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.

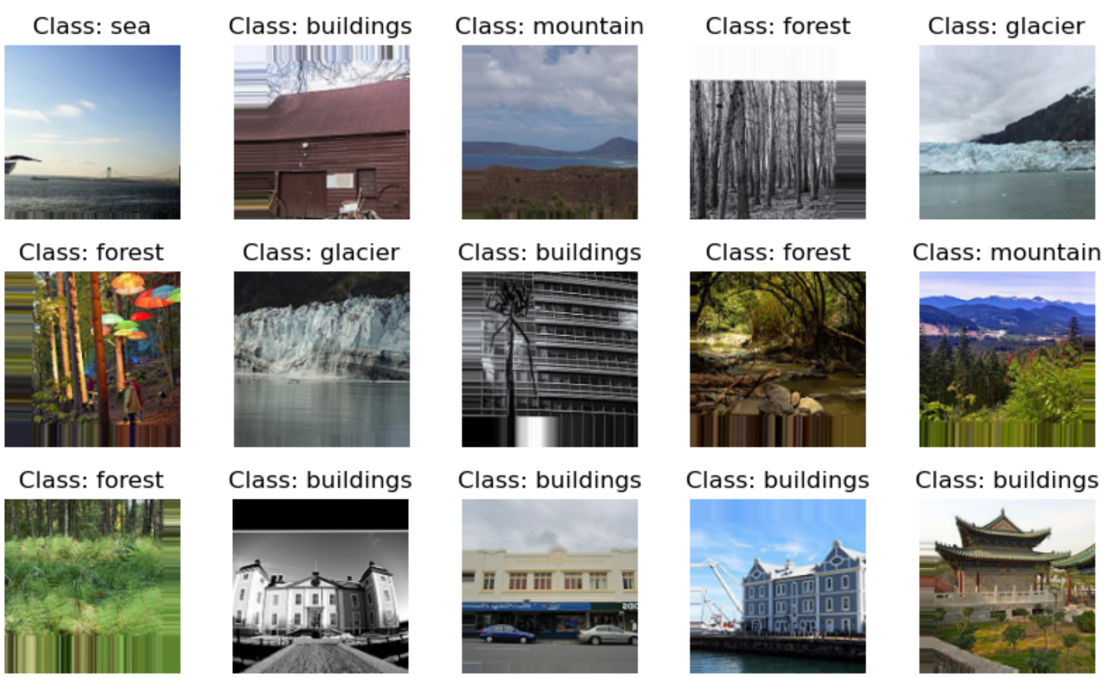
Adding dropout or weight decay can help to address both of the above mentioned issues. We can also try adjusting the hyperparameters such as learning rate, batch size, or number of epochs.

[20]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[21]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

img_size = 150
batch_size = 32

# TODO: add data augmentation for omptimized model
train_datagen = ImageDataGenerator(
    rescale=1/255.,
    rotation_range=1,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.005,
    zoom_range=0.05,
    horizontal_flip=True,
    fill_mode='nearest'
)
test_datagen = ImageDataGenerator(rescale=1/255.)

train_generator = train_datagen.flow_from_directory(
    './dataset/seg_train/seg_train',
    target_size=(img_size, img_size),
    batch_size=batch_size,
    shuffle=True,
```

```
    class_mode='sparse'
)
test_generator = test_datagen.flow_from_directory(
    './dataset/seg_test/seg_test',
    target_size=(img_size, img_size),
    batch_size=batch_size,
    shuffle=True,
    class_mode='sparse'
)
labels = ['buildings', 'forest', 'glacier', 'mountain', 'sea']


samples = train_generator.__next__()
images = samples[0]
target = samples[1]

plt.figure(figsize = (10,10))
for i in range(15):
    plt.subplot(5,5,i+1)
    plt.subplots_adjust(hspace=0.3,wspace=0.3)
    plt.imshow(images[i])
    plt.title(f"Class: {labels[int(target[i])]}")
    plt.axis('off')
```

```
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.

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.

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.

