# transfer-learning-different-configurations

December 23, 2022

[2]:
```python
# This Python 3 environment comes with many helpful analytics libraries␣
↪installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
↪docker-python
# For example, here's several helpful packages to load
from fastai.imports import *
from fastai.vision.all import *
import shutil
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from tensorflow.keras import regularizers
from tensorflow.keras import *


# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
↪all files under the input directory

import os
# for dirname, _, filenames in os.walk('/kaggle/input'):
#     for filename in filenames:
#         print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that␣
↪gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved␣
↪outside of the current session
```

## 1 making a food classifier with transfer learning

in this notebook we use the Food-101 dataset and transfer learning to efficiently train a classification model to classify 101 categories of food

## 1.1 loading our data

this dataset is also available in HDF5 format, we used this post to get some more information about it en learn how to use it.

[3]:
```
batch_size = 32
img_height = 224
img_width =224
```

[ ]:
```
import h5py
file = h5py.File('/kaggle/input/food41/food_c101_n1000_r384x384x3.h5', 'r')

images_train = file['images'][...]
category_labels_train = file['category'][...]#one-hot encoded representation of␣
↪our labels
category_names_train = file['category_names'][...]#name of each category

file.close()

file = h5py.File('/kaggle/input/food41/food_test_c101_n1000_r128x128x3.h5', 'r')

images_test = file['images'][...]
category_labels_test = file['category'][...]
category_names_test = file['category_names'][...]

file.close()
```

[ ]:
```
labels_train = np.where(category_labels_train == True)[1]
labels_test = np.where(category_labels_test == True)[1]
```

[ ]:
```
np.argmax(category_labels_test,axis=1).shape,images_test.shape
```

```
((1000,), (1000, 128, 128, 3))
```

[ ]:
```
# Split the training set into training and validation
images_val, images_train = images_train[0:int(len(images_train)*0.2)],␣
↪images_train[int(len(images_train)*0.2):]
category_labels_val, category_labels_train = labels_train[0:
↪int(len(labels_train)*0.2)], labels_train[int(len(labels_train)*0.2):]
category_names_val, category_names_train = category_names_train[0:
↪int(len(category_names_train)*0.2)],␣
↪category_names_train[int(len(category_names_train)*0.2):]
```

[ ]:
```
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.03),
    layers.RandomZoom(height_factor=(-0.1, 0.1)),
    layers.RandomContrast(factor=0.05),
```

```
])
```

```
2022-12-15 11:43:58.026061: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:43:58.027143: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:43:58.028218: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:43:58.028982: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:43:58.029799: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:43:58.030585: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:43:58.032268: I tensorflow/core/platform/cpu_feature_guard.cc:142]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2022-12-15 11:43:58.291439: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:43:58.292379: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:43:58.293184: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:43:58.293902: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
```

```
node, so returning NUMA node zero
2022-12-15 11:43:58.294638: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:43:58.295361: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:44:07.971875: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:44:07.972857: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:44:07.973571: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:44:07.974267: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:44:07.974942: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:44:07.975603: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 13349 MB memory:  -> device:
0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5
2022-12-15 11:44:07.980681: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2022-12-15 11:44:07.981391: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device
/job:localhost/replica:0/task:0/device:GPU:1 with 13349 MB memory:  -> device:
1, name: Tesla T4, pci bus id: 0000:00:05.0, compute capability: 7.5
```

We will not rescale the images because the pretrained models that we're going to use already have a rescaling layer.(We realised this after spending countles hours optimising our models and swapping out architectures ( ° °;)

[ ]:
```
resize = tf.keras.Sequential([
  layers.Resizing(img_width, img_height),
```

```
])
```

[ ]:
```
class_names=[]
```

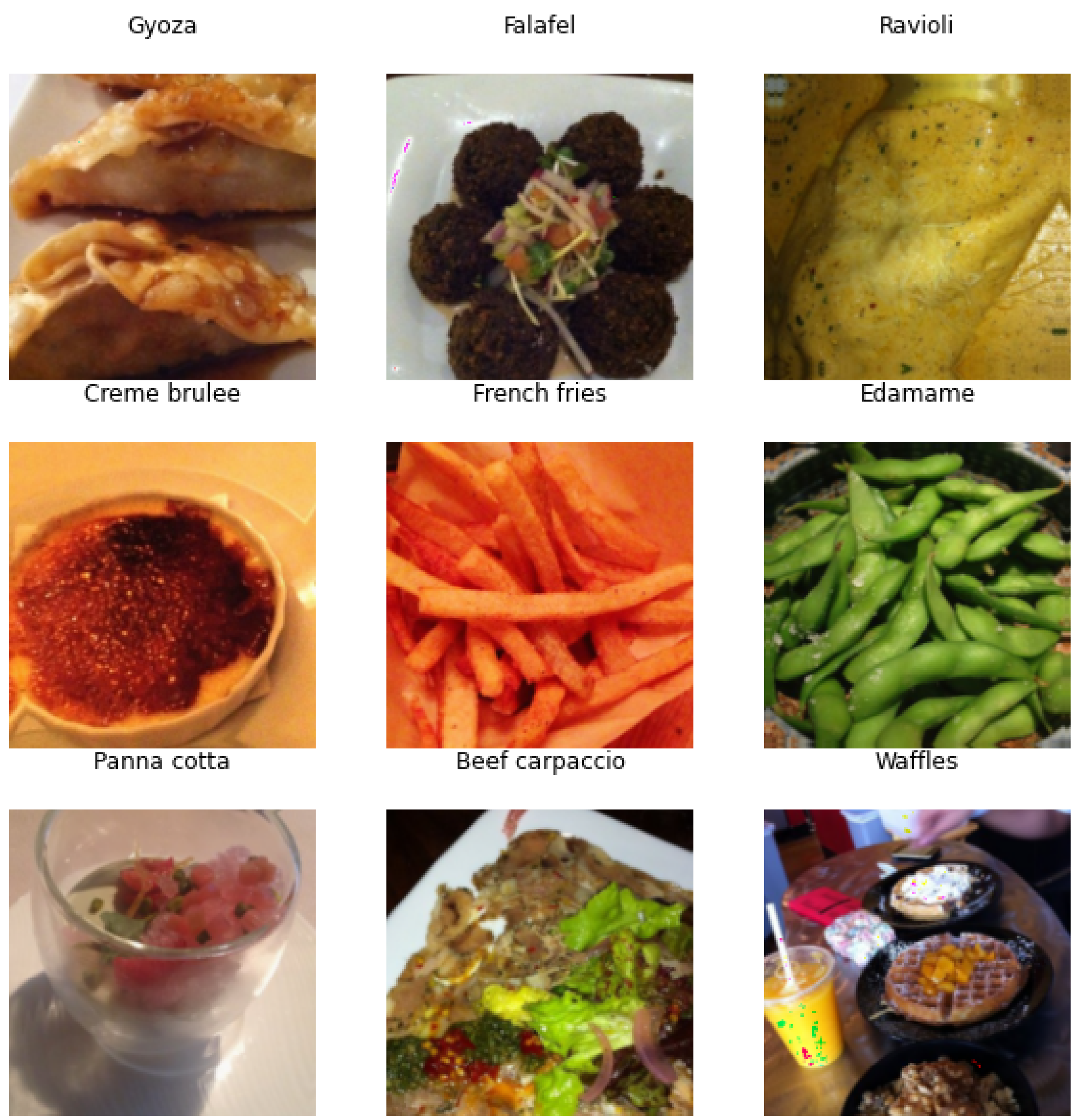
[ ]:
```
fo = open("/kaggle/input/food41/meta/meta/labels.txt")
for line in fo:
    class_names.append(line)
fo.close()
```

