7	42.3%	59.6%	66.2%	71.1%	71.0%	72.1%	73.8%	75.0%	74.3%	74.1%	79.3%
8	43.0%	61.2%	66.6%	70.7%	72.2%	73.0%	74.8%	76.4%	73.9%	74.7%	79.8%
9	44.7%	62.2%	67.9%	72.0%	72.9%	74.1%	75.6%	76.8%	75.9%	76.1%	80.8%
10	45.2%	63.9%	69.6%	73.4%	74.3%	74.6%	76.4%	77.4%	76.6%	76.8%	81.7%
11	44.5%	64.7%	70.3%	73.9%	74.6%	75.3%	77.1%	78.2%	77.0%	78.0%	82.1%
12	46.3%	65.1%	71.2%	74.9%	75.5%	75.5%	77.8%	78.5%	78.0%	78.7%	82.9%
13	47.6%	66.3%	71.1%	75.2%	75.6%	75.7%	78.4%	79.0%	78.5%	79.3%	83.4%
14	49.4%	66.1%	71.6%	75.6%	76.2%	76.7%	79.1%	79.3%	78.8%	79.7%	83.8%
15	51.5%	66.9%	72.2%	76.1%	76.6%	77.5%	79.5%	79.7%	79.2%	80.1%	84.3%

Testing

However, the testing error (after each epoch below) seems to suggest that the some degree of data augmentation actually improves final accuracy (i.e. by augmenting the training data, we actually improve generalization of the model, which does make intuitive sense). However, rate of learning for the test error is indeed slower with data augmentation, which does make sense as well since we are adding random noise which the model takes time to adjust to.

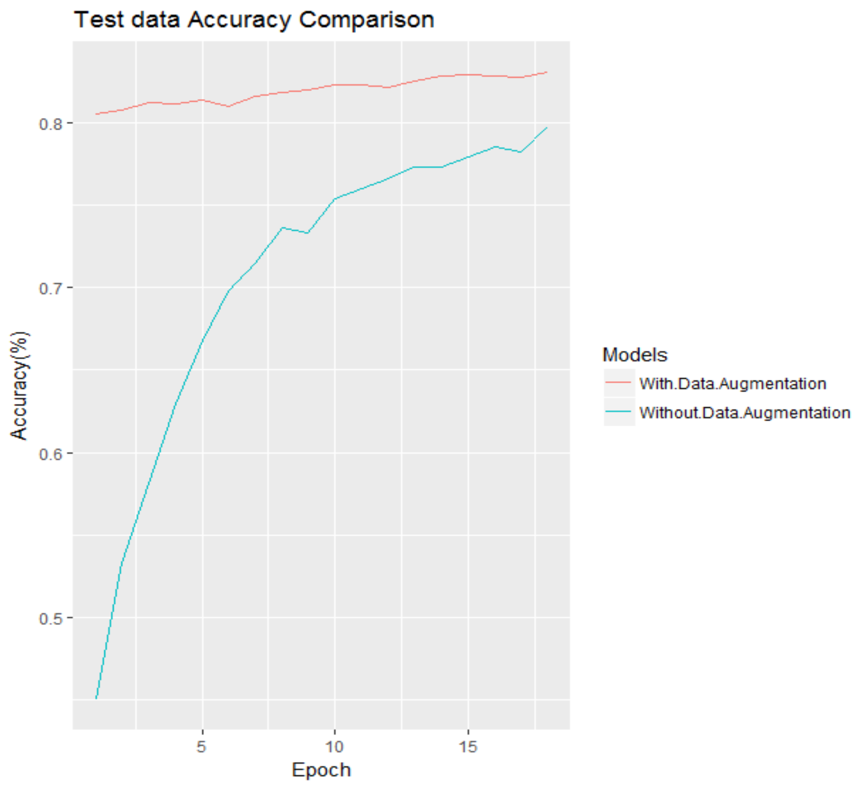
	Degree of Augmentation (i)										
Epoch #	10	9	8	7	6	5	4	3	2	1	0
1	37.1%	52.9%	47.4%	58.4%	53.9%	61.4%	61.7%	61.1%	59.4%	61.6%	64.9%
2	42.3%	56.6%	60.1%	65.1%	66.6%	69.1%	67.6%	68.8%	69.2%	69.0%	71.7%
3	46.7%	59.1%	62.3%	68.5%	70.5%	72.0%	71.2%	73.7%	71.9%	73.1%	75.2%
4	47.7%	61.5%	67.6%	74.0%	70.7%	69.4%	73.8%	74.9%	71.6%	73.5%	77.7%
5	50.5%	65.2%	71.9%	71.4%	68.9%	71.9%	75.6%	77.2%	74.8%	75.6%	77.8%
6	50.3%	67.0%	73.8%	76.9%	74.2%	76.2%	74.8%	78.7%	76.7%	75.7%	77.3%
7	47.5%	68.6%	72.4%	73.3%	77.6%	78.0%	79.1%	78.5%	78.1%	73.3%	79.4%
8	51.3%	71.0%	74.1%	77.7%	73.8%	78.2%	79.6%	80.3%	78.8%	78.6%	80.4%
9	56.4%	70.1%	74.9%	77.0%	78.3%	79.0%	79.3%	80.2%	78.6%	80.1%	79.0%
10	53.7%	72.2%	76.9%	78.2%	78.6%	79.8%	80.8%	82.1%	80.3%	79.8%	82.5%
11	51.9%	73.4%	77.3%	79.9%	80.1%	79.0%	80.8%	81.9%	80.8%	80.5%	80.8%
12	58.7%	73.8%	73.3%	79.5%	80.5%	80.2%	82.0%	82.8%	80.9%	81.8%	80.0%
13	53.7%	74.4%	76.7%	80.2%	79.0%	79.9%	83.4%	80.6%	81.2%	81.0%	80.4%
14	59.3%	75.3%	77.4%	81.4%	79.9%	81.6%	81.5%	83.1%	82.4%	82.5%	82.8%
15	61.5%	76.6%	78.3%	80.1%	81.8%	81.8%	82.9%	83.3%	82.3%	82.9%	82.1%

