Project Phase 2b, Project Competition Participation
Ryan Smith
11/21/24

(a) Task

Name and short description of what the task is.

Bacterial Morphology Classification, by Yola

This task aims to classify bacteria images into three classes: cocci, bacilli, and spirilla. Participants will build and validate their models using labeled training and validation sets, and then predict classes for an unlabeled test set. Also, this is an image classification task. The other two project competitions I am working on are mainly text based, both related to Reddit posts in csv files.

(b) Approaches

Describe how you approached the task. Include any preprocessing steps you needed to complete to prepare the data, how you evaluated your approaches using your own, which kinds of models you tried, how you selected hyperparameters, and so on. This should be the longest section and you may include any results/discussion from your various model runs here to demonstrate the effort that you put forth to arrive at a high-quality model for the task.

The General Approach

The general approach is of picking two main models, then hyperparameter tuning them. There won't be a real goal of the number of tunings, but it will be at least probably five tuning attempts per model. I will have the standard metrics printout in the python notebook code and later in this document. I will at the end pick the best model and tuning and print that out to the preds.txt file for submission to the competition.

Preprocessing

How the Approach was Evaluated Based on My Own

I looked at this similar to how I looked at my project. The workflow was essentially:

- Dataset Loading
- Preprocessing (basic preprocessing)
- Feature Engineering (none needed with this)
- Model 1 Building
- Model 1 Training
- Model 1 Validation
    - Metrics
    - Hyperparameter Tuning
    - Metrics on the Tuned Model
    - Comparison of the Models
- Model 1 Testing
    - Metrics on the Testing Model
- Conclusion on Model 1

Then repeat this process for a second model, and a conclusion on the models and hyperparameter tunings.

This seems to be a fairly standard workflow that makes sense and makes a project go smoother.

The Models

We are going to use two models that are good for image-based datasets: CNN and Transfer Learning with Pre-trained Models. The Transfer Learning one is pre-trained and needs fine-tuning. This is the first time working with the Transfer Learning model, and before this I knew very little about it. This was a good learning experience with both the Transfer Learning model and the CNN, which I did have some experience with, but not much.

The Transfer Learning is apparently mainly trained on basics of images, such as edges, corners, distinguishable features, etc. Then we fine tune it with our classes that we add to the layers, replacing the build-in ones. This is a popular model in the Keras library of TensorFlow, which is the one used.

I tuned the hyperparameters over five times including the GridSearchCV settings search option, looking for the best settings.

CNN (Keras Sequential)

Convolutional Neural Networks are good for image-based projects and is why it was chosen for this. These are based on 4 or five main layers, from input, to convolutional, to pooling, to flattening, then to a dense layer. The input layer takes in the image, the pooling convolutional layer does some basic processing including with filers (kernels) that scan the image. The pooling layer works with a feature map, that was created in the previous layer, and reduces the spatial

dimensions. Flattening is of the data to a 1D vector, then a final dense layer that connects all the neurons.

A somewhat unique aspect of CNNs are their use of the validation set directly with the training, to prevent overfitting and make use of immediate feedback on the training.

Another thing we will do is save our preprocessed images, of mainly resizing to 128x128 and normalizing to 0-1 values, to file with NumPy arrays. This makes it easier to hyperparameter tune and not have to go through the preprocessing each time.

Hyperparameters

Hyperparameter	**Usage**	**Impact**
batch_size	It is in flow_from_directory to define the number of images processed per batch.	Affects memory usage, speed of training, and gradient update noise.
epochs	It is in model.fit to define the number of complete passes through the training dataset.	Too many equals potentially overfitting, and too few can equal underfitting.
learning_rate	It is in Adam(learning_rate=…) to control how much weights are adjusted during training.	Balances convergence speed and stability. Affects the optimization process directly.

The batch_size is probably the least of an effect, with epochs and learning_rate potentially causing considerably changes in the results. In my view, these seemed like a decent combination of hyperparameters to tune in our task.

Default Settings

With the default settings, we received the metrics of:

Validation loss: 2.7656784057617188
Validation accuracy: 0.3645833432674408
Total test images being processed: 120

Tuning, 150x150 image sizes

Validation loss: 1.098673701286316
Validation accuracy: 0.34375
Total test images being processed: 120

Tuning, 224x224 image sizes, data augmentation

Data Augmentation actually performed well here with image sizes of 224x224. It was performing poorly with 150x150 image sizes.

Validation loss: 1.0825647115707397
Validation accuracy: 0.4375
Total test images being processed: 120

Tuning, Epochs of 8 and modified learning_rate 0.0001, imag size 256x256

Validation loss: 1.430116057395935
Validation accuracy: 0.3541666567325592
Total test images being processed: 120

Tuning, 256x256 image size, 12 Epochs, learning rate 0.001

Validation loss: 1.239443302154541
Validation accuracy: 0.4583333432674408
Total test images being processed: 120

Tuning GridSearchCV (Keras Model version uses Keras Tuner)

This is not actually GridSearchCV, but the Keras version using Keras Tuner

Validation loss: 39.39779281616211
Validation accuracy: 0.34166666865348816
Total test images being processed: 120

This result is surprisingly low for the search method.

