rueegg_wissiak_optimized

March 3, 2023

1 Optimized Model

This model does not show under- or overfitting and performs well on both, training and testing data. Afterwards, a brief description on how to tackle the challenges of an optimal model complexity.

To address underfitting, one approach is to increase the complexity of the model by adding more layers or increasing the number of filters in each layer. To address overfitting, we can try several approaches. One approach is to simplify the model by removing some layers or decreasing the number of filters in each layer. Another approach is to use less epochs for example.

Adding dropout or weight decay can help to address both of the above mentioned issues. We can also try adjusting the hyperparameters such as learning rate, batch size, or number of epochs.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
%run rueegg_wissiak_data_augmentation.ipynb
```

```
Found 2666 images belonging to 5 classes.
Found 2499 images belonging to 5 classes.
```

1.1 Building the Model

Here we use the same model as the overfitting one but we add some extras to reduce the overfitting behavior.

Regularization: reduce the impact impact of the weights. The weights then end up having less impact on the loss function which determines the error between the actual label and predicted label. This reduces complexity of the model and therefore reduces overfitting. We are adding regularization only to those layers which have the largest number of parameters according to the model summary.

Dropout Layers: The benefit of using dropout is no node in the network will be assigned with high parameter values, as a a result the parameter values will be dispersed and the output of the current layer will not depend on a single node. E.g. Dropout(0.2) drops the input layers at a probability of 0.2.

To **improve generalization** of the model, data augmentation is a useful tool. With data augmentation we can add artificial effects to the images such as shearing, stretching, flipping, rotating and translating. Through these effects, the images always appear differently each time they appear in the training step and therefore the CNN doesn't adapt to the exact images but rather learns about the relative features inside of an image.

[3]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout
from tensorflow.keras.regularizers import l2
```

```
model = Sequential()

model.add(Conv2D(32, (3,3), input_shape= (img_size,img_size,3), activation =␣
 ↪'relu', padding = 'same')) #padding = same size output
model.add(MaxPooling2D())

model.add(Conv2D(64, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D())

model.add(Conv2D(128, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D())

model.add(Conv2D(256, (3,3), activation = 'relu', padding = 'same'))
model.add(MaxPooling2D())

model.add(Conv2D(512, (3,3), activation = 'relu', padding = 'same',␣
 ↪kernel_regularizer=l2(l=0.0001)))
model.add(MaxPooling2D())
model.add(Dropout(0.05))

model.add(Conv2D(1024, (3,3), activation = 'relu', padding = 'same',␣
 ↪kernel_regularizer=l2(l=0.001)))
model.add(MaxPooling2D())
model.add(Dropout(0.15))

model.add(Conv2D(512, (3,3), activation = 'relu', padding = 'same',␣
 ↪kernel_regularizer=l2(l=0.001)))
model.add(MaxPooling2D())
model.add(Dropout(0.15))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dense(5, activation = 'softmax'))

model.compile(optimizer = 'adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

```
2023-03-03 09:09:26.330336: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

[4]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 32)      896

 max_pooling2d (MaxPooling2D  (None, 75, 75, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 75, 75, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 37, 37, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 37, 37, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 18, 18, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 18, 18, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 9, 9, 256)        0
 2D)

 conv2d_4 (Conv2D)           (None, 9, 9, 512)         1180160

 max_pooling2d_4 (MaxPooling  (None, 4, 4, 512)        0
 2D)

 dropout (Dropout)           (None, 4, 4, 512)         0

 conv2d_5 (Conv2D)           (None, 4, 4, 1024)        4719616

 max_pooling2d_5 (MaxPooling  (None, 2, 2, 1024)       0
 2D)

 dropout_1 (Dropout)         (None, 2, 2, 1024)        0

 conv2d_6 (Conv2D)           (None, 2, 2, 512)         4719104

 max_pooling2d_6 (MaxPooling  (None, 1, 1, 512)        0
 2D)

