Deep Learning project on image classification

Transfer Learning for pretrained models for image classification allows us to use a pre-existing model, trained on a huge dataset, for our tasks. Consequently, it reduces the cost of training new deep-learning models, and since the datasets have been vetted, we can be assured of their quality.

For this project, I decided to practice transfer learning for image classification using medical images, a binary classifier. I was curious to compare setup and performance of a few popular models which are widely-used in the industry. I expected all these models to perform very similarly and since my dataset is not very large, I expected data augmentation to improve performance of the models.

There are total of four files, the summary, models with RMSprop optimizer, models with RMSprop optimizer and data augmentation and models with Adam optimizer. I was able to run the final models through twice to check for consistency.

I found a dataset of 2000 MRI images of brains, half with stroke and half without.
https://www.kaggle.com/datasets/buraktaci/mri-stroke/data

I decided to do a kind of a grid search: to compare the performance of self-built and pre-trained models with and without data augmentation, with all layers frozen and some layers trainable, with higher and lower learning rates, with 2 different optimizers: RMSprop and Adam.

Getting the models to work, setting the parameters and picking the top for each model took a lot of time. There was a lot of back and forth with various combinations of image size, data preprocessing, getting the shapes and inputs and outputs right, whether to add Pooling, Flatten, batchNormalization or Dropout at the end.

Baseline model

First, I built a simple convolutional neural network with 5 layers with 32-64 neurons each, to have as a baseline. The model performed very poorly, output was random.

Then I improved the baseline model, I added 1 layer and many more neurons (64 – 1024 per layer). The updated model worked quite good, getting 70 – 80% accuracy for both optimizers, but with data augmentation, the validation accuracy was flat at random 50%. I was surprised, because I expected the opposite to happen.

It made me think, that my data set was too small. Flatline validation accuracy could be caused by too high learning rate as well. My learning rate for this model was 1e-5. That took me to thinking that medical images, MRI images in my case, might require different kind of augmentation.

Pretrained models

As a second step, I picked 4 widely-used pre-trained models: VGG19, ResNet50, InceptionV3 and EfficientNetB0. Even picking these four took a while, especially with all the different variants. I picked the lower end variations, since my problem is relatively simple, just a binary classifier.

1. VGG16

VGG16 was introduced in 2014 and has remained one of the most popular models. It is developed by the Visual Geometry Group at the University of Oxford, VGG models are known for their simplicity and depth. Deep networks, simple architecture using only 3×3 convolutions. I used VGG19. It has 26 layers. It's main downside is a huge number of parameters which makes it long to train. I got VGG19 to work fairly quickly. And true, for me too, this took longest to train, 2 to 6 times longer per epoch than others. First I tried out all frozen layers with 0.01 learning rate, with an extra dense layer, Dropout and GlobalAveragePooling. It got validation accuracy of around 80% but only 60% with data augmentation.
It's main downside is a huge number of parameters which makes it long to train. I got VGG19 to work fairly quickly. And true, for me too, this took longest to train, 2 to 6 times longer per epoch than others. First I tried out all frozen layers with 0.01 learning rate, with an extra dense layer, Dropout and GlobalAveragePooling. It got validation accuracy of around 80% but only 60% with data augmentation.

After that, I unfroze 3 layers and decreased the learning rate to 1e-5. Keeping everything else the same, the model got validation accuracy of 97% with RMSprop, 99% with Adam but only 79% with data augmentation.

2. ResNet50

Second model to try out is ResNet50, short for residual network. ResNet was released in 2015. The main motivation behind this image classification model is to avoid poor accuracy as the model went on to become deeper and to tackle vanishing gradient issue. Deep architectures (up to 152 layers), Residual blocks to allow gradients to flow through shortcut connections. ResNet50 has 179 layers.

I spent a long time trying to get ResNet50 to perform well and I don't think I quite succeeded. I worked with various input variations, data preprocessing and top. I ended up adding an extra dense layer, GlobalAveragePooling and BatchNormalization. The models reached over 80% validation accuracy, but the validation accuracy and loss curves are deeply jagged.

The ResNet50 with 3 unfrozen layers overfit even quicker. ResNet50 was the only model, where data augmentation gave slightly better results.

3. InceptionV3

The third model I used, was InceptionV3 by Google. Inception was also released in 2014 and it is faster to train. Inception network uses inception modules to capture multi-scale features. Inception modules with convolutional filters of multiple sizes. Efficient architecture balancing accuracy and computational cost. IceptionV3 has 315 layers.

I could not get InceptionV3 working using its own data preprocessing function, but it worked with VGG19 preprocessing function. Here too I used an extra dense layer, GlobalAveragePooling and Dropout on top. For the all frozen variation, the models reached 89% validation accuracy.

InceptionV3 with 12 unfrozen layers did not perform as well, reaching 60% accuracy.

4. **EfficientNetB0**

Finally, I used EfficientNetB0 by Google. It's innovation is Compound Scaling, that is scaling all the dimensions, depth, width, and resolution, of the images with a stable coefficient to improve the performance of the network and increase the accuracy and time complexity. Efficient and accurate. EfficientNetB0 has 243 layers.

EfficientNetB0 was especially particular about inputs, but after I resized all images to 224 by 224, it performed best of all, reaching almost perfect validation accuracy. This was by far the quickest model to train. Strangly, the validation accuracy was better than training accuracy. The best validation accuracy across all models was EfficientNetB0 with Adam optimizer, but even EfficientNetB0 with augmentation performed relatively well.

Conclusion

I was able to tune a model variant in each category to perform better than random and some to high degree of validation accuracy. Some models did, probably, not reach their full potential, leaving the model development for further study as well as research and experimentation about data augmentation specifically for medical images.