Image Size Tuning

With 11 Epochs and an image size for all images of 224x224 we get:

Validation loss: 1.0825647115707397
Validation accuracy: 0.4375
Total test images being processed: 120

With 11 Epochs and an image size for all images of 150x150 we get:

Validation loss: 1.098673701286316
Validation accuracy: 0.34375
Total test images being processed: 120

Better results with 224x224, as would be expected since the algorithms are working on more data. Because the data sets are not that big, the computational trade-off is not much, so we can easily compute the 224x224 which is a standard for image processing. 150x150 is also often used.

Data Augmentation Tuning

I tried some data augmentation of shifting images 20% vertically, and horizontally, along with flipping them, and scaling them. These all resulted in lower accuracy. 224x224 seemed to do ok with data augmentation though.

Transfer Learning

We are using MobileNetV2 from Keras for the Transfer Learning implementation. We also downloaded weights from google related to this. The download file is only about 9 megabytes and is in the dataset directory of files needed to be downloaded to run the code. As I mention in the python notebook in the beginning, the datasets and files need to be downloaded from GitHub, as the paths are not url paths to the GitHub files. I thought it would be better for customization purposes to have a local copy that could be modified by the user if wanted. The downloads for all three CodaLabs competitions are not that big also, I think about 50 or so megabyte total for all three. It is mainly just the image set, which is this one. The other two are reddit posts related.

MobileNetV2 performed considerably better also. I go over that below with various tunings, but in general it performs very well compared to CNN Sequential. From my research, the main reason is the pre-trained weights.

Hyperparameters

We will work with the same hyperparameters mainly, of learning_rate, batch_size, and epochs. We will also work with modifying some preprocessor concepts of image size and potentially scaling.

Default Settings

Tuning, 32/8/.0001 for the three hyperapameters

This gave great results compared to the CNN model. .70 accuracy.

Validation loss: 0.7374851107597351
Validation accuracy: 0.7083333134651184
Total test images being processed: 120

Tuning, 16/12/.001 for the three hyperparameters

This is also 256x256. **This is the winner of both models and tunings**. Below we have a similar tuning with 32 epochs though, which did not fair as well. The 16 epochs with 256x256 might be the main factors here.

Validation loss: 0.6899534463882446
Validation accuracy: 0.7857142686843872
Total test images being processed: 120

Tuning, 150x150 image size

Validation loss: 0.9839985966682434
Validation accuracy: 0.7053571343421936
Total test images being processed: 120

Loss here is different than previous ones. Accuracy is similar. I am researching how the loss difference can be so much while the accuracy is similar, which you would not expect. It is expected the loss would be higher. Here is my analysis currently: The model may start with a higher loss, which decreases as it learns, while accuracy stabilizes relatively quickly. Overfitting can also contribute to higher loss despite stable accuracy.

Tuning, 224x224 image size

This is apparently a good image size for the MobileNetV2 model.

Validation loss: 0.7883021235466003
Validation accuracy: 0.7410714030265808
Total test images being processed: 120

Results here are good. On the higher end.

Tuning, 256x256 image size, 32/12/.001

Validation loss: 0.840329647064209
Validation accuracy: 0.6875
Total test images being processed: 120

Summary

We tested two models, CNN and Transfer Learning. The CNN was with Keras Sequential, and the other with MobileNetV2 from Keras as well. We tried approximately 15 tunings each. I don't have every single tuning documented, as I did adjust some of them and just tested them, with documenting the better results, which is approximately 5 per model listed prior in this document, and in the python notebook code. I think model 1 CNN has about 8 full code tunings, and the other 5 or so in the python notebook. I learned much from this. This was the first time I worked with Transfer Learning, and I had minimal experience with CNN before this.

(c) Best Results

Which approach worked best? Why do you think that is?

Tuning, 16/12/.001 for the three hyperparameters

This is also 256x256. **This is the winner of both models and tunings**. Below we have a similar tuning with 32 epochs though, which did not fair as well. The 16 epochs with 256x256 might be the main factors here.

Validation loss: 0.6899534463882446
Validation accuracy: 0.7857142686843872
Total test images being processed: 120

The main thing is the pre-trained model for the Transfer Learning, I believe. This is from ImageNet and from over a million images it was trained on. This pre-trained model is in the dataset directory for download for use in this, or you could link by url to it.

My main analysis of the winning model:

The combination of a batch size of 16, 12 epochs, a learning rate of 0.001, and an image size of 256x256 pixels has proven to be the most effective configuration for this transfer learning model using MobileNetV2. This configuration allows the model to leverage detailed image features, achieve stable and accurate gradient updates, and avoid overfitting, resulting in superior validation loss and accuracy compared to other tested configurations.