rueegg_wissiak_optimized

March 6, 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.

[64]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[65]:
```
from tensorflow.keras.utils import image_dataset_from_directory

img_size = 150
batch_size = 64
seed = 31

train_ds = image_dataset_from_directory(
    './dataset/seg_train/seg_train',
    validation_split=0.2,
    subset="training",
    labels="inferred",
    seed=seed,
    image_size=(img_size, img_size),
    batch_size=batch_size
)
val_ds = image_dataset_from_directory(
    './dataset/seg_train/seg_train',
    validation_split=0.2,
    subset="validation",
    labels="inferred",
    seed=seed,
    image_size=(img_size, img_size),
    batch_size=batch_size
```

```
)
test_ds = image_dataset_from_directory(
    './dataset/seg_test/seg_test',
    labels="inferred",
    seed=seed,
    image_size=(img_size, img_size),
    batch_size=batch_size
)
```

```
Found 2666 files belonging to 5 classes.
Using 2133 files for training.
Found 2666 files belonging to 5 classes.
Using 533 files for validation.
Found 2499 files belonging to 5 classes.
```

[66]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import RandomRotation, RandomZoom, Rescaling,␣
 ↪RandomFlip

data_augmentation = Sequential(
  [
    RandomFlip("horizontal", input_shape=(img_size, img_size, 3)),
    RandomRotation(0.1),
    RandomZoom(0.1),
  ]
)
```

[67]:
```
plt.figure(figsize=(5, 5))
for image, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(image)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```

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.

1.1.1 Regularization

Regularization is used to 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. We are using L2 (Ridge) regularization since it predetermined from the task. We are adding L2 mainly to the layers that add the most parameters to the CNN.

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.

1.1.2 Generalization

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.

1.1.3 Optimizer

For the optimized model we chose Adam over the competitors because it is the most common among SGD. We tried out SGD but it performed very poorly compared to Adam which might be due to insufficient configuration of the learning rate. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order (mean) and second-order (uncentered variance) moments. Its default implementation already provides a form of annealed learning, beta_1=0.9 for the first-order moment and beta_2=0.999 for the second-order moment.

1.1.4 Activation Function

The following article states that ReLU is the overall the best suited activation function so based on this we decided to use ReLU for our optimized model.

1.1.5 Batch Size

The batch size defines how many samples (images here) run through the Neural Network before the weights get adapted. It is recommended to use mini batches to update the Neural Network multiple times during an epoch. We've tried out differnt batch sizes with the same seed on the image generator TODO

[68]:
```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout
from tensorflow.keras.regularizers import l2
```

[69]:
```
def create_model(l2_param=0.001):
    model = Sequential()

    model.add(data_augmentation)
    model.add(Rescaling(1./255))

    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=l2_param)))
```

```
    model.add(MaxPooling2D())
    model.add(Dropout(0.1))

    model.add(Conv2D(1024, (3,3), activation = 'relu', padding = 'same',␣
↪kernel_regularizer=l2(l=l2_param)))
    model.add(MaxPooling2D())
    model.add(Dropout(0.15))

    model.add(Conv2D(512, (3,3), activation = 'relu', padding = 'same',␣
↪kernel_regularizer=l2(l=l2_param)))
    model.add(MaxPooling2D())

    model.add(Dropout(0.3))
    model.add(Flatten())

    model.add(Dense(256, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(5, activation = 'softmax'))

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

    return model
```

1.2 Cross Validation for L2 Parameter

To find the optimal L2 regularization parameter we are using GridSearchCV from sklearn by applying a k=5 Cross Validation. As the to-be-optimized score we use, as per default, the accuracy. To be able to use GridSearchCV we must wrap the model to be compatible with the sklearn ecosystem.