[ ]:
```
def prepare_dataset(image,label, batch_size=32, b_shuffle=True,augment=True):
    # transform input data into tf.data
    ds = tf.data.Dataset.from_tensor_slices((image, label))

    ds = ds.map(map_func = preprocessing ,num_parallel_calls = tf.data.
 ↪experimental.AUTOTUNE)
    # normally you only need to shuffle the training data
    if b_shuffle == True:
        ds = ds.shuffle(len(ds))
    # normally you only need to augment the training data
    if augment:
        ds = ds.map(lambda x, y: (data_augmentation(x, training=True),␣
 ↪y),num_parallel_calls=tf.data.experimental.AUTOTUNE)
    ds = ds.batch(batch_size)
    ds = ds.cache()
    ds = ds.prefetch(buffer_size = tf.data.experimental.AUTOTUNE)
    return ds



def preprocessing(image, label):
    image = resize(image)
    return image, label


train_ds = prepare_dataset(images_train, category_labels_train,augment=True)
val_ds = prepare_dataset(images_val, category_labels_val, b_shuffle =␣
 ↪False,augment=False)
test_ds = prepare_dataset(images_test,labels_test, b_shuffle =␣
 ↪False,augment=False)
```

[ ]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))

for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
```

```
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```

```
2022-12-15 11:44:10.616472: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)
2022-12-15 11:44:12.006115: W
tensorflow/core/kernels/data/cache_dataset_ops.cc:768] The calling iterator did
not fully read the dataset being cached. In order to avoid unexpected truncation
of the dataset, the partially cached contents of the dataset  will be discarded.
This can happen if you have an input pipeline similar to
`dataset.cache().take(k).repeat()`. You should use
`dataset.take(k).cache().repeat()` instead.
```



### 1.2 Testing out different architectures and methods

[13]:
```
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=5,
    min_delta=0.01,
    restore_best_weights=True,
)

plateau = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.2,
    patience=3,
    min_delta=0.01,
    cooldown=0,
    verbose=1
)
```

#### 1.2.1 EfficientNetB3 with fine tuning the top layer

[14]:
```
EfficientNetB3=tf.keras.applications.EfficientNetB3(
    include_top=False,
    weights="imagenet",
    input_shape=(img_width, img_width, 3),
    pooling=None,
    classes=101,
)
```

```
Downloading data from https://storage.googleapis.com/keras-
applications/efficientnetb3_notop.h5
43941888/43941136 [==============================] - 0s 0us/step
43950080/43941136 [==============================] - 0s 0us/step
```

[15]:
```
base_out = EfficientNetB3.output
x = tf.keras.layers.GlobalAveragePooling2D()(base_out)
output = layers.Dense(len(class_names), activation='softmax')(x)

model_TL = models.Model(EfficientNetB3.input, output)
```

[16]:
```
len(EfficientNetB3.layers)
```

[16]: 384

we will first freeze the base model and train our classification layers on top because we don't want to break our models weights.The pretrained weights we're using are from Imagenet. We have a chance of breaking the pretrained model because our source domain is much different from our target domain. Generaly only the lower level layers (that recognize less complex shapes) will be usefull for us. The further on top the node is, the more specialized it becomes at detecting specific elements in images.

[17]:
```
# freezing the base_model:
for layer in EfficientNetB3.layers[:]:
    layer.trainable = False
```

[18]:
```
from tensorflow.keras.optimizers import Adam
model_TL.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                 optimizer="adam",
                 metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])
```

[19]:
```
history = model_TL.fit(train_ds,
                        validation_data=val_ds,
                        epochs=2,
                        verbose=1)
```

```
Epoch 1/2
```

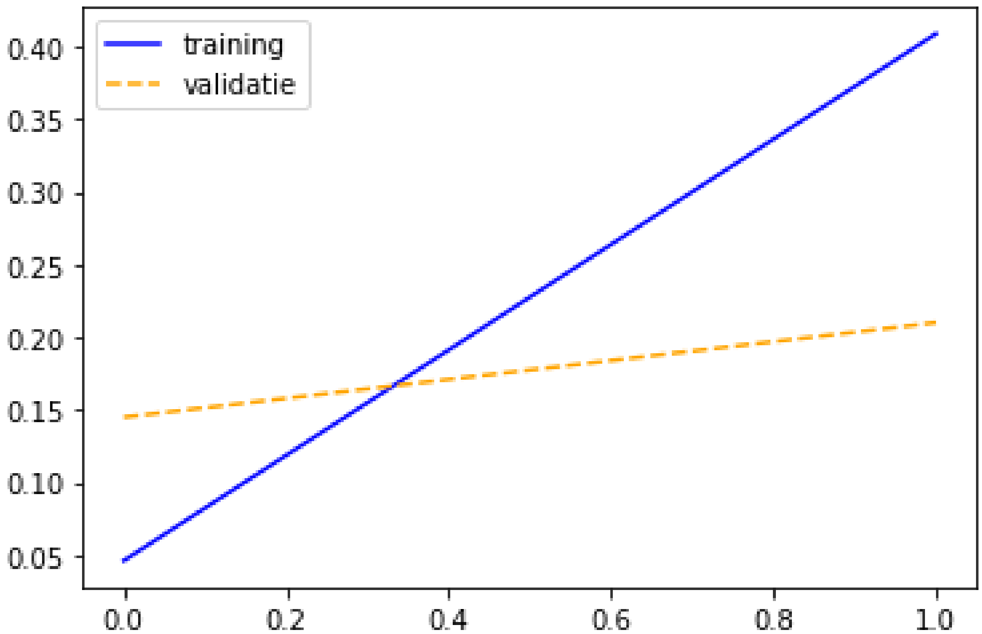
```
2022-12-16 20:22:56.122335: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369]
Loaded cuDNN version 8005
```

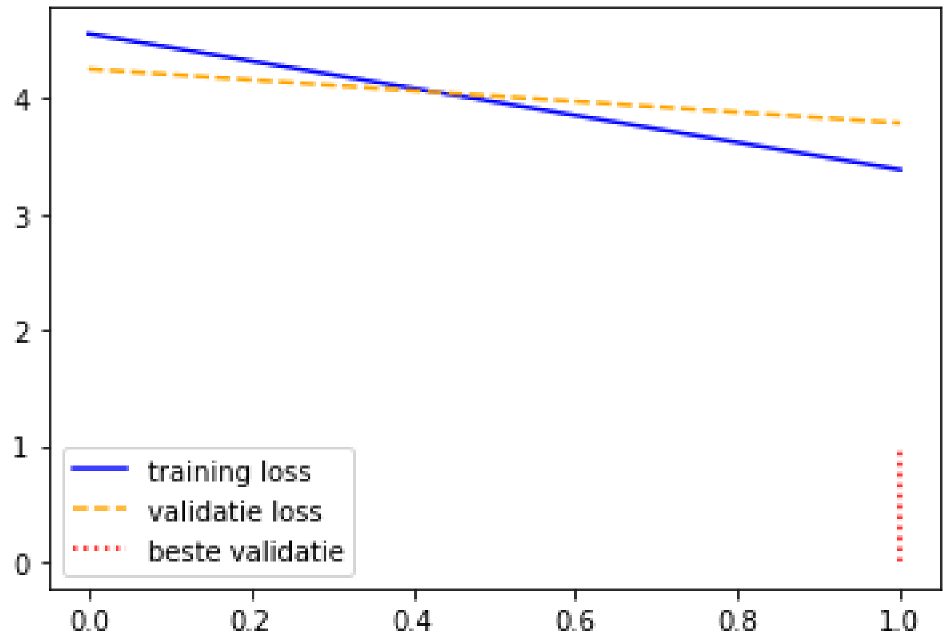
```
25/25 [==============================] - 28s 441ms/step - loss: 4.5543 -
sparse_categorical_accuracy: 0.0463 - val_loss: 4.2512 -
val_sparse_categorical_accuracy: 0.1450
Epoch 2/2
25/25 [==============================] - 4s 143ms/step - loss: 3.3822 -
sparse_categorical_accuracy: 0.4087 - val_loss: 3.7852 -
val_sparse_categorical_accuracy: 0.2100
```

