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 [3]. 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 [3] 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