rueegg_wissiak_underfitting

March 7, 2023

1 The Dataset

We used this dataset for our miniproject: https://www.kaggle.com/datasets/puneet6060/intel-image-classification

The dataset contains 6 classes of images: `building`, `forest`, `glacier`, `mountain`, `sea`, and `street`. We decided to delete the class `street` and it's corresponding pictures to simplify the learning process. The images are divided into folders with their respective labels. We used the `image_dataset_from_directory` method of Keras to convert the images into a TensorFlow dataset object for training.

1.1 Importing the Data

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn import metrics
from tensorflow import keras
from tensorflow.keras import layers
```

```
/opt/miniconda3/envs/aiap/lib/python3.9/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version
of SciPy (detected version 1.24.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

[2]:
```python
training_img_path = "./dataset/seg_train/seg_train/"

img_size = 50
batch_size = 32
seed = 42   # the seed will make sure the two datasets are not overlapping

train_ds = keras.utils.image_dataset_from_directory(
    training_img_path,
    validation_split=0.2,
    subset="training",
    labels="inferred",
    label_mode='int',
    seed=seed,
    image_size=(img_size, img_size),
    batch_size=batch_size
)

val_ds = keras.utils.image_dataset_from_directory(
    training_img_path,
    validation_split=0.2,
    subset="validation",
    labels="inferred",
    label_mode='int',
    seed=seed,
    image_size=(img_size, img_size),
    batch_size=batch_size
)

print(train_ds.class_names)
```

```
Found 11652 files belonging to 5 classes.
Using 9322 files for training.
Metal device set to: Apple M1 Pro

systemMemory: 32.00 GB
maxCacheSize: 10.67 GB


2023-03-07 19:40:01.932956: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305]
Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel
may not have been built with NUMA support.
2023-03-07 19:40:01.933110: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0
MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id:
<undefined>)
```

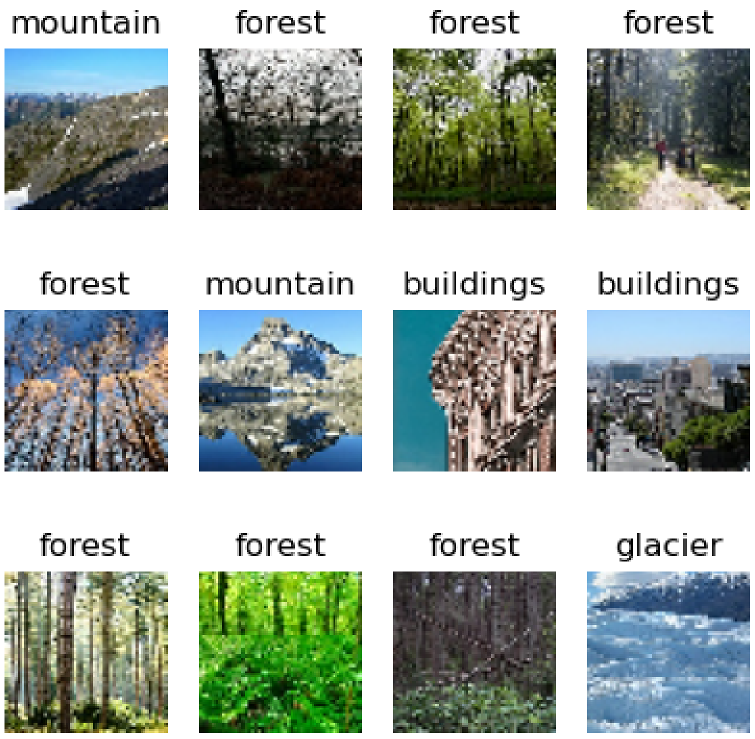
```
Found 11652 files belonging to 5 classes.
Using 2330 files for validation.
['buildings', 'forest', 'glacier', 'mountain', 'sea']
```

Let's see the first few images of the training dataset. Here, we'll define a function that we'll use again later in the notebook.

To ensure the model won't eat up too many of our computer's resources, we downscaled the images from $150 \times 150px$ to $50 \times 50px$.