[70]:
```
import itertools
param_grid=dict(
    l2=[0.01, 0.001, 0.0001],
    batch_size=[16, 32, 128],
    x3=[1, 2, 3]
)

params = list(param_grid.get(x) for x in list(param_grid.keys()))
param_permutations = list(itertools.product(*params))
print(param_permutations)
for perm in param_permutations:
    print(perm)
    for k in perm:
        print(k)
```

```
[(0.01, 16, 1), (0.01, 16, 2), (0.01, 16, 3), (0.01, 32, 1), (0.01, 32, 2),
(0.01, 32, 3), (0.01, 128, 1), (0.01, 128, 2), (0.01, 128, 3), (0.001, 16, 1),
```

```
(0.001, 16, 2), (0.001, 16, 3), (0.001, 32, 1), (0.001, 32, 2), (0.001, 32, 3),
(0.001, 128, 1), (0.001, 128, 2), (0.001, 128, 3), (0.0001, 16, 1), (0.0001, 16,
2), (0.0001, 16, 3), (0.0001, 32, 1), (0.0001, 32, 2), (0.0001, 32, 3), (0.0001,
128, 1), (0.0001, 128, 2), (0.0001, 128, 3)]
(0.01, 16, 1)
0.01
16
1
(0.01, 16, 2)
0.01
16
2
(0.01, 16, 3)
0.01
16
3
(0.01, 32, 1)
0.01
32
1
(0.01, 32, 2)
0.01
32
2
(0.01, 32, 3)
0.01
32
3
(0.01, 128, 1)
0.01
128
1
(0.01, 128, 2)
0.01
128
2
(0.01, 128, 3)
0.01
128
3
(0.001, 16, 1)
0.001
16
1
(0.001, 16, 2)
0.001
16
2
```

```
(0.001, 16, 3)
0.001
16
3
(0.001, 32, 1)
0.001
32
1
(0.001, 32, 2)
0.001
32
2
(0.001, 32, 3)
0.001
32
3
(0.001, 128, 1)
0.001
128
1
(0.001, 128, 2)
0.001
128
2
(0.001, 128, 3)
0.001
128
3
(0.0001, 16, 1)
0.0001
16
1
(0.0001, 16, 2)
0.0001
16
2
(0.0001, 16, 3)
0.0001
16
3
(0.0001, 32, 1)
0.0001
32
1
(0.0001, 32, 2)
0.0001
32
2
```

```
(0.0001, 32, 3)
0.0001
32
3
(0.0001, 128, 1)
0.0001
128
1
(0.0001, 128, 2)
0.0001
128
2
(0.0001, 128, 3)
0.0001
128
3
```

