Image Classification Using TensorFlow

Mezenner Abdelhak

05/12/2024

1 Data Pre-Processing

2 Model Structure

```
regularization_value = 0.0001
model = Sequential([
        Conv2D(16,(3,3),
                strides=1,
                #kernel_initializer='he_normal',
                activation='relu',
                kernel_regularizer=l2(regularization_value),
                input_shape = input_shape
                ),
        MaxPooling2D(),
        Conv2D(32,(3,3),
                strides=1,
                #kernel_initializer='he_normal',
                activation='relu',
                kernel_regularizer=l2(regularization_value)
                ),
        MaxPooling2D(),
        Conv2D(16,(3,3),
                strides=1,
                #kernel_initializer='he_normal',
                activation='relu',
                kernel_regularizer=l2(regularization_value)
                ),
        MaxPooling2D(),
        Flatten(),
        Dense(64,
                #kernel_initializer='he_normal',
                activation='relu',
                kernel_regularizer=l2(regularization_value)
                ),
        Dropout(0.4),
        Dense(32,
                #kernel_initializer='he_normal',
                activation='relu',
                kernel_regularizer=l2(regularization_value)
                ),
        Dropout(0.2),
        Dense(5,
                activation='softmax',
                #kernel_initializer='glorot_uniform'
                )
])
```

1. Sequential: its the structure where we stack our tensors layers. we could use **.sum-**

mary() method to display the sequential content.

2. Conv2D: convolutional layer that uses convolutional kernet which is a filter matrix that extract the features from an image, resulting in a new tensor of outputs or called also feature map that goes down to the next layer as input.

3. MaxPooling2D: a pooling operation that selects the maximum element from the region of the feature map covered by the filter. the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

4. Flatten: When applied to a multi-dimensional tensor output from a convolutional or pooling layer, the flatten layer simply collapses all dimensions except the batch dimension, resulting in a one dimensional array. For example, if the output tensor has dimensions (batch_size, height, width, channels), the flatten layer would reshape it to (batch_size, height * width * channels).

5. Dense : The regular neural network layer that contains multiple neurons, each **N+1** Neuron receives input from all the **N** Neurons.

6. Droupout: Its a layer that helps to prevent overfitting by randomly nullifying outputs, meaning it sets random outputs from random neurons to 0 during the training process, controlled by the **rate** parameter which is a fraction of the input units to drop.

3 Hyper-Parameters Optimization

Here we want to search for the best hyperparameters that our model uses to fit the training data which helps to maximize the accuracy and minimize the loss.
hyper-Parameters list here is :

```
batch_sizes = [16,32]
learning_rates =[0.1,0.01,0.05,0.001]
Epochs = 100
```

Batch Size	**Learning Rate**	**Val Acc**	**Val Loss**
16	0.1	0.2888	1.6090
16	0.01	0.1530	1.6114
16	0.05	0.1207	1.6108
16	**0.001**	**0.9138**	**0.3091**
32	0.1	0.1961	1.6156
32	0.01	0.1509	1.6115
32	0.05	0.1832	1.6113
32	**0.001**	**0.9159**	**0.2578**

Table 1: Table of First Print Results

Conclusion: we conclude that we should use learning rates around 0.001 to tune our model, so we will fit our model again but with different **hyper-parametes** this time.

Second Print Results:

```
learning_rates =[0.005,0.001,0.0005,0.0001]
```

Batch Size	Learning Rate	Val Acc	Val Loss
16	0.005	0.1767	1.6099
16	0.001	0.9224	0.2586
16	**0.0005**	**0.9159**	**0.2519**
16	0.0001	0.8750	0.3982
32	0.005	0.1767	1.6106
32	0.001	0.9181	0.2756
32	**0.0005**	**0.9159**	**0.2497**
32	0.0001	0.8427	0.4740

Table 2: Table of Second Print Results

Conclusion: we conclude from this results that we should use learning rates between 0.0005 ¡ lr ¡ 0.001

3.1 Fetching The Best Hyper-Parameters

```
best_result = min(results1, key=lambda x: x['val_loss'])

best_result_hist = best_result['history']

print("\nBest parameters:")
print(f"Batch Size: {best_result['batch_size']}")
print(f"Learning Rate: {best_result['learning_rate']:.4f}")
print(f"Validation Accuracy: {best_result['val_accuracy']:.4f}")
print(f"Validation Loss: {best_result['val_loss']:.4f}")
print(f"Accuracy: {best_result_hist['accuracy'][-1]:.4f}")
print(f"Loss: {best_result_hist['loss'][-1]:.4f}")
```

First Print Results: From the first run : Best parameters: Batch Size: 32 Learning Rate: 0.0010 Validation Accuracy: 0.9159 Validation Loss: 0.2578 Accuracy: 0.8912 Loss: 0.3102 **Second Print Results:** The second run after changing the Learning rates list : Best parameters: Batch Size: 32 Learning Rate: 0.0005 Validation Accuracy: 0.9159 Validation Loss: 0.2497 Accuracy: 0.9009 Loss: 0.2634

3.2 Hyper-Parameters Tuning Final Results

The final run after changing the learning rates to be between 0.0005 ¡ lr ¡ 0.0010 and fixing the batch size to be 32. Best parameters: Batch Size: 32 Learning Rate: 0.0005 Validation Accuracy: 0.9397 Validation Loss: 0.1974 Accuracy: 0.8877 Loss: 0.2755

4 Model Fitting With Best Hyper-Parameters

After we had the best **hyper-parameters** (learning rate: 0.0005, batch size: 32), we fitted a new model instance that we call `final_model1` to our dataset divided into batches of 32.

4.1 Evaluation on Test Set

```
final_test_batches = prepare_dataset(test_dataset, best_result['
    batch_size'],is_training=False)

test_loss, test_accuracy = final_model1.evaluate(
    final_test_batches)
print(f"\nFinal Test Accuracy: {test_accuracy:.4f},Final Test
    Loss: {test_loss:.4f}")
```

Print Results: Final Test Accuracy: 0.9246,Final Test Loss: 0.2040

5 Model Saving

Now finally we could save our model so its usable in the future for further optimization Or predictions.

```
keras_model_name = day_save_model_number+'_'+saved_date+'_'+
    test_accuracy+'.keras'
final_model1.save(os.path.join('models',keras_model_name))
```

The naming convention used here is:

1. `model_number` is the number of our model, in this case its 1 cause we generated only one final model.
2. `saved_date` its the date of the saving, in this case its 2024_12_05.
3. `test_accuracy` its our final test accuracy, in this case its 0.924