 dropout_2 (Dropout)         (None, 1, 1, 512)         0

 flatten (Flatten)           (None, 512)               0

 dense (Dense)               (None, 256)               131328
```

```
 dense_1 (Dense)              (None, 5)                 1285

=================================================================
Total params: 11,139,909
Trainable params: 11,139,909
Non-trainable params: 0
_________________________________________________________________
```

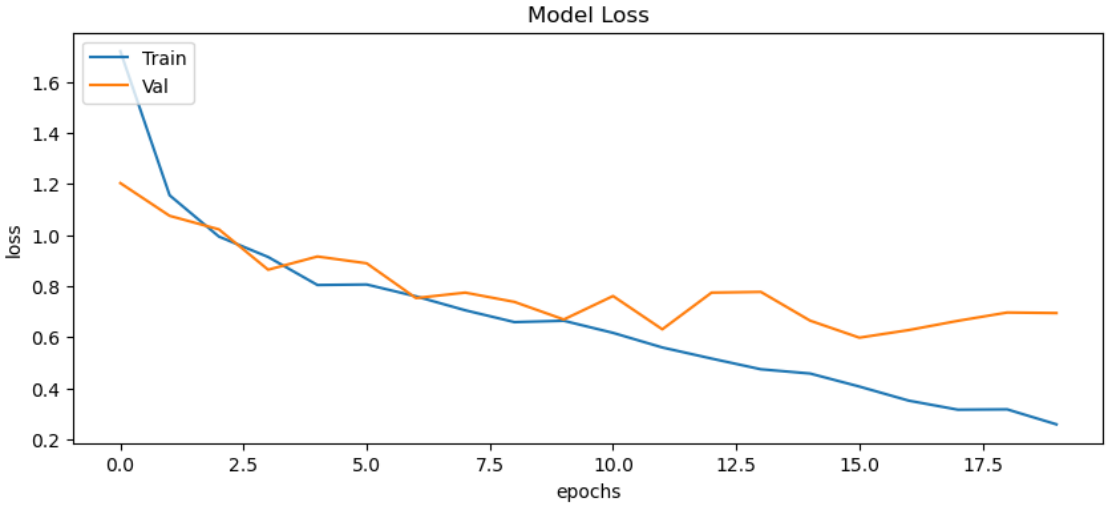
1.2 Training the Model

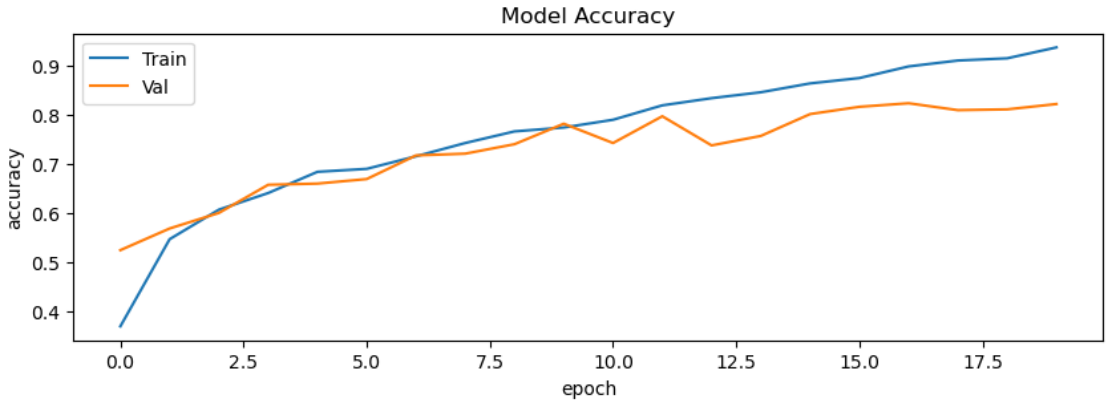
[5]:
```
history = model.fit(train_generator, validation_data=test_generator, epochs=20)
```