look at the loss curves

[20]:
```
train_loss_values = history.history['loss']
val_loss_values = history.history['val_loss']
best_val_idx = np.argmin(val_loss_values)
num_epochs = range(len(train_loss_values))

plt.plot(num_epochs, train_loss_values, label='training loss', color='blue',␣
 ↪ls='-')
plt.plot(num_epochs, val_loss_values, label='validatie loss', color='orange',␣
 ↪ls='--')
plt.vlines(x=best_val_idx, ymin=0, ymax=1, label='beste validatie',␣
 ↪color='red', ls=':')
plt.legend()
plt.figure(0)
train_loss_values = history.history['sparse_categorical_accuracy']
val_loss_values = history.history['val_sparse_categorical_accuracy']
```

```
num_epochs = range(len(train_loss_values))

plt.plot(num_epochs, train_loss_values, label='training', color='blue', ls='-')
plt.plot(num_epochs, val_loss_values, label='validatie', color='orange',␣
 ↪ls='--')
plt.legend()
plt.figure(1)
plt.show()
```






```
[21]: loss, categorical_accuracy = model_TL.evaluate(test_ds, verbose=1)
      loss, categorical_accuracy
```

```
32/32 [==============================] - 5s 141ms/step - loss: 3.7637 -
sparse_categorical_accuracy: 0.2070
```

```
[21]: (3.7636868953704834, 0.2070000022649765)
```

```
[22]: # unfreezing the base model but keeping the first 190 weights frozen:

      for layer in EfficientNetB3.layers[:190]:
          layer.trainable = False

      for layer in EfficientNetB3.layers[190:]:
          layer.trainable = True
```

recompile the model with slow LR

```
[23]: model_TL.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                        optimizer=Adam(
                      learning_rate=0.0002,
                      beta_1=0.9,
                      beta_2=0.999,
                      epsilon=1e-08 ),
                        metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])
```

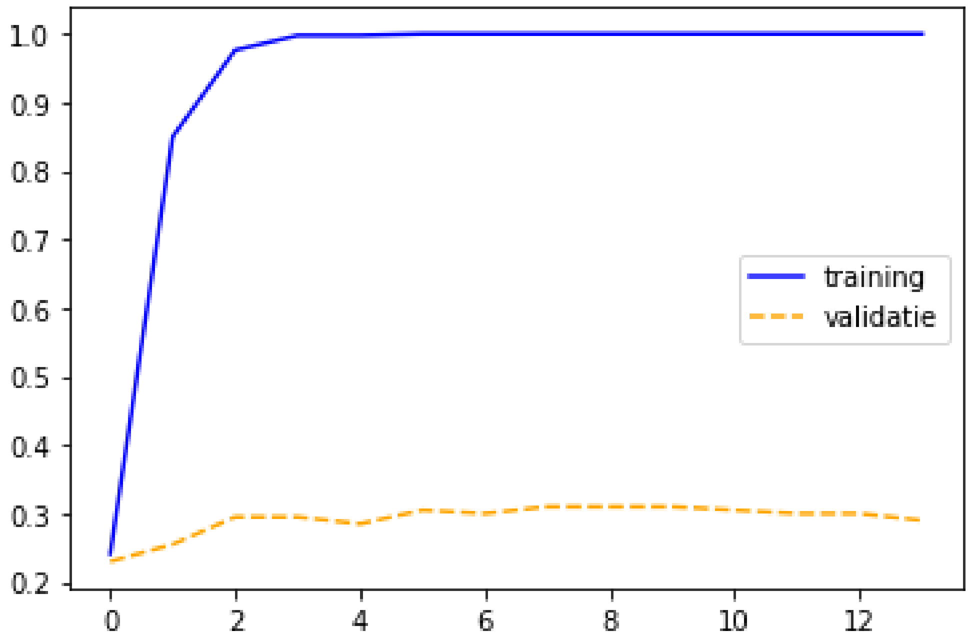
[24]:
```python
history = model_TL.fit(train_ds,
                         validation_data=val_ds,
                         epochs=20,
                         callbacks=[early_stopping, plateau],
                         verbose=1)
```

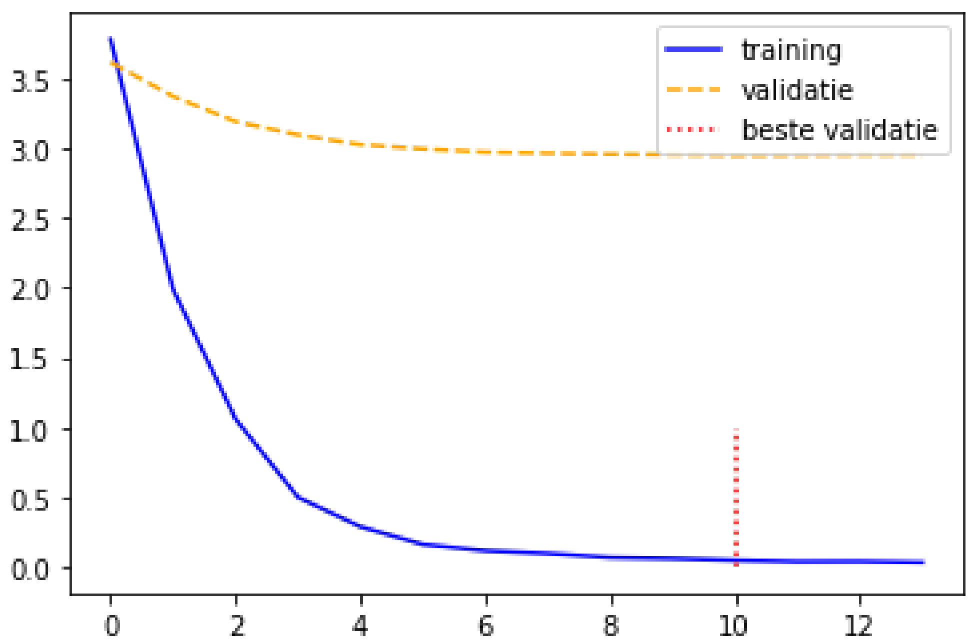
```
Epoch 1/20
25/25 [==============================] - 20s 334ms/step - loss: 3.7838 -
sparse_categorical_accuracy: 0.2412 - val_loss: 3.6198 -
val_sparse_categorical_accuracy: 0.2300
Epoch 2/20
25/25 [==============================] - 6s 241ms/step - loss: 1.9903 -
sparse_categorical_accuracy: 0.8500 - val_loss: 3.3750 -
val_sparse_categorical_accuracy: 0.2550
Epoch 3/20
25/25 [==============================] - 6s 242ms/step - loss: 1.0604 -
sparse_categorical_accuracy: 0.9762 - val_loss: 3.1947 -
val_sparse_categorical_accuracy: 0.2950
Epoch 4/20
25/25 [==============================] - 6s 244ms/step - loss: 0.4982 -
sparse_categorical_accuracy: 0.9975 - val_loss: 3.0989 -
val_sparse_categorical_accuracy: 0.2950
Epoch 5/20
25/25 [==============================] - 6s 246ms/step - loss: 0.2861 -
sparse_categorical_accuracy: 0.9975 - val_loss: 3.0292 -
val_sparse_categorical_accuracy: 0.2850
Epoch 6/20
25/25 [==============================] - 6s 250ms/step - loss: 0.1589 -
sparse_categorical_accuracy: 1.0000 - val_loss: 2.9981 -
val_sparse_categorical_accuracy: 0.3050
Epoch 7/20
25/25 [==============================] - 6s 247ms/step - loss: 0.1134 -
sparse_categorical_accuracy: 1.0000 - val_loss: 2.9748 -
val_sparse_categorical_accuracy: 0.3000
Epoch 8/20
25/25 [==============================] - 6s 249ms/step - loss: 0.0937 -
sparse_categorical_accuracy: 1.0000 - val_loss: 2.9664 -
val_sparse_categorical_accuracy: 0.3100
Epoch 9/20
25/25 [==============================] - 6s 253ms/step - loss: 0.0649 -
sparse_categorical_accuracy: 1.0000 - val_loss: 2.9599 -
val_sparse_categorical_accuracy: 0.3100
Epoch 10/20
25/25 [==============================] - 6s 252ms/step - loss: 0.0577 -
sparse_categorical_accuracy: 1.0000 - val_loss: 2.9563 -
val_sparse_categorical_accuracy: 0.3100
Epoch 11/20
```

```
25/25 [==============================] - 6s 255ms/step - loss: 0.0450 -
sparse_categorical_accuracy: 1.0000 - val_loss: 2.9506 -
val_sparse_categorical_accuracy: 0.3050
Epoch 12/20
25/25 [==============================] - 6s 256ms/step - loss: 0.0355 -
sparse_categorical_accuracy: 1.0000 - val_loss: 2.9520 -
val_sparse_categorical_accuracy: 0.3000

Epoch 00012: ReduceLROnPlateau reducing learning rate to 3.9999998989515007e-05.
Epoch 13/20
25/25 [==============================] - 6s 256ms/step - loss: 0.0365 -
sparse_categorical_accuracy: 1.0000 - val_loss: 2.9545 -
val_sparse_categorical_accuracy: 0.3000
Epoch 14/20
25/25 [==============================] - 6s 258ms/step - loss: 0.0296 -
sparse_categorical_accuracy: 1.0000 - val_loss: 2.9556 -
val_sparse_categorical_accuracy: 0.2900
```