[3]:
```
def show_dataset():
  plt.figure(figsize=(5, 5))
  for images, labels in train_ds.take(1):
    for i in range(12):
      ax = plt.subplot(3, 4, i + 1)
      plt.imshow(images[i].numpy().astype("uint8"))
      plt.title(train_ds.class_names[labels[i]])
      plt.axis("off")

show_dataset()
```

```
2023-03-07 19:40:02.516955: W
tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU
frequency: 0 Hz
```



2 Underfitting Model

We will create a model that is too simple and won't describe the data accurately enough.

2.1 Building the Model

We start with a convolutional layer that'll import/rescale the images to `img_size`, using the ReLU activation function (for all the convolution and dense layers).

Then, run a kernel of 3×3 over each image 16 times. `padding=same` is referring to the padding of the image (needed because of the kernel) being filled with zeros.

Following that is another convolution layer and max pooling layer. Next, the image will be flattened into a vector, ready to be fed to the following dense layer. In this case, we first wanted to apply dropout to the layer, but we ended up just halfing the amount of nodes of that layer. Why? We learned that the difference between applying `dropout(0.5)` and halving the nodes of the layer is that dropout randomly drops out nodes during each training iteration, which means that **different** nodes will be dropped out in each iteration. This allows the network to learn more robust and generalizable representations of the data, as it is forced to rely on a subset of nodes in each iteration, which prevents overfitting. While halving the nodes reduces the number of nodes in the layer permanently, which means that the network has less capacity to learn and represent complex patterns in the data. This then leads to underfitting, where the model is not able to capture the important features in the data. Which is exactly what we want. Also, dropout is present during training, but not during inference.

The last dense layer represents the output layer, having a shared softmax activation layer to determine the probabilities of the 5 different classes.

This task was interesting because we initially built a model that was too simple, even for underfitting. After the second epoch, it couldn't learn any more information because it simply was not complex enough. As a result, the accuracy stalled at around 0.25 and stayed the same for all the remaining epochs. We were not happy with that, so we now created a model that can actually improve with each epoch while still underfitting.

[4]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout

model = Sequential([
    Conv2D(6, (3,3), input_shape=(img_size,img_size,3), activation='relu',␣
 ↪padding='same'),
    MaxPooling2D(),

    Conv2D(6, (3,3), activation='relu', padding='same'),
    MaxPooling2D(),

    Flatten(),
    Dense(5, activation='softmax')
])
model._name='underfitting_model'

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

model.summary()
```

```
Model: "underfitting_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 50, 50, 6)         168
 max_pooling2d (MaxPooling2D  (None, 25, 25, 6)        0
 )

 conv2d_1 (Conv2D)           (None, 25, 25, 6)         330

 max_pooling2d_1 (MaxPooling  (None, 12, 12, 6)        0
 2D)

 flatten (Flatten)           (None, 864)               0

 dense (Dense)               (None, 5)                 4325

=================================================================
Total params: 4,823
Trainable params: 4,823
Non-trainable params: 0
_________________________________________________________________
```

2.2 Training the Model

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

```
Epoch 1/20
  1/292 […] - ETA: 1:32 - loss: 43.0652 - accuracy:
0.2188

2023-03-07 19:40:03.020454: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

292/292 [==============================] - ETA: 0s - loss: 2.8189 - accuracy:
0.2807