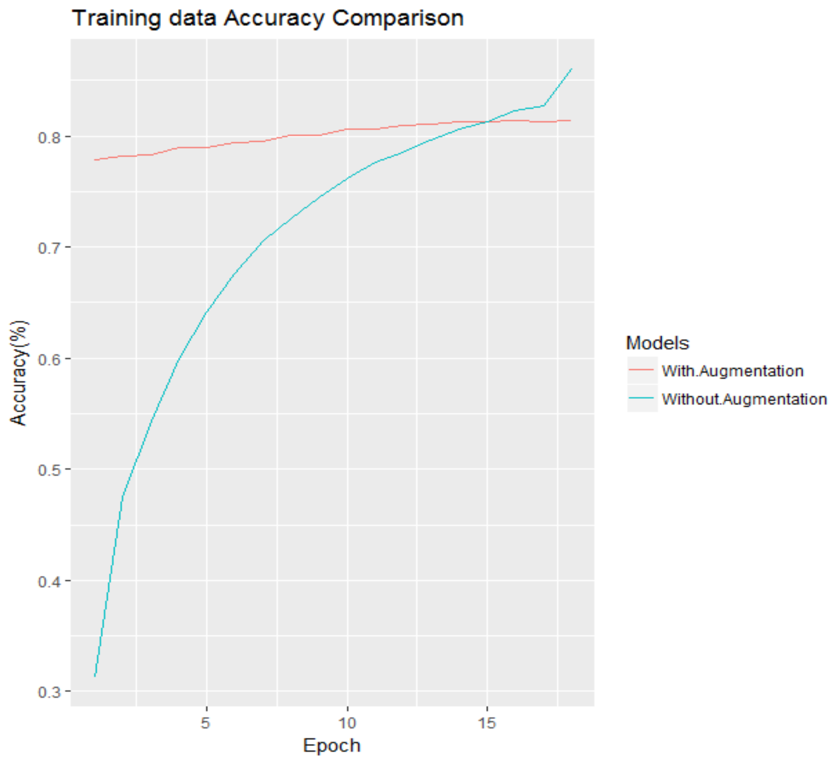
Conclusion

All in all, we conclude in this section that slight data augmentation, while not necessarily good for training accuracy, improves generalization of the CNN to the validation/test set, and thus is ideal. However, the extent to which the data should be augmented and in what ways will need to be further investigated along with allowing more iterations to allow full convergence of the model. Furthermore, these results are for a specific CNN, and other CNN would have to be tested as well to be sure that the results and findings here generalize.

Another model using pre-processing data

For this deeper network model, we also tried one of pre-processing data, image augmentation from the keras deep learning library. Due to the time and computational constraints, we only tried horizontal flipping and width and height shift up to 10%.
First, as we described previsouly, we create an image generator by calling the 'ImageDataGenerator()' function and input a list of parameters describing the alterations that we would like it to perform on the images. We then call the 'fit()' function on our image generator which will apply the changes to the image batch by batch.
We also used "dropout" technique, consists of setting to zero the dropout of each hidden layer with probability of 0.2. The neurons which are dropped out in this way do not contribute to the forward pass and do not participate in back-propagation. Therefore, dropout technique allows to learn more robust features with different random subsets of the other neurons. Without dropout, our network will exhibit substantial overfitting.
We also used a weight constraint of "maxnorm" which will scale the weight matrix buy the factor that reduces the norm to the parameter if the L2 Norm of our weights exceeds the parameter. This is another kind of regularization and with this technique, we can regularize directly. We used the keras code. This seems to work especially well in combination with a dropout layer.
(attached file:Deep learning model.ipynb)
(http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf)
(https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf)