[25]:

```python
train_loss_values = history.history['loss']
val_loss_values = history.history['val_loss']
best_val_idx = np.argmin(val_loss_values)
num_epochs = range(len(train_loss_values))

plt.plot(num_epochs, train_loss_values, label='training', color='blue', ls='-')
plt.plot(num_epochs, val_loss_values, label='validatie', color='orange',␣
 ↪ls='--')
plt.vlines(x=best_val_idx, ymin=0, ymax=1, label='beste validatie',␣
 ↪color='red', ls=':')
plt.legend()
plt.figure(0)
train_loss_values = history.history['sparse_categorical_accuracy']
val_loss_values = history.history['val_sparse_categorical_accuracy']
num_epochs = range(len(train_loss_values))

plt.plot(num_epochs, train_loss_values, label='training', color='blue', ls='-')
plt.plot(num_epochs, val_loss_values, label='validatie', color='orange',␣
 ↪ls='--')
plt.legend()
plt.figure(1)
plt.show()
```




[26]:
```
loss, categorical_accuracy = model_TL.evaluate(test_ds, verbose=1)
loss, categorical_accuracy
```

```
32/32 [==============================] - 4s 114ms/step - loss: 3.1993 -
sparse_categorical_accuracy: 0.2630
```

[26]: (3.199324131011963, 0.2630000114440918)

[27]:
```
model_TL.save_weights('EfficientNetB3_finetuned.h5')
```

### 1.2.2 EfficientNetB3 finetune with a more complex top layer

[28]:
```
EfficientNetB3=tf.keras.applications.EfficientNetB3(
    include_top=False,
    weights="imagenet",
    input_shape=(img_width, img_width, 3),
    pooling=None,
    classes=101,
)

base_out = EfficientNetB3.output
x = tf.keras.layers.GlobalAveragePooling2D()(base_out)
x = layers.Dense(256, activation='relu',kernel_regularizer=regularizers.l2(0.
 ↪001))(x)
x = layers.Dropout(0.5)(x)
output = layers.Dense(len(class_names), activation='softmax')(x)

model_TL_complex = models.Model(EfficientNetB3.input, output)

# freezing the base_model:
for layer in EfficientNetB3.layers[:]:
    layer.trainable = False

from tensorflow.keras.optimizers import Adam
model_TL_complex.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                 optimizer="adam",
                 metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])

history = model_TL_complex.fit(train_ds,
                         validation_data=val_ds,
                         epochs=4,
                         verbose=1)
# unfreezing the base model but keeping the first 190 weights frozen:

for layer in EfficientNetB3.layers[:190]:
    layer.trainable = False

for layer in EfficientNetB3.layers[190:]:
    layer.trainable = True

model_TL_complex.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                 optimizer=Adam(
                learning_rate=0.0002,
```

```
                beta_1=0.9,
                beta_2=0.999,
                epsilon=1e-08 ),
                 metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])

history = model_TL_complex.fit(train_ds,
                         validation_data=val_ds,
                         epochs=20,
                         callbacks=[early_stopping, plateau],
                         verbose=1)
```

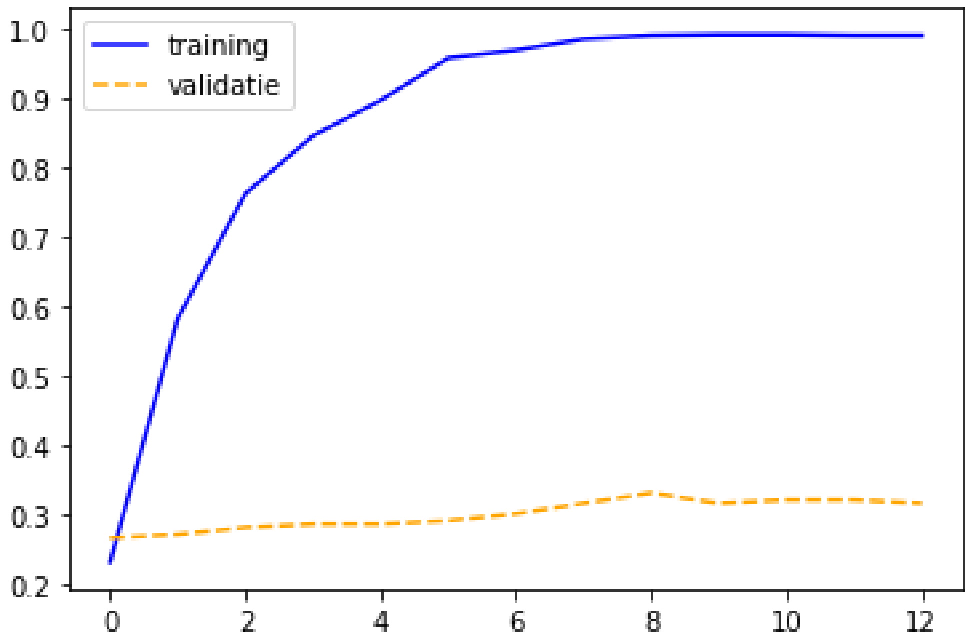
```
Epoch 1/4
25/25 [==============================] - 14s 234ms/step - loss: 5.0701 -
sparse_categorical_accuracy: 0.0300 - val_loss: 4.8103 -
val_sparse_categorical_accuracy: 0.1050
Epoch 2/4
25/25 [==============================] - 4s 160ms/step - loss: 4.2710 -
sparse_categorical_accuracy: 0.1887 - val_loss: 4.4552 -
val_sparse_categorical_accuracy: 0.1600
Epoch 3/4
25/25 [==============================] - 4s 157ms/step - loss: 3.5050 -
sparse_categorical_accuracy: 0.3550 - val_loss: 4.0535 -
val_sparse_categorical_accuracy: 0.2350
Epoch 4/4
25/25 [==============================] - 4s 157ms/step - loss: 2.8753 -
sparse_categorical_accuracy: 0.4950 - val_loss: 3.7850 -
val_sparse_categorical_accuracy: 0.2750
Epoch 1/20
25/25 [==============================] - 19s 342ms/step - loss: 4.1006 -
sparse_categorical_accuracy: 0.2300 - val_loss: 3.8173 -
val_sparse_categorical_accuracy: 0.2650
Epoch 2/20
25/25 [==============================] - 7s 266ms/step - loss: 2.7366 -
sparse_categorical_accuracy: 0.5825 - val_loss: 3.7632 -
val_sparse_categorical_accuracy: 0.2700
Epoch 3/20
25/25 [==============================] - 7s 269ms/step - loss: 1.9212 -
sparse_categorical_accuracy: 0.7625 - val_loss: 3.6851 -
val_sparse_categorical_accuracy: 0.2800
Epoch 4/20
25/25 [==============================] - 7s 260ms/step - loss: 1.4611 -
sparse_categorical_accuracy: 0.8462 - val_loss: 3.6088 -
val_sparse_categorical_accuracy: 0.2850
Epoch 5/20
25/25 [==============================] - 7s 262ms/step - loss: 1.1571 -
sparse_categorical_accuracy: 0.8975 - val_loss: 3.6123 -
val_sparse_categorical_accuracy: 0.2850
```