```
Epoch 1/20
84/84 [==============================] - 102s 1s/step - loss: 1.7190 - accuracy:
0.3698 - val_loss: 1.2028 - val_accuracy: 0.5246
Epoch 2/20
84/84 [==============================] - 101s 1s/step - loss: 1.1551 - accuracy:
0.5469 - val_loss: 1.0747 - val_accuracy: 0.5686
Epoch 3/20
84/84 [==============================] - 102s 1s/step - loss: 0.9935 - accuracy:
0.6069 - val_loss: 1.0217 - val_accuracy: 0.6002
Epoch 4/20
84/84 [==============================] - 102s 1s/step - loss: 0.9135 - accuracy:
0.6407 - val_loss: 0.8638 - val_accuracy: 0.6575
Epoch 5/20
84/84 [==============================] - 102s 1s/step - loss: 0.8039 - accuracy:
0.6838 - val_loss: 0.9155 - val_accuracy: 0.6599
Epoch 6/20
84/84 [==============================] - 104s 1s/step - loss: 0.8059 - accuracy:
0.6898 - val_loss: 0.8888 - val_accuracy: 0.6691
Epoch 7/20
84/84 [==============================] - 101s 1s/step - loss: 0.7594 - accuracy:
0.7153 - val_loss: 0.7527 - val_accuracy: 0.7171
Epoch 8/20
84/84 [==============================] - 101s 1s/step - loss: 0.7050 - accuracy:
0.7423 - val_loss: 0.7741 - val_accuracy: 0.7207
Epoch 9/20
84/84 [==============================] - 102s 1s/step - loss: 0.6586 - accuracy:
0.7659 - val_loss: 0.7377 - val_accuracy: 0.7399
Epoch 10/20
84/84 [==============================] - 102s 1s/step - loss: 0.6639 - accuracy:
0.7738 - val_loss: 0.6689 - val_accuracy: 0.7815
Epoch 11/20
84/84 [==============================] - 101s 1s/step - loss: 0.6166 - accuracy:
0.7896 - val_loss: 0.7607 - val_accuracy: 0.7423
Epoch 12/20
84/84 [==============================] - 101s 1s/step - loss: 0.5594 - accuracy:
```

```
0.8188 - val_loss: 0.6305 - val_accuracy: 0.7967
Epoch 13/20
84/84 [==============================] - 101s 1s/step - loss: 0.5159 - accuracy:
0.8335 - val_loss: 0.7741 - val_accuracy: 0.7375
Epoch 14/20
84/84 [==============================] - 102s 1s/step - loss: 0.4740 - accuracy:
0.8455 - val_loss: 0.7769 - val_accuracy: 0.7567
Epoch 15/20
84/84 [==============================] - 102s 1s/step - loss: 0.4574 - accuracy:
0.8635 - val_loss: 0.6644 - val_accuracy: 0.8011
Epoch 16/20
84/84 [==============================] - 102s 1s/step - loss: 0.4064 - accuracy:
0.8743 - val_loss: 0.5977 - val_accuracy: 0.8159
Epoch 17/20
84/84 [==============================] - 102s 1s/step - loss: 0.3513 - accuracy:
0.8980 - val_loss: 0.6275 - val_accuracy: 0.8231
Epoch 18/20
84/84 [==============================] - 102s 1s/step - loss: 0.3158 - accuracy:
0.9100 - val_loss: 0.6637 - val_accuracy: 0.8091
Epoch 19/20
84/84 [==============================] - 101s 1s/step - loss: 0.3171 - accuracy:
0.9145 - val_loss: 0.6962 - val_accuracy: 0.8107
Epoch 20/20
84/84 [==============================] - 101s 1s/step - loss: 0.2584 - accuracy:
0.9366 - val_loss: 0.6941 - val_accuracy: 0.8215
```

[6]:
```
%run rueegg_wissiak_model_visualization.ipynb
```





[7]:
```
%run rueegg_wissiak_model_evaluation.ipynb
```

```
79/79 [==============================] - 13s 169ms/step
Predicted classes: [4 4 4 … 4 1 0]
True labels: [0 0 0 … 4 4 4]
Accuracy:
 0.2108843537414966
```