[22]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.optimizers.schedules import ExponentialDecay

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'))

lr_schedule = ExponentialDecay(
    initial_learning_rate=1e-2,
    decay_steps=10000,
    decay_rate=0.9)
model.compile(optimizer = SGD(learning_rate=lr_schedule),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

[23]:
```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 conv2d_35 (Conv2D)          (None, 150, 150, 32)      896

 max_pooling2d_35 (MaxPoolin  (None, 75, 75, 32)       0
 g2D)

 conv2d_36 (Conv2D)          (None, 75, 75, 64)        18496

 max_pooling2d_36 (MaxPoolin  (None, 37, 37, 64)       0
 g2D)

 conv2d_37 (Conv2D)          (None, 37, 37, 128)       73856

 max_pooling2d_37 (MaxPoolin  (None, 18, 18, 128)      0
 g2D)

 conv2d_38 (Conv2D)          (None, 18, 18, 256)       295168

 max_pooling2d_38 (MaxPoolin  (None, 9, 9, 256)        0
 g2D)

 conv2d_39 (Conv2D)          (None, 9, 9, 512)         1180160

 max_pooling2d_39 (MaxPoolin  (None, 4, 4, 512)        0
 g2D)

 dropout_15 (Dropout)        (None, 4, 4, 512)         0

 conv2d_40 (Conv2D)          (None, 4, 4, 1024)        4719616

 max_pooling2d_40 (MaxPoolin  (None, 2, 2, 1024)       0
 g2D)

 dropout_16 (Dropout)        (None, 2, 2, 1024)        0

 conv2d_41 (Conv2D)          (None, 2, 2, 512)         4719104

 max_pooling2d_41 (MaxPoolin  (None, 1, 1, 512)        0
 g2D)

 dropout_17 (Dropout)        (None, 1, 1, 512)         0

 flatten_5 (Flatten)         (None, 512)               0

 dense_10 (Dense)            (None, 256)               131328

 dense_11 (Dense)            (None, 5)                 1285

=================================================================
```