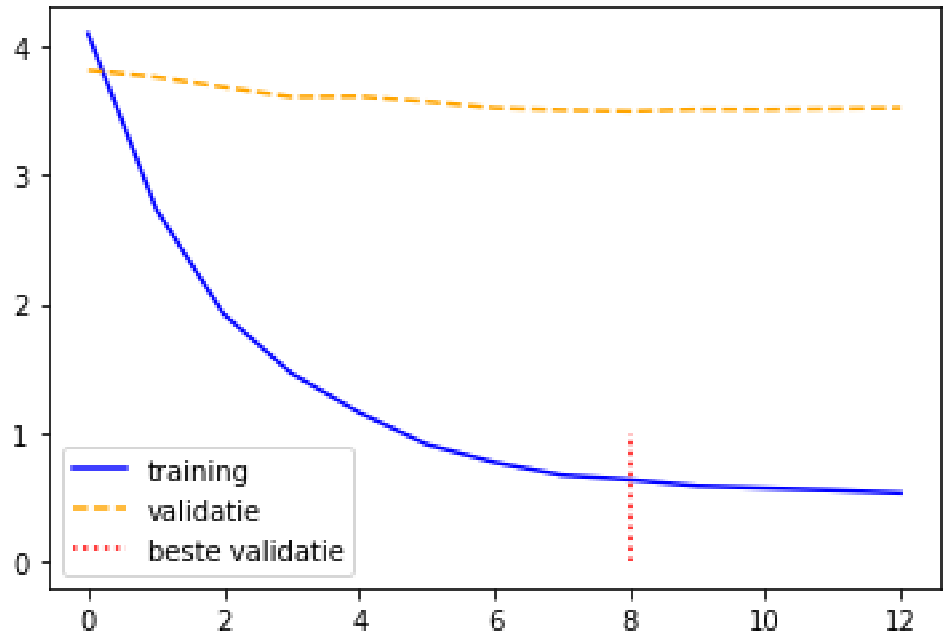
```
Epoch 6/20
25/25 [==============================] - 7s 261ms/step - loss: 0.9082 -
sparse_categorical_accuracy: 0.9588 - val_loss: 3.5724 -
val_sparse_categorical_accuracy: 0.2900
Epoch 7/20
25/25 [==============================] - 6s 258ms/step - loss: 0.7684 -
sparse_categorical_accuracy: 0.9700 - val_loss: 3.5225 -
val_sparse_categorical_accuracy: 0.3000
Epoch 8/20
25/25 [==============================] - 6s 260ms/step - loss: 0.6688 -
sparse_categorical_accuracy: 0.9862 - val_loss: 3.5066 -
val_sparse_categorical_accuracy: 0.3150
Epoch 9/20
25/25 [==============================] - 7s 263ms/step - loss: 0.6330 -
sparse_categorical_accuracy: 0.9912 - val_loss: 3.4973 -
val_sparse_categorical_accuracy: 0.3300
Epoch 10/20
25/25 [==============================] - 7s 266ms/step - loss: 0.5843 -
sparse_categorical_accuracy: 0.9925 - val_loss: 3.5118 -
val_sparse_categorical_accuracy: 0.3150
Epoch 11/20
25/25 [==============================] - 7s 263ms/step - loss: 0.5677 -
sparse_categorical_accuracy: 0.9925 - val_loss: 3.5099 -
val_sparse_categorical_accuracy: 0.3200

Epoch 00011: ReduceLROnPlateau reducing learning rate to 3.9999998989515007e-05.
Epoch 12/20
25/25 [==============================] - 7s 260ms/step - loss: 0.5517 -
sparse_categorical_accuracy: 0.9912 - val_loss: 3.5162 -
val_sparse_categorical_accuracy: 0.3200
Epoch 13/20
25/25 [==============================] - 7s 260ms/step - loss: 0.5309 -
sparse_categorical_accuracy: 0.9912 - val_loss: 3.5225 -
val_sparse_categorical_accuracy: 0.3150
```

[29]:

```
train_loss_values = history.history['loss']
val_loss_values = history.history['val_loss']
best_val_idx = np.argmin(val_loss_values)
num_epochs = range(len(train_loss_values))

plt.plot(num_epochs, train_loss_values, label='training', color='blue', ls='-')
plt.plot(num_epochs, val_loss_values, label='validatie', color='orange',␣
 ↪ls='--')
plt.vlines(x=best_val_idx, ymin=0, ymax=1, label='beste validatie',␣
 ↪color='red', ls=':')
plt.legend()
plt.figure(0)
```

```
train_loss_values = history.history['sparse_categorical_accuracy']
val_loss_values = history.history['val_sparse_categorical_accuracy']
num_epochs = range(len(train_loss_values))

plt.plot(num_epochs, train_loss_values, label='training', color='blue', ls='-')
plt.plot(num_epochs, val_loss_values, label='validatie', color='orange',␣
 ↪ls='--')
plt.legend()
plt.figure(1)
plt.show()
```






[30]:
```
loss, categorical_accuracy = model_TL_complex.evaluate(test_ds, verbose=1)
loss, categorical_accuracy
```

```
32/32 [==============================] - 4s 115ms/step - loss: 3.6424 -
sparse_categorical_accuracy: 0.2720
```

[30]:
```
(3.64237642288208, 0.2720000147819519)
```

[31]:
```
model_TL_complex.save_weights('EfficientNetB3_complex_finetuned.h5')
```

the more complex top layer did not make a big difference

### 1.2.3 EfficientNetB3 with different fine tuning approach

instead of keeping the first 190 layers frozen we will try to see what the effect is if we keep the first 380 layers frozen and anly finetine a handfull of convolution layers on top instead of half of the pretrained model

[32]:
```
EfficientNetB3=tf.keras.applications.EfficientNetB3(
    include_top=False,
    weights="imagenet",
    input_shape=(img_width, img_width, 3),
    pooling=None,
    classes=101,
)

base_out = EfficientNetB3.output
```

```python
x = tf.keras.layers.GlobalAveragePooling2D()(base_out)
x = layers.Dense(256, activation='relu',kernel_regularizer=regularizers.l2(0.
↪001))(x)
x = layers.Dropout(0.5)(x)
output = layers.Dense(len(class_names), activation='softmax')(x)

model_TL_complex_finetune2 = models.Model(EfficientNetB3.input, output)

# freezing the base_model:
for layer in EfficientNetB3.layers[:]:
    layer.trainable = False

from tensorflow.keras.optimizers import Adam
model_TL_complex_finetune2.compile(loss=tf.keras.losses.
↪SparseCategoricalCrossentropy(),
                 optimizer="adam",
                 metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])

history = model_TL_complex_finetune2.fit(train_ds,
                         validation_data=val_ds,
                         epochs=4,
                         verbose=1)
# unfreezing the base model but keeping the first 380 weights frozen:

for layer in EfficientNetB3.layers[:380]:
    layer.trainable = False

for layer in EfficientNetB3.layers[380:]:
    layer.trainable = True

model_TL_complex_finetune2.compile(loss=tf.keras.losses.
↪SparseCategoricalCrossentropy(),
                 optimizer=Adam(
                learning_rate=0.0002,
                beta_1=0.9,
                beta_2=0.999,
                epsilon=1e-08 ),
                 metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])

history = model_TL_complex_finetune2.fit(train_ds,
                         validation_data=val_ds,
                         epochs=20,
                         callbacks=[early_stopping, plateau],
                         verbose=1)
```

