# COL341

Assignment 3 Report

## Week 1. 10/10/2019.

We've started approaching this assignment by implementing various well-known neural network architectures and modifying them to find the best performing one.

So far we've tried

- The sequential model from assignment 2, with added dropouts after each block of layers — 35.38% accuracy.
- A modified version of the sequential model from assignment 2 — 43.93% accuracy. Modifications being
    - Increase the number of layers, i.e. make the model deeper.
    - Dropout after blocks of layers. We added more dropouts in the later parts, less dropouts in the earlier parts of the network.
    - Regularize weights after each convolution layer.
    - Increase patience of early stopping callback.
- ResNet-50 [1] with 3 layer blocks — 26.7% accuracy.
- The previous model was overfitting, so we reduced the number of layers and tried again. This time there was no overfitting, the model stopped training due to early stopping criteria — 28.56% accuracy.

In the coming weeks, we're planning to implement Res2NeXt-29 [2], optimize it as much as we can and then add squeeze-and-extraction blocks [3] to each block of that.

### Citations

- [1] https://arxiv.org/abs/1512.03385
- [2] https://arxiv.org/abs/1904.01169v2
- [3] https://arxiv.org/abs/1709.01507v4

## Week 2. 17/10/2019.

This week we tried to modify Resnet and make it better. Alongside that, we've also implemented DenseNet.

We started by the following models

- ResNet-50 [1] with added Dropouts. It didn't help improve the accuracy much — 28.3% accuracy.

- ResNet-18 [1] with added Dropouts. — 33.4% accuracy.
- DenseNet [2], with the following variants
    - With 4 layers per dense block, and growth rate k=8 — 39.8% accuracy.
    - With 6 layers per dense block, and growth rate k=8 — 38.6% accuracy.
    - Same as the last one, with added dropouts, one for every block — 39.4% accuracy.
- VGG-16 [3], a sequential model — 30.65% accuracy.

Then we got frustrated because whatever model we were testing on, none of them were giving any substantially good results. All of them were stuck below 40% accuracy. So we thought there must be something fundamentally wrong with our approach, and we started scrutinizing the whole pipeline.

That's when found out that we've been reshaping the model incorrectly. There was no clear instruction about how the image pixel values are flattened in the CSV, so we just assumed it was flattened in the standard numpy way. Thus, we've been feeding improperly structured images to the neural networks—the image pixels were not in the right format.

Once we corrected this, we started getting better results with the same models.

- ResNet-18 [1] with added Dropouts — 49.15% accuracy.
- ResNet-50 [1] with added Dropouts — 38.3% accuracy.
- DenseNet [2], the following variants thereof
    - 8 layers per dense block, growth rate k=12 — 51.92% accuracy.
    - 4 layers per dense block, growth rate k=8 — 49.8% accuracy.
    - Same as the last one, with added dropouts, one for every block — 55.4% accuracy.
    - The DenseNet models were overfitting a lot, so we increased dropouts. This time used the same model as above, but with more dropouts, one for every dense layer. This was the best performing model of this week — 62.92% accuracy.
- The modified version of the model from assignment 2, which we used in week 1 [see report of week 1 above] and was giving the best results in that week — 61.42% accuracy.

So far, the best working model has been DenseNet [2]. The model has 3 dense blocks, each block containing 4 layers. Each layer in a block takes all the outputs of the previous layers combined as input. This way, DenseNet [2] is a lot more dense than standard sequential models.

Other than that, the sequential model with 61.42% accuracy has been performing very well too. What's better is, this one wasn't overfitting at all, so with some modifications and sufficient training time, this can perform a lot better.

### Plan for the Next Week

We plan to perfect the two best performing models from this week, and also possibly integrate SENet into them.

## Citations

- [1] https://arxiv.org/abs/1512.03385
- [2] https://arxiv.org/abs/1608.06993
- [3] https://github.com/geifmany/cifar-vgg

## Week 3. 24/10/2019.

This week, we implemented two new models. and tried different variations of the models we've used before.