2023-03-07 19:40:06.189433: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

292/292 [==============================] - 4s 12ms/step - loss: 2.8189 -
accuracy: 0.2807 - val_loss: 1.5614 - val_accuracy: 0.3047
Epoch 2/20
292/292 [==============================] - 4s 12ms/step - loss: 1.4083 -
accuracy: 0.3932 - val_loss: 1.3505 - val_accuracy: 0.4090
Epoch 3/20
292/292 [==============================] - 4s 12ms/step - loss: 1.2653 -
accuracy: 0.4557 - val_loss: 1.2346 - val_accuracy: 0.4966
Epoch 4/20
```

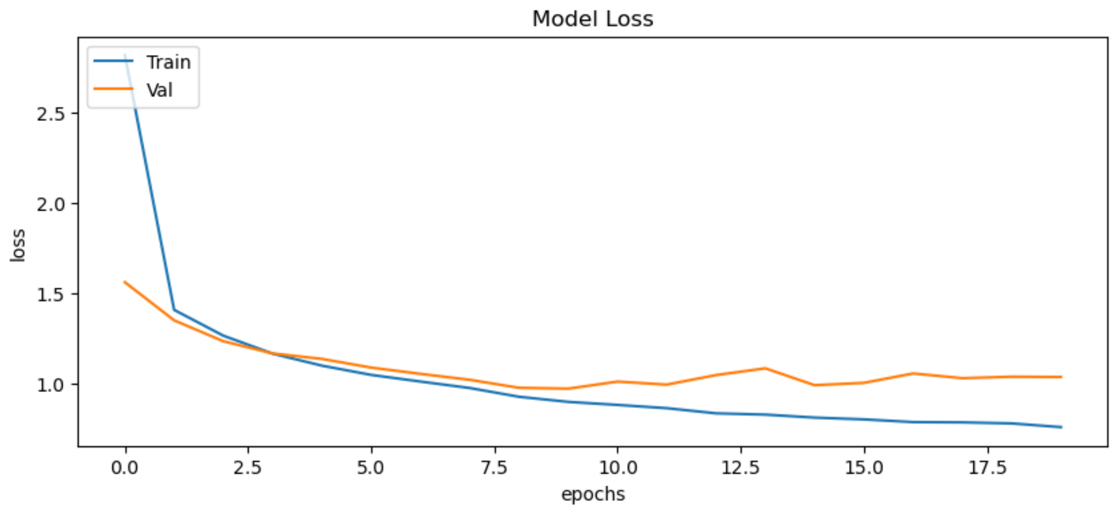
```
292/292 [==============================] - 4s 12ms/step - loss: 1.1659 -
accuracy: 0.5169 - val_loss: 1.1671 - val_accuracy: 0.5240
Epoch 5/20
292/292 [==============================] - 4s 12ms/step - loss: 1.0995 -
accuracy: 0.5427 - val_loss: 1.1368 - val_accuracy: 0.5382
Epoch 6/20
292/292 [==============================] - 4s 13ms/step - loss: 1.0483 -
accuracy: 0.5651 - val_loss: 1.0888 - val_accuracy: 0.5176
Epoch 7/20
292/292 [==============================] - 4s 13ms/step - loss: 1.0114 -
accuracy: 0.5602 - val_loss: 1.0541 - val_accuracy: 0.5309
Epoch 8/20
292/292 [==============================] - 4s 13ms/step - loss: 0.9758 -
accuracy: 0.5781 - val_loss: 1.0212 - val_accuracy: 0.5631
Epoch 9/20
292/292 [==============================] - 4s 13ms/step - loss: 0.9272 -
accuracy: 0.5971 - val_loss: 0.9769 - val_accuracy: 0.5820
Epoch 10/20
292/292 [==============================] - 5s 15ms/step - loss: 0.8989 -
accuracy: 0.6120 - val_loss: 0.9722 - val_accuracy: 0.5961
Epoch 11/20
292/292 [==============================] - 3s 12ms/step - loss: 0.8821 -
accuracy: 0.6250 - val_loss: 1.0113 - val_accuracy: 0.5991
Epoch 12/20