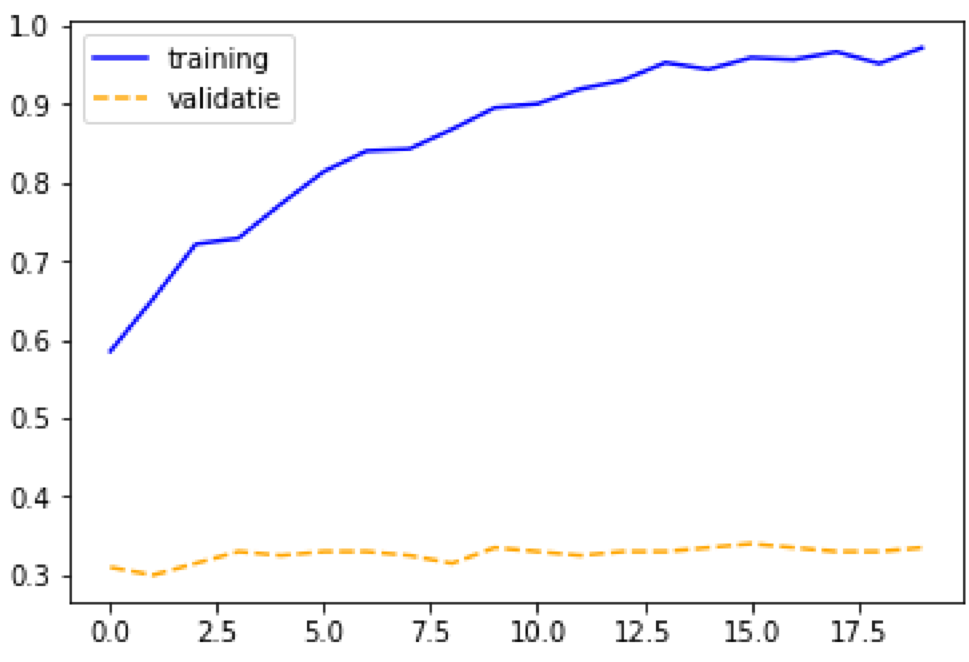
```
Epoch 1/4
25/25 [==============================] - 14s 243ms/step - loss: 5.0753 -
```

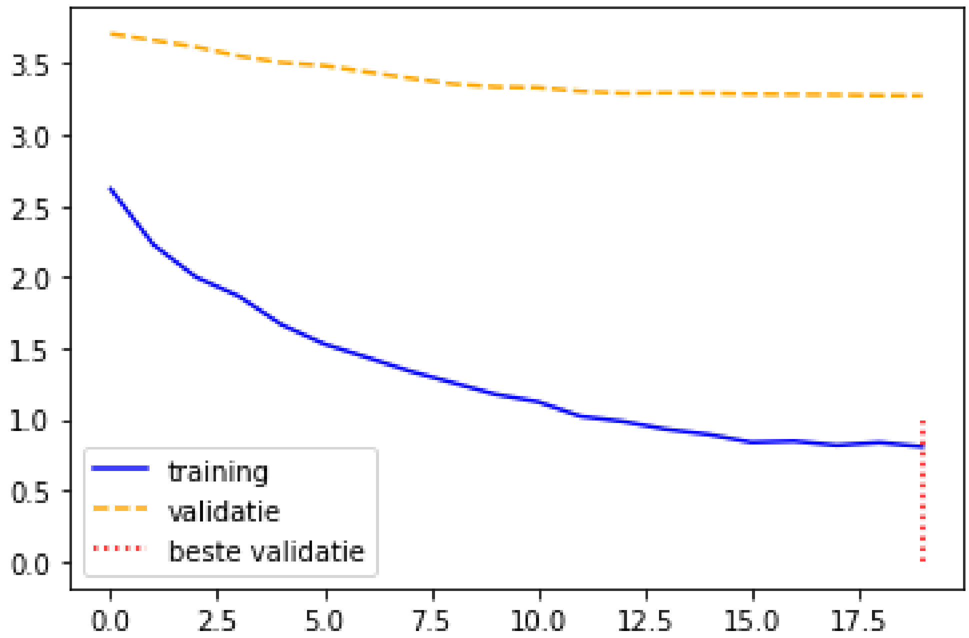
```
sparse_categorical_accuracy: 0.0188 - val_loss: 4.8029 -
val_sparse_categorical_accuracy: 0.1050
Epoch 2/4
25/25 [==============================] - 4s 156ms/step - loss: 4.2892 -
sparse_categorical_accuracy: 0.2100 - val_loss: 4.4132 -
val_sparse_categorical_accuracy: 0.2050
Epoch 3/4
25/25 [==============================] - 4s 157ms/step - loss: 3.5443 -
sparse_categorical_accuracy: 0.3550 - val_loss: 4.0576 -
val_sparse_categorical_accuracy: 0.2300
Epoch 4/4
25/25 [==============================] - 4s 158ms/step - loss: 2.8461 -
sparse_categorical_accuracy: 0.4950 - val_loss: 3.7444 -
val_sparse_categorical_accuracy: 0.3050
Epoch 1/20
25/25 [==============================] - 14s 247ms/step - loss: 2.6189 -
sparse_categorical_accuracy: 0.5850 - val_loss: 3.7088 -
val_sparse_categorical_accuracy: 0.3100
Epoch 2/20
25/25 [==============================] - 4s 159ms/step - loss: 2.2280 -
sparse_categorical_accuracy: 0.6513 - val_loss: 3.6636 -
val_sparse_categorical_accuracy: 0.3000
Epoch 3/20
25/25 [==============================] - 4s 160ms/step - loss: 1.9994 -
sparse_categorical_accuracy: 0.7212 - val_loss: 3.6165 -
val_sparse_categorical_accuracy: 0.3150
Epoch 4/20
25/25 [==============================] - 4s 158ms/step - loss: 1.8653 -
sparse_categorical_accuracy: 0.7287 - val_loss: 3.5529 -
val_sparse_categorical_accuracy: 0.3300
Epoch 5/20
25/25 [==============================] - 4s 160ms/step - loss: 1.6652 -
sparse_categorical_accuracy: 0.7725 - val_loss: 3.5065 -
val_sparse_categorical_accuracy: 0.3250
Epoch 6/20
25/25 [==============================] - 4s 159ms/step - loss: 1.5289 -
sparse_categorical_accuracy: 0.8138 - val_loss: 3.4852 -
val_sparse_categorical_accuracy: 0.3300
Epoch 7/20
25/25 [==============================] - 4s 160ms/step - loss: 1.4346 -
sparse_categorical_accuracy: 0.8400 - val_loss: 3.4413 -
val_sparse_categorical_accuracy: 0.3300
Epoch 8/20
25/25 [==============================] - 4s 160ms/step - loss: 1.3381 -
sparse_categorical_accuracy: 0.8425 - val_loss: 3.3984 -
val_sparse_categorical_accuracy: 0.3250
Epoch 9/20
25/25 [==============================] - 4s 157ms/step - loss: 1.2572 -
```

```
sparse_categorical_accuracy: 0.8675 - val_loss: 3.3579 -
val_sparse_categorical_accuracy: 0.3150
Epoch 10/20
25/25 [==============================] - 4s 158ms/step - loss: 1.1750 -
sparse_categorical_accuracy: 0.8950 - val_loss: 3.3371 -
val_sparse_categorical_accuracy: 0.3350
Epoch 11/20
25/25 [==============================] - 4s 156ms/step - loss: 1.1228 -
sparse_categorical_accuracy: 0.9000 - val_loss: 3.3318 -
val_sparse_categorical_accuracy: 0.3300
Epoch 12/20
25/25 [==============================] - 4s 157ms/step - loss: 1.0213 -
sparse_categorical_accuracy: 0.9187 - val_loss: 3.3063 -
val_sparse_categorical_accuracy: 0.3250
Epoch 13/20
25/25 [==============================] - 4s 156ms/step - loss: 0.9848 -
sparse_categorical_accuracy: 0.9300 - val_loss: 3.2921 -
val_sparse_categorical_accuracy: 0.3300
Epoch 14/20
25/25 [==============================] - 4s 166ms/step - loss: 0.9302 -
sparse_categorical_accuracy: 0.9525 - val_loss: 3.2949 -
val_sparse_categorical_accuracy: 0.3300
Epoch 15/20
25/25 [==============================] - 4s 157ms/step - loss: 0.8933 -
sparse_categorical_accuracy: 0.9438 - val_loss: 3.2924 -
val_sparse_categorical_accuracy: 0.3350
Epoch 16/20
25/25 [==============================] - 4s 158ms/step - loss: 0.8384 -
sparse_categorical_accuracy: 0.9588 - val_loss: 3.2864 -
val_sparse_categorical_accuracy: 0.3400

Epoch 00016: ReduceLROnPlateau reducing learning rate to 3.9999998989515007e-05.
Epoch 17/20
25/25 [==============================] - 4s 157ms/step - loss: 0.8466 -
sparse_categorical_accuracy: 0.9563 - val_loss: 3.2840 -
val_sparse_categorical_accuracy: 0.3350
Epoch 18/20
25/25 [==============================] - 4s 157ms/step - loss: 0.8171 -
sparse_categorical_accuracy: 0.9663 - val_loss: 3.2819 -
val_sparse_categorical_accuracy: 0.3300
Epoch 19/20
25/25 [==============================] - 4s 159ms/step - loss: 0.8367 -
sparse_categorical_accuracy: 0.9513 - val_loss: 3.2765 -
val_sparse_categorical_accuracy: 0.3300
Epoch 20/20
25/25 [==============================] - 4s 158ms/step - loss: 0.8075 -
sparse_categorical_accuracy: 0.9712 - val_loss: 3.2753 -
val_sparse_categorical_accuracy: 0.3350
```