A new model we tried is Wide-ResNet [1]. It's essential ResNet with more number of channels per layer, thus the model being wider. We tried with different parameters, such as the width and depth of the model. Below are the different variants we've tried

- With depth of 16, and k=4, k is a parameter which determines the width of the model. This has been the best performing model this week — 64.35% accuracy.
- Same as the above with dropouts added of rate 0.3 before each convolution layer. This didn't help much — 61.2% accuracy.
- We tried increasing the width, k=8 — 64.02% accuracy.
- Then we increased the layers too, total layers being 22 — 64.33% accuracy.

The best performing model of last week was DenseNet, so we tried to improve it by changing its parameters.

- With 4 layers per dense block and k=10 — 62.82% accuracy.
- With 6 layers per dense block and k=8 — 61.18% accuracy.
- With 6 layers per dense block and k=10 — 60.40% accuracy.
- With 4 layers per dense block, k=12, and with SGD optimizer like described in the paper instead of adam — 63.98% accuracy.

Then we tried adding Squeeze-and-Extraction [2] layers in between every dense block, that didn't help much, below are the parameters

- With 4 layers per dense block and k=8 and SENet — 60.20% accuracy.

So far, Wide-ResNet [1] has been promising, and according to the official paper after a sufficient number of epochs it should give good results. We have only run it for 2 hours, so we'll see how it performs after we give it 6 hours to train itself.

Citations

- [1] https://arxiv.org/abs/1605.07146v4
- [2] https://arxiv.org/abs/1709.01507v4

## Week 4. 31/10/2019.

We didn't make much progress this week, we just ran the previous model for longer amounts of time to see if they would make a breakthrough. So we increased the patience in our older models, and ran them for 6 hours instead of 2 hours.

- The best performing model from last week, i.e. Wide-ResNet with k=4 and depth=16 — 65.55% accuracy.
- Wide-ResNet with k=8 and depth=22. This model is still running, and the validation accuracy till now is — 67.12% accuracy.

The 2nd model, which is the best performing one, is still running, so we couldn't submit the predictions file to Moodle. So we've submitted the predictions file of the first model.

## Week 6. 14/11/2019

For the final week, we tried enhancing Densenet and Wide-ResNet, getting them as close to the paper as possible, and running them for different parameters and for longer periods of time. The results were as follows.

- For DenseNet, we added a compression factor of 0.5 in each transition layer, and a bottleneck layer in each dense laye, as described in the paper [1]. Two variants of this model was tried.
    - Growth rate k=12, depth=40 — 52.02% accuracy.
    - Growth rate k=24, depth
- For Wide-ResNet, we changed the optimizer used to SGD, with learning rate=0.1, learning decay=0.0005 and nesterov momentum=0.9, as described in the paper [2]. We tried this model with different width and depth.
    - Width=16 and depth=4 — 66.23% accuracy.
    - Width=16 and depth=8 — 68.21% accuracy.
    - Width=22 and depth=8 — 69.51% accuracy.
    - Width=16 and depth=4 with added dropout 0.3 at before every convolution layer — 65.06% accuracy.

    - Width=16 and depth=8 with added dropout 0.3 at before every convolution layer — 66.23% accuracy.
    - Width=22 and depth=8 with added dropout 0.3 at before every convolution layer — 66.40% accuracy.
    - Width=22 and depth=4 with added dropout 0.3 at before every convolution layer — 60.49% accuracy.

The best performing model was Wide-ResNet, and the accuracy we could achieve is 69.51%. As we increased the width and depth, the model was increasingly giving better results. Also, adding dropouts did not help, it rather decreased the performance of the model. Also, using SGD optimizer with proper fine-tuned parameters made a big impact and increased accuracy from 65.23% to 69.51%, despite the width and depth being same.