CS604 – Deep Learning for Computer Vision

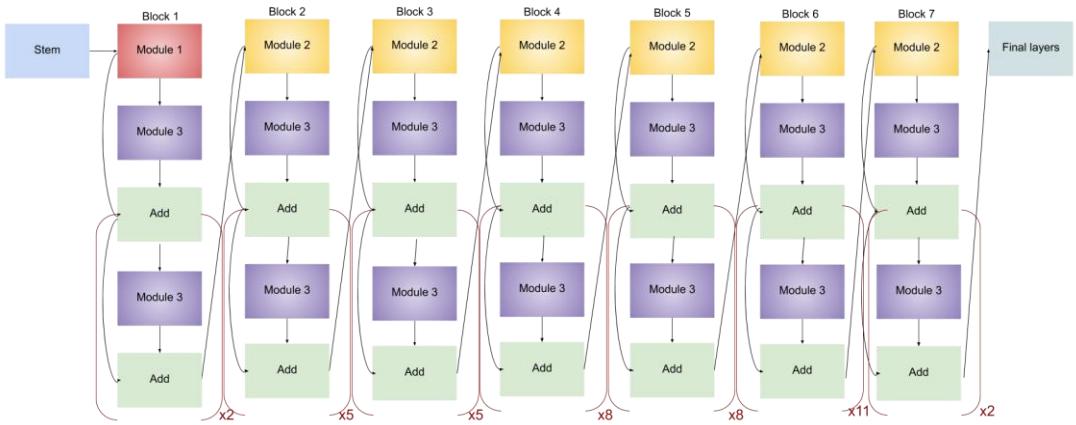
Assignment 1:

Image Classification - Report

Noah Kuntner

Description:

Over the course of the project, I have used multiple pre-trained models, such as DenseNet, ResNetV2, InceptionResNet, whereas I have been able to score the highest with EfficientNetB7. EfficientNetB7 is the largest iteration of the EfficientNet pre-trained models, which takes 66 million parameters in comparison to the smaller EfficientNetB0, which takes 5.3 million. Due to the large size, I have ended up training the model on Colab, as I had to wait more than 5+ hours for each run on my local machine, which posed to be a large bottleneck.



EfficientModelB7 – General Architecture

Structure of the model:

In my model, I have used a diverse set of inputs, as I have tried to take in diverse values for the validation split, the size of batches, and the number of epochs. I have found that in this case, a small size of the validation split does not negatively change the outcome too much, whereas I suspect this to be the case, because I am already using large pre-trained models and that given that there does exist a large imbalance between the training set and the test set, as the test set is significantly larger than the training set (around 5.6k images against 13k images), this does not affect the model strongly.

Furthermore, after creating the datasets for training and validation sets, I have implemented some short explanatory descriptive analytics showcasing the number of samples belonging to each set and followingly, printed out 9 images of the training dataset to showcase the data inputs.

Then I have created the model taking as input the shape and width of the input data and imported the EfficientNetB7 pre-trained model, using its Imagenet-weights and setting “include_top” to false, which is defined to prevent the fully-connected output layer to be loaded and this enables one to add other layers to model. Logically, after importing the EfficientNetB7 I have freezed the layers of the model and only trained the additional hidden layers.

Hereby, I have tried to implement multiple different models, such as also branching models and concatenated models, whereas I have been unsuccessful in creating a branched model that would score better than just a model with another additional layer, as this added complexity did not improve the performance significantly. Hereby, I may refer to the fact that EfficientNetB7 has already been trained on million images of the Imagenet-dataset and thus it is not surprising that trying to create a branched model on top of this model with a small dataset of around 5,600 images does not lead to a significant improvement, but only raises the loss of the model itself if the added complexity is not used in an efficient way, which is very hard to implement. Thus, I have found the greatest success in achieving a higher accuracy in a lower amount of added complexity, as my models had strong problems in overfitting, which I have seen in the lack of the robustness of the model, as my categorical accuracy metrics strongly differed themselves from those achieved on Kaggle. Hereby I may refer to cases in which I have achieved in multiple cases a categorical accuracy of 100% (95% in validation categorical accuracy) in training, but only achieved with the same models’ scores lower than 90% in my submissions on Kaggle..

For the additional hidden layers I have used a rather high dropout rate of 0.8, which I found to be more useful, as again my model was easily hitting high categorical accuracy values, but did improve much harder in the validation categorical accuracy. Furthermore, I have used ‘relu’ as the activation function of the hidden layer, which proved to be more efficient than ‘softmax’, ‘sigmoid’ or ‘tanh’. On the other hand, for the output layer, I have used the ‘softmax’-activation, which as expected scored better than the other activation function in the output layer.

I have tried out with numerous runs the differences in the compiler optimisers, whereas I have tried out RMSProp, Adam and SGD, whereas the Stochastic Gradient Descent managed to obtain me the highest scores. Hereby, I may note that I have tried to implement a learning rate scheduler within the Stochastic Gradient Descent, which unfortunately only increased my loss significantly and thus did not help me improve my model this time, which I still may need to look into. Finally, I have run my models for 10 epochs, 15 epochs, 20 epochs and at times 50 epochs, whereas I must note that I have not seen a significant improvement with running the model for more than 15 epochs at a time and only achieved issues with overfitting, while the loss and validation loss mostly stayed the same, while the time taken increased drastically.

After having trained the model I have unfreezed the last 20 layers of the model and re-compiled the model again. Hereby, I have chosen Adam as optimiser, as I have achieved higher values in validation categorical accuracy with Adam than with Stochastic Gradient Descent and RMSProp. Finally, I have then ran the model again with the unfreezed model and finally plotted the results of the loss and the categorical accuracy. My highest final score on Kaggle has been 0.93512.

Followingly, I have then predicted the values of the images of the test set and saved the results in a .csv-file. I have found this assignment very interesting, as it introduced me to the vast world of different pre-trained models for computer vision and I have found it interesting comparing the different experiences and results while comparing the different runtimes and efficiencies of the models.