[33]:

```
train_loss_values = history.history['loss']
val_loss_values = history.history['val_loss']
best_val_idx = np.argmin(val_loss_values)
num_epochs = range(len(train_loss_values))

plt.plot(num_epochs, train_loss_values, label='training', color='blue', ls='-')
plt.plot(num_epochs, val_loss_values, label='validatie', color='orange',␣
↪ls='--')
plt.vlines(x=best_val_idx, ymin=0, ymax=1, label='beste validatie',␣
↪color='red', ls=':')
plt.legend()
plt.figure(0)
train_loss_values = history.history['sparse_categorical_accuracy']
val_loss_values = history.history['val_sparse_categorical_accuracy']
num_epochs = range(len(train_loss_values))

plt.plot(num_epochs, train_loss_values, label='training', color='blue', ls='-')
plt.plot(num_epochs, val_loss_values, label='validatie', color='orange',␣
↪ls='--')
plt.legend()
plt.figure(1)
plt.show()
```






[34]:
```
loss, categorical_accuracy = model_TL_complex_finetune2.evaluate(test_ds,␣
↪verbose=1)
loss, categorical_accuracy
```

```
32/32 [==============================] - 4s 117ms/step - loss: 3.2588 -
sparse_categorical_accuracy: 0.3380
```

[34]: (3.25883412361145, 0.33799999952316284)

[35]:
```
model_TL_complex_finetune2.save_weights('EfficientNetB3_complex_finetuned2.h5')
```

this approach yielded the best results untill now

#### 1.2.4 EfficientNetB3 fully training the model

[36]:
```
EfficientNetB3=tf.keras.applications.EfficientNetB3(
    include_top=False,
    weights="imagenet",
    input_shape=(img_width, img_width, 3),
    classes=101,
)
```

we will train the whole model at once

[37]:
```
EfficientNetB3.trainable=True
```

[38]:
```python
model_TL = tf.keras.Sequential([
    EfficientNetB3,
    layers.GlobalAveragePooling2D(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(101, activation='softmax'),
])
```

[39]:
```python
model_TL.compile(
    optimizer='adam',
    loss = 'sparse_categorical_crossentropy',
    metrics=['sparse_categorical_accuracy']
)

model_TL.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
efficientnetb3 (Functional)  (None, 7, 7, 1536)        10783535
_________________________________________________________________
global_average_pooling2d_3 ( (None, 1536)              0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               196736
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 101)               13029
=================================================================
Total params: 10,993,300
Trainable params: 10,905,997
Non-trainable params: 87,303
_________________________________________________________________
```

[40]:
```python
EPOCHS=20

with tf.device('/GPU:0'):
    hist = model_TL.fit(
        train_ds,
        validation_data = val_ds,
        epochs = 20,
        callbacks=[early_stopping, plateau],
    )
```

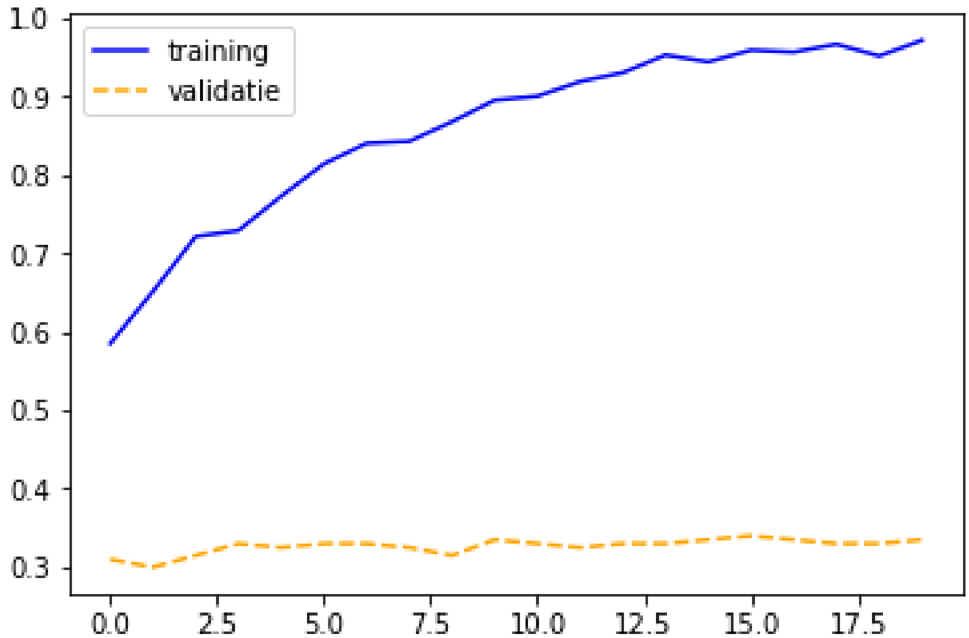
```
Epoch 1/20
25/25 [==============================] - 32s 723ms/step - loss: 4.6415 -
sparse_categorical_accuracy: 0.0175 - val_loss: 4.5477 -
```