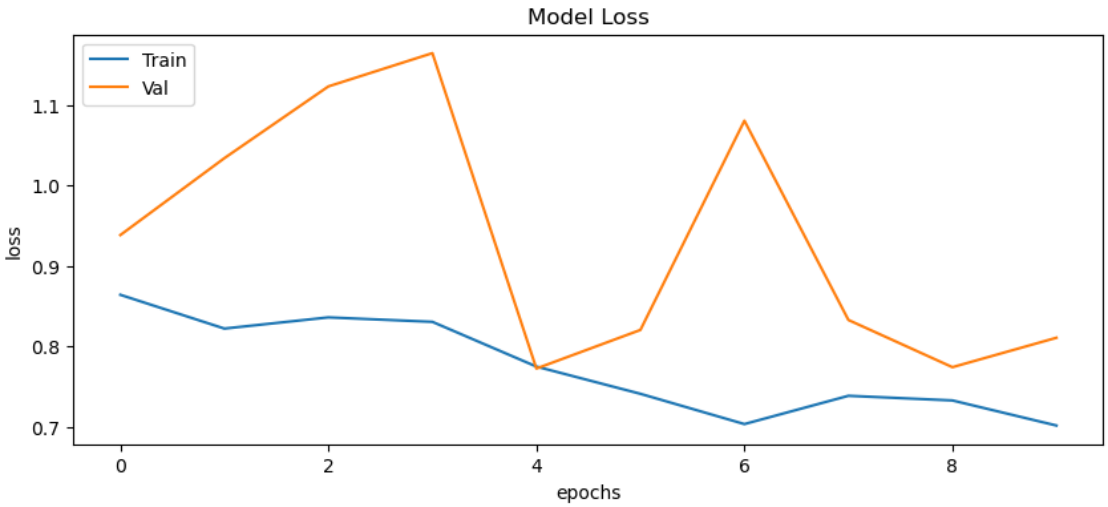
1.3 Training the Model

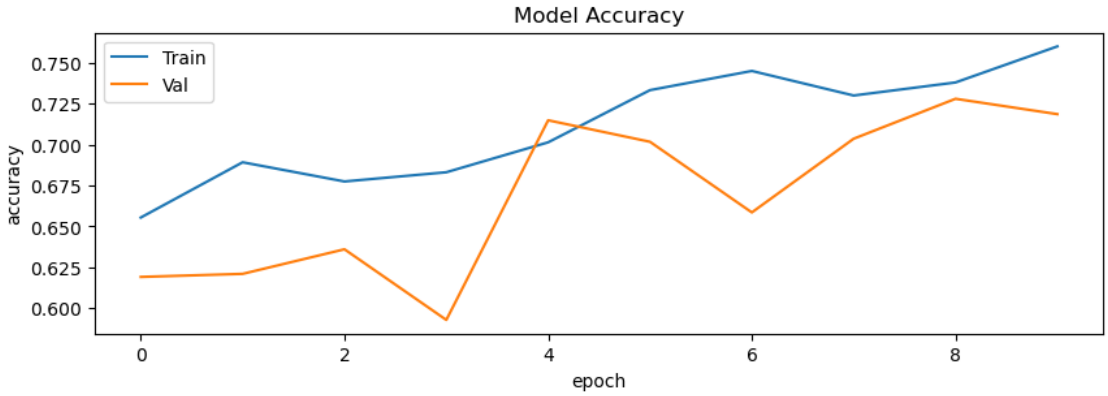
[71]:
```
history = model.fit(train_ds, validation_data=val_ds, epochs=10)
```

```
Epoch 1/10
34/34 [==============================] - 66s 2s/step - loss: 0.8641 - accuracy:
0.6554 - val_loss: 0.9384 - val_accuracy: 0.6191
Epoch 2/10
34/34 [==============================] - 66s 2s/step - loss: 0.8222 - accuracy:
0.6892 - val_loss: 1.0338 - val_accuracy: 0.6210
Epoch 3/10
34/34 [==============================] - 2043s 62s/step - loss: 0.8361 -
accuracy: 0.6774 - val_loss: 1.1229 - val_accuracy: 0.6360
Epoch 4/10
34/34 [==============================] - 65s 2s/step - loss: 0.8305 - accuracy:
0.6831 - val_loss: 1.1642 - val_accuracy: 0.5929
Epoch 5/10
34/34 [==============================] - 65s 2s/step - loss: 0.7750 - accuracy:
0.7014 - val_loss: 0.7725 - val_accuracy: 0.7148
Epoch 6/10
34/34 [==============================] - 65s 2s/step - loss: 0.7412 - accuracy:
0.7332 - val_loss: 0.8205 - val_accuracy: 0.7017
Epoch 7/10
34/34 [==============================] - 65s 2s/step - loss: 0.7036 - accuracy:
0.7450 - val_loss: 1.0803 - val_accuracy: 0.6585
Epoch 8/10
34/34 [==============================] - 65s 2s/step - loss: 0.7387 - accuracy:
0.7300 - val_loss: 0.8328 - val_accuracy: 0.7036
Epoch 9/10
34/34 [==============================] - 65s 2s/step - loss: 0.7329 - accuracy:
0.7379 - val_loss: 0.7742 - val_accuracy: 0.7280
```

```
Epoch 10/10
34/34 [==============================] - 65s 2s/step - loss: 0.7018 - accuracy:
0.7600 - val_loss: 0.8108 - val_accuracy: 0.7186
```

[72]:
```
%run rueegg_wissiak_model_visualization.ipynb
```





[73]:
```
model.evaluate(test_ds)
```

```
40/40 [==============================] - 13s 323ms/step - loss: 0.8089 -
accuracy: 0.7119
```

[73]:
```
[0.8089357018470764, 0.7118847370147705]
```