292/292 [==============================] - 4s 12ms/step - loss: 0.8640 -
accuracy: 0.6367 - val_loss: 0.9945 - val_accuracy: 0.5987
Epoch 13/20
292/292 [==============================] - 4s 13ms/step - loss: 0.8355 -
accuracy: 0.6543 - val_loss: 1.0474 - val_accuracy: 0.5953
Epoch 14/20
292/292 [==============================] - 4s 13ms/step - loss: 0.8284 -
accuracy: 0.6557 - val_loss: 1.0847 - val_accuracy: 0.6009
Epoch 15/20
292/292 [==============================] - 4s 15ms/step - loss: 0.8120 -
accuracy: 0.6646 - val_loss: 0.9913 - val_accuracy: 0.5983
Epoch 16/20
292/292 [==============================] - 5s 18ms/step - loss: 0.8021 -
accuracy: 0.6694 - val_loss: 1.0042 - val_accuracy: 0.5923
Epoch 17/20
292/292 [==============================] - 5s 17ms/step - loss: 0.7871 -
accuracy: 0.6779 - val_loss: 1.0562 - val_accuracy: 0.5979
Epoch 18/20
292/292 [==============================] - 5s 18ms/step - loss: 0.7855 -
accuracy: 0.6795 - val_loss: 1.0300 - val_accuracy: 0.6047
Epoch 19/20
292/292 [==============================] - 6s 20ms/step - loss: 0.7797 -
accuracy: 0.6781 - val_loss: 1.0383 - val_accuracy: 0.5996
Epoch 20/20
```

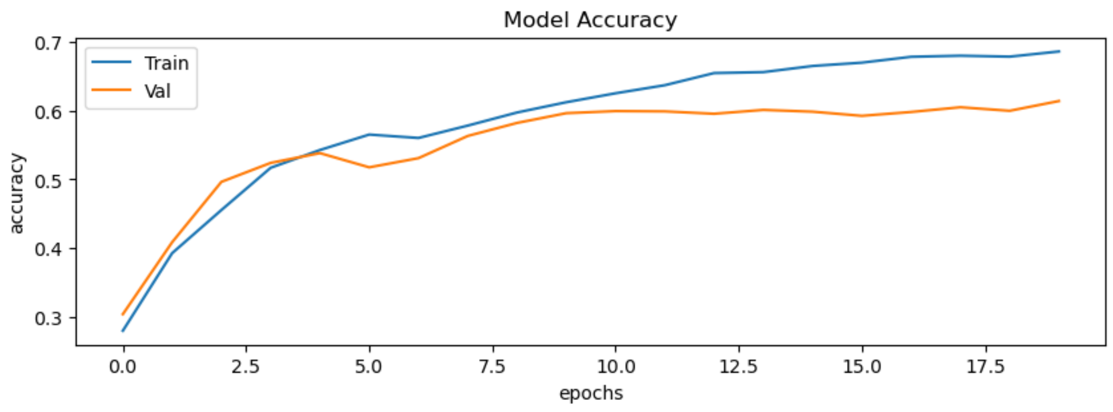
```
292/292 [==============================] - 5s 17ms/step - loss: 0.7593 -
accuracy: 0.6856 - val_loss: 1.0366 - val_accuracy: 0.6137
```

2.3 Visualize and analyze the model

Now, we'll have a look at how well the model performs.

[6]:
```python
def visualize_loss_and_accuracy():
    # Plot the training and validation Loss
    plt.figure(figsize=(10,4))
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.ylabel('loss')
    plt.xlabel('epochs')
    plt.legend(['Train','Val'], loc= 'upper left')
    plt.show()

