Practical 3: Transfer Learning

Contributing Students:
Barış Özakar

January 6, 2021

BASE MODEL DECISION

Here we will tell about our process of building and training our model and our decisions while doing so. Firstly, we needed to decide on a pretrained model. Before we made our final choice of Xception model, we also tried ResNet50 and InceptionV3. For all of these pretrained models trained on the "ImageNet" dataset, we took all the layers except the top layers to do the transfer learning. We froze the base model layers and added dense layers to be trained on the CIFAR 10 dataset. We have realized that ResNet50 had a long training time, possibly caused by the sheer size of the base model. Therefore, it took very long to train ResNet50. Judging by the first few epoch accuracies, we decided that Xception model is the model that we wish to use as our base model.

MODEL DESIGN

Our model consists of a Lambda layer, the Xception base model and fully connected layers (which have batch normalization and dropout layers inbetween). The Lambda layer resizes the input (32, 32) into (224, 224) which makes sure Xception model behaves optimally as it was originally trained on a (224, 224) input. The Dense layers have a converging scheme going from (256, 128, 64, 10). Without dropout and batch normalization layers, we have realized that the models that we trained were overfitting. Therefore, we added these layers. However, these layers significantly increased our training time.

TRAINING

In the CIFAR-10 dataset, we have a total of 60000 images consisting of 10 categories. In order to train and validate our model, we have separated the initial dataset into 35000, 15000

and 10000 images for training, validation and testing datasets respectively. For our training, we have appropriately used categorical cross-entropy as our loss function. In terms of optimizers, we have experimented with SGD, RMSProp and Adam optimizers, however we finally empirically decided on Adam as it converged faster with a learning rate of 0.001 for the initial training with frozen base model layers. We have tried other learning rate values however, learning rate of 0.001 had the best convergence properties. We used a batch size of 32 with 15 epochs and our training and validation set accuracies were 91.18% and 86.69% respectively.

FINE-TUNING

After the first training, we have fine-tuned the model by unfreezing the base model. However, we made sure that it kept running in inference mode, which was ensured by passing training=False when calling it. This meant that the batchnorm layers would not update their batch statistics. This prevented the batchnorm layers from undoing all the training we've done so far. For the fine-tuning training, we made sure to use a very low learning rate since we do not want the model to overfit. Therefore, we used a learning rate of 0.00001. For fine-tuning, we again used a batch size of 32 with 15 epochs and our training and validation set accuracies improved up to 98.53% and 94.40% respectively. Upon testing our fine-tuned model on the test set, we have achieved an accuracy of 94.13%, which is above the benchmark expectations of 80%.

DIFFICULTIES

Long training times and getting the accuracy over 80% was among the main difficulties that we faced. In order to solve long training times, we have decided to use Google Colab so that training was faster. Getting the accuracy over 80% took empirical observations and changes to our model and our hyperparameters (addition of dropout and batchnormalization layers, determining a learning rate that is suitable, adding the right amount of dense layers/neurons to allow for successful decision making).

REFERENCES

[1] https://keras.io/guides/transfer_learning/

[2] https://keras.io/api/applications/