Clustering the "Grapevine Leaves Image" Dataset

Parsa Tasbihgou
Prof. Sajedi
August 6, 2022

In this project we wanted to build a model to effectively classify a dataset consisting of five species of grapevine leaves.

I have used Google colab environment to develop my model for its capabilities to accelerate model training using GPUs and also providing cloud storage to store datasets and also models and other byproducts.

I experimented three approaches to find the most suitable method for this problem; The first two were unfortunately not very successful. I will discuss them briefly blow, but more detailed explanations will be given on the third method.

1. Creating a custom sequential model with randomly initialized weights: I created layered sequential convolutional neural network in Keras, with 4 convolution layers (and corresponding pooling layers) and 3 dense layers. However this model struggled to reach accuracies greater than 40%.
2. Secondly I experimented with ResNet network. Though this was a huge improvement compared to my previous attempt, this method still appeared to be not the best candidate for further development, since the best results generated by this architecture hardly surpassed 88%.
3. The third method I experimented with was the EfficientNet architecture that could easily achieve accuracies above 90%, and with more epochs more that that. Hence I continued tuning this architecture for this specific problem.

First the dataset is loaded from the mounted drive. The class name is present in each image's filename so class names can easily be extracted. After that the loaded data is split into three subsets, Training, Validation and Testing.

I tried to expand my training data set by data augmentation, however this proved to not only not improving the results but in mostly it lowered the models accuracy. I think one of the reasons that data augmentation did not help is that images in this dataset are extremely similar so any perturbation to the training

data, introduces unnecessary inaccuracies and prevents the model to converge properly. As this was confirmed by experimenting with an autoencoder. We will discuss the autoencoder more later.

I found an EfficientNet3 model on the internet with relatively good tuned parameters and extra layers for added accuracy. This network uses the ImageNet weights as initial weights. Not using random weights appears to be the single most important improvement to any network, since training a network with more than 3 million trainable parameters from scratch is extremely difficult and time consuming.

I changed the parameters a little to get the best possible result, however too much change to the parameters proved to quickly lower the accuracy and convergence speed.

One phenomena that was unexpected, was that by not using augmented training data, both the training accuracy and validation accuracy improved greatly, and I thought this could be a sign of the model overfitting on the training set, however after evaluating the test data on the model I observed that test data accuracy had also improved considerably, that is one of the reasons that I believe most perturbation to the images possibly lower the models accuracy.

With the slightly adjusted EfficientNet3 network I was able to achieve 96% accuracy that is very close to the 97% achieved by the original article.

I trained an autoencoder network with 2 convolutional and pooling layers and two dense layers, I also tested {10, 15, 30, 40, 50} as number of nodes in the feature space (bottleneck width) . Increasing the feature dimension seemed to improve reconstruction accuracy, however an encoder with 50 output nodes preformed less than an encoder with the same architecture but with 40 output nodes.

I used an autoencoder network to reduce the dimensionality of input data for the neural network, but this proved to lower the accuracy. So instead I added the output of the encoder as a feature vector to my neural network (similar to what is used in the original article, but not the same). The network that I used to obtain 96% accuracy used a dense layer after the EfficientNet3 convolutional network so it was easy to add the new feature vector. I named the new network Advanced EfficientNet (AEN). AEN is a non sequential neural net, so it is implemented as a functional model in Keras.

Note that the autoencoder part of AEN does not add any new trainable parameters to the network except the weight of the connections from the encoder output to the following dense layer. Therefor it can be trained beforehand and the pertained weights are just plugged in.

Among the methods that I tried using an autoencoder concept, AEN achieves the best result by far, as no other method was able to pass 80%. However the use of an autoencoder doesn't add any real value to this problem, but I believe in other applications this method could be effective.

Finally the results including the confusion matrix, cross-validation results and accuracy and loss can be found in the note book (linked bellow) and the weights are stored in the mounted google drive. Finally the best result I obtained was 96% accuracy with 0.93 loss.

Note Book:https://colab.research.google.com/drive/1OKisYz1P_8x7z7uBzL09Vhh1sSS6ily-?usp=sharing