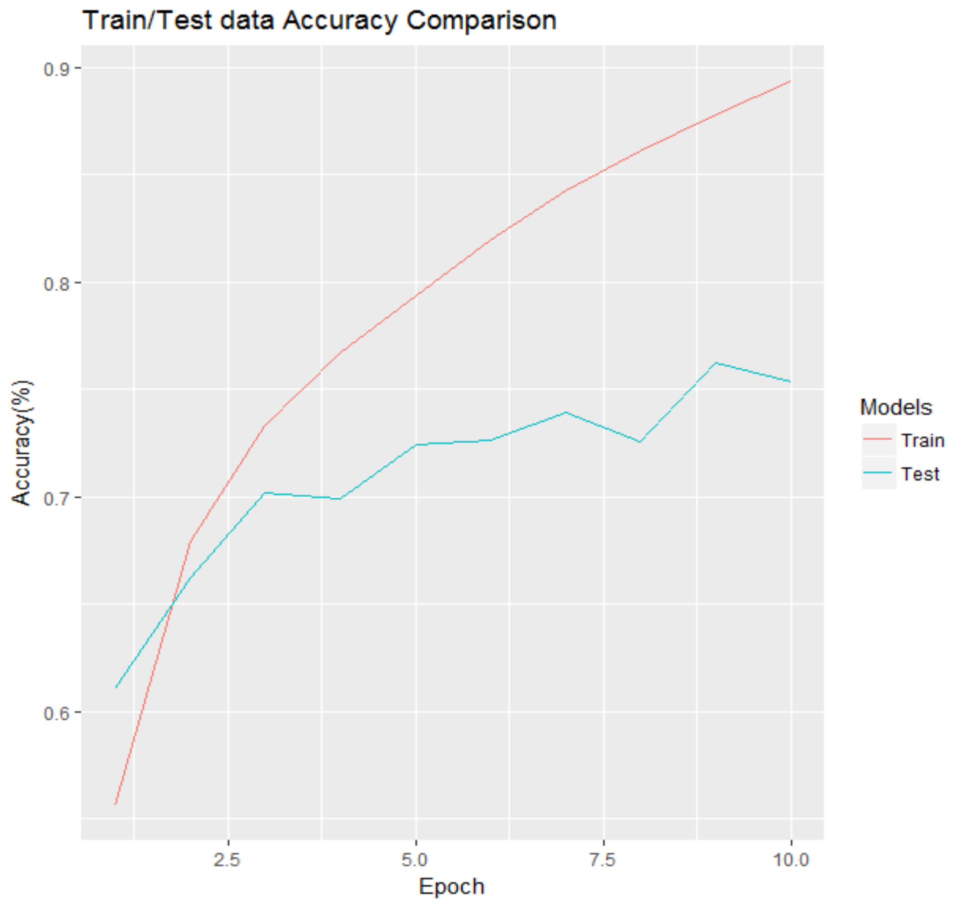



Training data Accuracy Comparison
Accuracy(%)
0.3
0.4
0.5
0.6
0.7
0.8
5
10
15
Epoch
Models
With.Augmentation
Without.Augmentation

ResNet model for comparison

Since the deeper network model is taking a lot of time with CPU, we also made some residual deeper network which is also called “ResNet”. It has a different structure than other neural network architectures. Basically, it improves the accuracy because it takes into account the weights of previous layers so that gradients don't vanish or explode. Actually, if you make a network deeper, it can hurt your ability to train the network to do well on the training set. Therefore, you do not want it to too deep. However, with ResNet, network can be deeper since it can skip connections which allows it to take the activation from one layer and suddenly feed it to another layer even much deeper in the neural network. And using that, you'll build ResNet which enables you to train very deep network without hurting the accuracy.

(attached file: resnet_1.ipynb, resnet_1.py)
(https://arxiv.org/pdf/1603.05027v2.pdf)
(https://arxiv.org/pdf/1512.03385.pdf)



Final conclusion on two models (Deep learning model and ResNet)

When compared to two models, the computation time of ResNet is much shorter. Otherwise, the accuracy on the data seems fine for the both although deeper network with pre-processed data shows the best accuracy. Meanwhile, the good feature of deeper network model with data augmentation is that it is efficient in reducing loss which shows about 80% in the first epoch.