    # Plot the model accuracy
    plt.figure(figsize=(10,3))
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epochs')
    plt.legend(['Train', 'Val'], loc='upper left')
    plt.show()

visualize_loss_and_accuracy()
```





As we see here, in the beginning, there's a gap in between the the validation and training loss and also in the accuracy. That is because of the randomness of the two different sample sets.

We can see from the learning curves that the model slowly converges and is indeed underfitting. Both the training and validation accuracy curves plateau at a low value, indicating that the model is not learning the patterns in the data well enough.

2.3.1 Make Prediction on the test dataset

[7]:
```
testing_img_path = "./dataset/seg_test/seg_test/"

test_ds = keras.utils.image_dataset_from_directory(
    testing_img_path,
    labels="inferred",
    label_mode='int',
    seed=seed,
    image_size=(img_size, img_size),
    batch_size=batch_size
)

predictions = model.predict(test_ds)
predicted_classes = np.argmax(predictions, axis=1)

true_classes = np.concatenate([y for x, y in test_ds], axis=0)

print('Predicted classes:', predicted_classes)
print('True labels:', true_classes)
```

```
Found 2499 files belonging to 5 classes.
24/79 [========>…] - ETA: 0s
```

```
2023-03-07 19:41:26.503843: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.
```

```
79/79 [==============================] - 0s 5ms/step
Predicted classes: [3 4 3 … 2 0 4]
True labels: [4 2 1 … 2 0 4]
```

2.3.2 Evaluate the performance of the model

[8]:
```
def evaluate_model():
    accuracy = model.evaluate(test_ds)[1]
    print(f'Accuracy: {accuracy}')

evaluate_model()
```

```
79/79 [==============================] - 1s 10ms/step - loss: 1.0694 - accuracy:
0.6098
Accuracy: 0.609843909740448
```

2.3.3 Classification Report

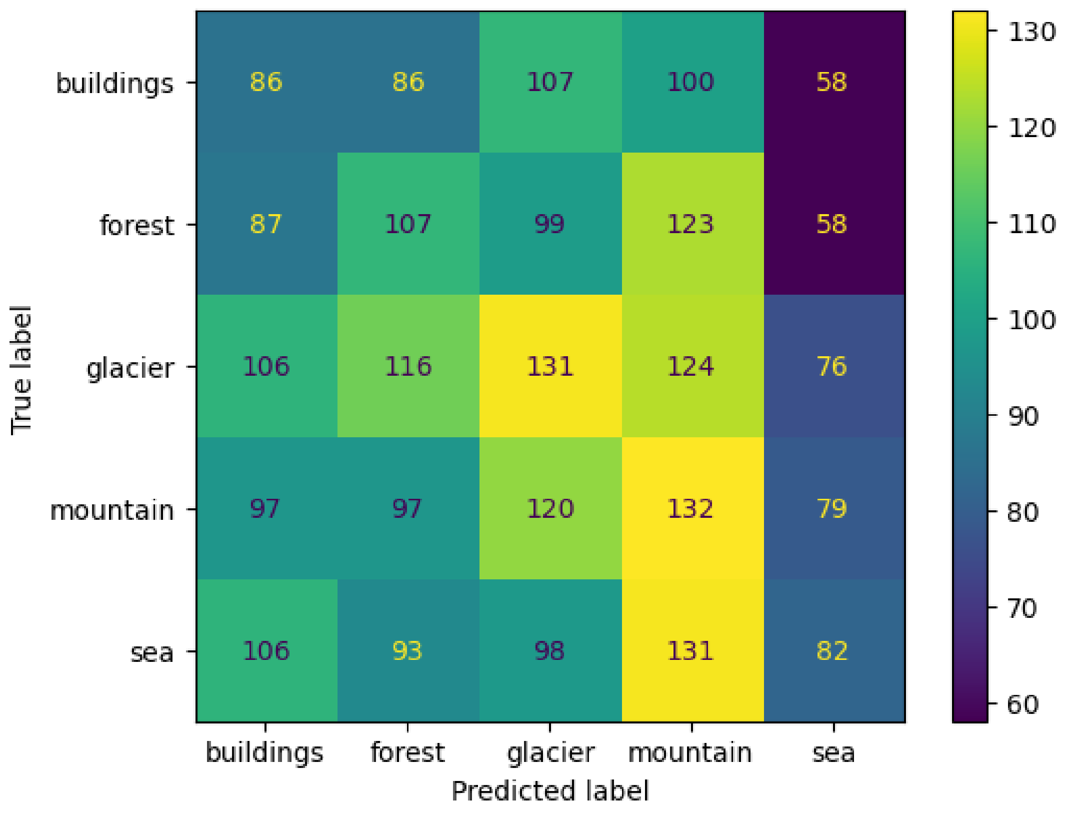
[9]:
```
print('Classification Report:\n', metrics.
 ↪classification_report(y_true=true_classes, y_pred=predicted_classes))

#https://www.kaggle.com/code/avantikab/intel-image-classification-cnn
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.18      0.20      0.19       437
           1       0.21      0.23      0.22       474
           2       0.24      0.24      0.24       553
           3       0.22      0.25      0.23       525
           4       0.23      0.16      0.19       510

    accuracy                           0.22      2499
   macro avg       0.22      0.21      0.21      2499
weighted avg       0.22      0.22      0.21      2499
```

2.3.4 Confusion Matrix

The confusion matrix is another way to visualize the performance of the model and wether it is underfitting or not.

[10]:
```
def print_confusion_matrix():
    confusion_matrix = metrics.confusion_matrix(y_true=true_classes,␣
 ↪y_pred=predicted_classes)
    display = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,␣
 ↪display_labels=test_ds.class_names)
    display.plot()
    plt.show()

print_confusion_matrix()
```



There is no clear diagonal line in the middle, in fact, the matrix shows that the mapping of the classes is all over the place. Which again confirms that this model is indeed underfitting.