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

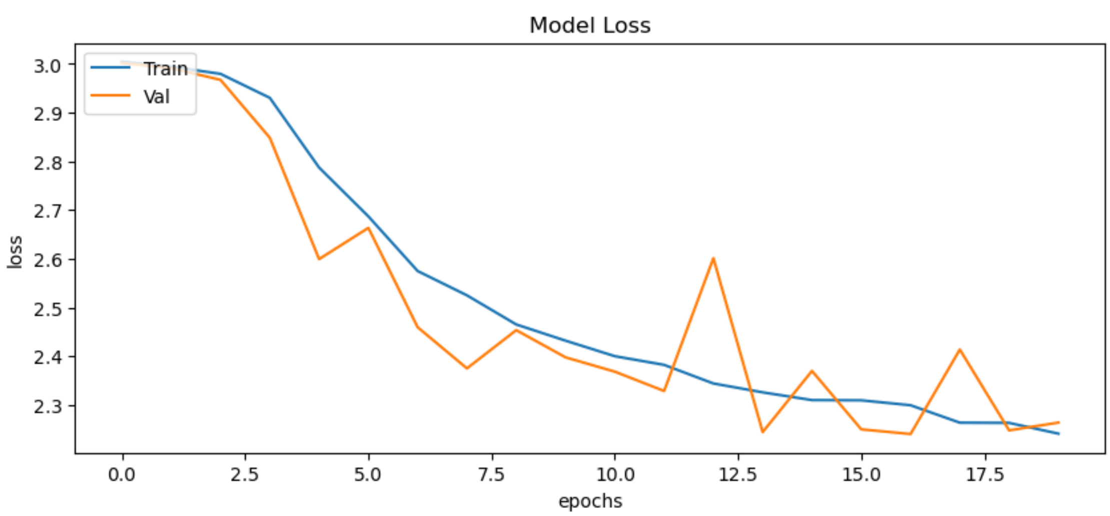
1.2 Training the Model

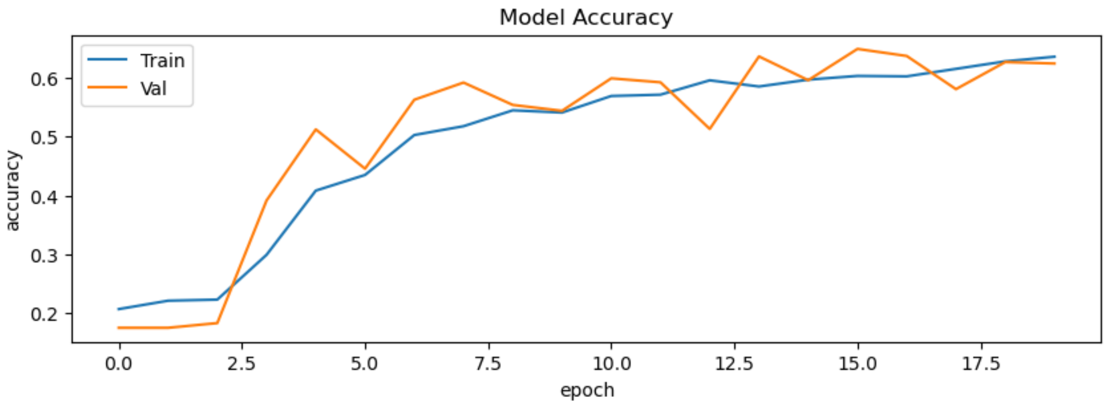
[24]:
```
history = model.fit(train_generator, validation_data=test_generator, epochs=20)
```

```
Epoch 1/20
84/84 [==============================] - 103s 1s/step - loss: 3.0046 - accuracy:
0.2067 - val_loss: 3.0021 - val_accuracy: 0.1749
Epoch 2/20
84/84 [==============================] - 103s 1s/step - loss: 2.9943 - accuracy:
0.2209 - val_loss: 2.9912 - val_accuracy: 0.1749
Epoch 3/20
84/84 [==============================] - 103s 1s/step - loss: 2.9795 - accuracy:
0.2228 - val_loss: 2.9672 - val_accuracy: 0.1829
Epoch 4/20
84/84 [==============================] - 103s 1s/step - loss: 2.9304 - accuracy:
0.2989 - val_loss: 2.8485 - val_accuracy: 0.3914
Epoch 5/20
84/84 [==============================] - 104s 1s/step - loss: 2.7874 - accuracy:
0.4081 - val_loss: 2.5995 - val_accuracy: 0.5122
Epoch 6/20
84/84 [==============================] - 103s 1s/step - loss: 2.6872 - accuracy:
0.4347 - val_loss: 2.6636 - val_accuracy: 0.4454
Epoch 7/20
84/84 [==============================] - 103s 1s/step - loss: 2.5751 - accuracy:
0.5026 - val_loss: 2.4598 - val_accuracy: 0.5626
Epoch 8/20
84/84 [==============================] - 104s 1s/step - loss: 2.5253 - accuracy:
0.5176 - val_loss: 2.3754 - val_accuracy: 0.5918
Epoch 9/20
84/84 [==============================] - 104s 1s/step - loss: 2.4655 - accuracy:
0.5446 - val_loss: 2.4539 - val_accuracy: 0.5538
Epoch 10/20
84/84 [==============================] - 103s 1s/step - loss: 2.4323 - accuracy:
0.5409 - val_loss: 2.3980 - val_accuracy: 0.5438
Epoch 11/20
84/84 [==============================] - 103s 1s/step - loss: 2.4004 - accuracy:
0.5690 - val_loss: 2.3687 - val_accuracy: 0.5990
Epoch 12/20
84/84 [==============================] - 103s 1s/step - loss: 2.3826 - accuracy:
0.5713 - val_loss: 2.3291 - val_accuracy: 0.5922
Epoch 13/20
84/84 [==============================] - 103s 1s/step - loss: 2.3445 - accuracy:
0.5956 - val_loss: 2.6014 - val_accuracy: 0.5130
```

```
Epoch 14/20
84/84 [==============================] - 103s 1s/step - loss: 2.3263 - accuracy:
0.5851 - val_loss: 2.2448 - val_accuracy: 0.6363
Epoch 15/20
84/84 [==============================] - 104s 1s/step - loss: 2.3104 - accuracy:
0.5968 - val_loss: 2.3702 - val_accuracy: 0.5958
Epoch 16/20
84/84 [==============================] - 103s 1s/step - loss: 2.3099 - accuracy:
0.6032 - val_loss: 2.2504 - val_accuracy: 0.6491
Epoch 17/20
84/84 [==============================] - 102s 1s/step - loss: 2.2998 - accuracy:
0.6024 - val_loss: 2.2408 - val_accuracy: 0.6371
Epoch 18/20
84/84 [==============================] - 103s 1s/step - loss: 2.2640 - accuracy:
0.6152 - val_loss: 2.4141 - val_accuracy: 0.5806
Epoch 19/20
84/84 [==============================] - 103s 1s/step - loss: 2.2636 - accuracy:
0.6279 - val_loss: 2.2483 - val_accuracy: 0.6267
Epoch 20/20
84/84 [==============================] - 103s 1s/step - loss: 2.2416 - accuracy:
0.6358 - val_loss: 2.2642 - val_accuracy: 0.6242
```

[25]:
```
%run rueegg_wissiak_model_visualization.ipynb
```





[26]:
```
%run rueegg_wissiak_model_evaluation.ipynb
```

```
79/79 [==============================] - 13s 168ms/step
Predicted classes: [4 2 0 … 0 0 2]
True labels: [0 0 0 … 4 4 4]
Accuracy:
 0.20968387354941978
```