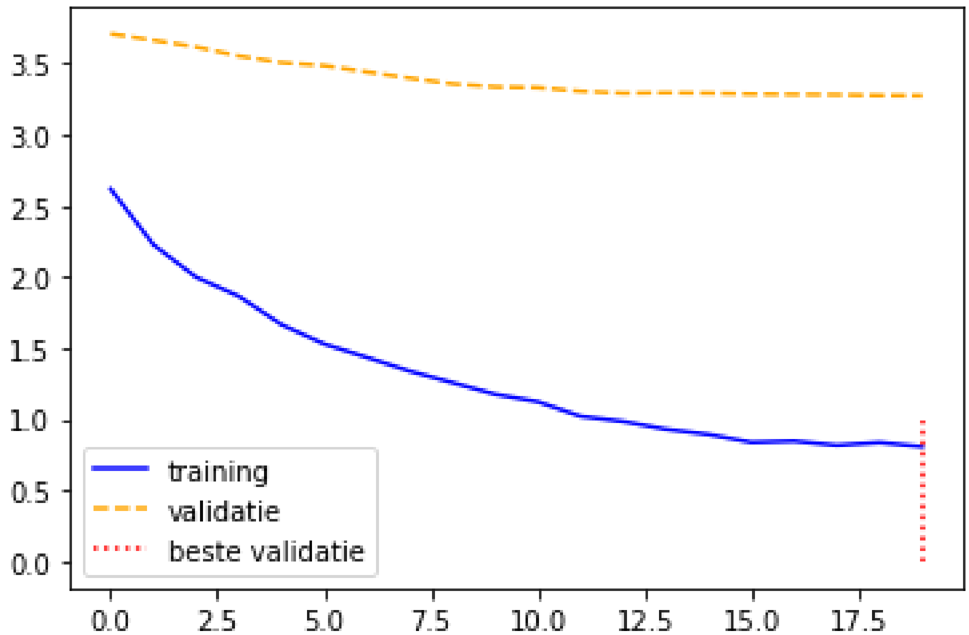
```
val_sparse_categorical_accuracy: 0.0450
Epoch 2/20
25/25 [==============================] - 16s 653ms/step - loss: 3.8647 -
sparse_categorical_accuracy: 0.1700 - val_loss: 4.4002 -
val_sparse_categorical_accuracy: 0.1150
Epoch 3/20
25/25 [==============================] - 16s 639ms/step - loss: 2.8178 -
sparse_categorical_accuracy: 0.3800 - val_loss: 4.8937 -
val_sparse_categorical_accuracy: 0.0950
Epoch 4/20
25/25 [==============================] - 16s 632ms/step - loss: 1.8425 -
sparse_categorical_accuracy: 0.5863 - val_loss: 4.5638 -
val_sparse_categorical_accuracy: 0.1850
Epoch 5/20
25/25 [==============================] - 16s 635ms/step - loss: 1.0952 -
sparse_categorical_accuracy: 0.7500 - val_loss: 4.7659 -
val_sparse_categorical_accuracy: 0.1850

Epoch 00005: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
Epoch 6/20
25/25 [==============================] - 16s 634ms/step - loss: 0.6136 -
sparse_categorical_accuracy: 0.8700 - val_loss: 4.3335 -
val_sparse_categorical_accuracy: 0.2050
Epoch 7/20
25/25 [==============================] - 16s 628ms/step - loss: 0.4041 -
sparse_categorical_accuracy: 0.9350 - val_loss: 4.1228 -
val_sparse_categorical_accuracy: 0.2100
Epoch 8/20
25/25 [==============================] - 16s 634ms/step - loss: 0.2993 -
sparse_categorical_accuracy: 0.9463 - val_loss: 4.0093 -
val_sparse_categorical_accuracy: 0.2200
Epoch 9/20
25/25 [==============================] - 16s 635ms/step - loss: 0.2281 -
sparse_categorical_accuracy: 0.9712 - val_loss: 3.9615 -
val_sparse_categorical_accuracy: 0.2300
Epoch 10/20
25/25 [==============================] - 16s 643ms/step - loss: 0.2170 -
sparse_categorical_accuracy: 0.9650 - val_loss: 3.9362 -
val_sparse_categorical_accuracy: 0.2450
Epoch 11/20
25/25 [==============================] - 16s 635ms/step - loss: 0.1552 -
sparse_categorical_accuracy: 0.9837 - val_loss: 3.9174 -
val_sparse_categorical_accuracy: 0.2450
Epoch 12/20
25/25 [==============================] - 16s 637ms/step - loss: 0.1513 -
sparse_categorical_accuracy: 0.9787 - val_loss: 3.9161 -
val_sparse_categorical_accuracy: 0.2300
Epoch 13/20
```

```
25/25 [==============================] - 16s 633ms/step - loss: 0.1220 -
sparse_categorical_accuracy: 0.9862 - val_loss: 3.9258 -
val_sparse_categorical_accuracy: 0.2400
Epoch 14/20
25/25 [==============================] - 16s 638ms/step - loss: 0.1269 -
sparse_categorical_accuracy: 0.9850 - val_loss: 3.9141 -
val_sparse_categorical_accuracy: 0.2400

Epoch 00014: ReduceLROnPlateau reducing learning rate to 4.0000001899898055e-05.
Epoch 15/20
25/25 [==============================] - 16s 634ms/step - loss: 0.0881 -
sparse_categorical_accuracy: 0.9975 - val_loss: 3.9034 -
val_sparse_categorical_accuracy: 0.2400
Epoch 16/20
25/25 [==============================] - 16s 630ms/step - loss: 0.1001 -
sparse_categorical_accuracy: 0.9937 - val_loss: 3.8981 -
val_sparse_categorical_accuracy: 0.2450
Epoch 17/20
25/25 [==============================] - 16s 638ms/step - loss: 0.0957 -
sparse_categorical_accuracy: 0.9875 - val_loss: 3.8916 -
val_sparse_categorical_accuracy: 0.2450
Epoch 18/20
25/25 [==============================] - 16s 640ms/step - loss: 0.0872 -
sparse_categorical_accuracy: 0.9925 - val_loss: 3.8923 -
val_sparse_categorical_accuracy: 0.2450
Epoch 19/20
25/25 [==============================] - 16s 630ms/step - loss: 0.0765 -
sparse_categorical_accuracy: 0.9937 - val_loss: 3.8931 -
val_sparse_categorical_accuracy: 0.2400
Epoch 20/20
25/25 [==============================] - 16s 637ms/step - loss: 0.0910 -
sparse_categorical_accuracy: 0.9900 - val_loss: 3.8984 -
val_sparse_categorical_accuracy: 0.2400

Epoch 00020: ReduceLROnPlateau reducing learning rate to 8.000000525498762e-06.
```

[41]:
```python
train_loss_values = history.history['loss']
val_loss_values = history.history['val_loss']
best_val_idx = np.argmin(val_loss_values)
num_epochs = range(len(train_loss_values))

plt.plot(num_epochs, train_loss_values, label='training', color='blue', ls='-')
plt.plot(num_epochs, val_loss_values, label='validatie', color='orange',
    ls='--')
plt.vlines(x=best_val_idx, ymin=0, ymax=1, label='beste validatie',
    color='red', ls=':')
plt.legend()
```

```python
plt.figure(0)
train_loss_values = history.history['sparse_categorical_accuracy']
val_loss_values = history.history['val_sparse_categorical_accuracy']
num_epochs = range(len(train_loss_values))

plt.plot(num_epochs, train_loss_values, label='training', color='blue', ls='-')
plt.plot(num_epochs, val_loss_values, label='validatie', color='orange',␣
 ↪ls='--')
plt.legend()
plt.figure(1)
plt.show()
```






[42]:
```
loss, categorical_accuracy = model_TL.evaluate(test_ds, verbose=1)
loss, categorical_accuracy
```

```
32/32 [==============================] - 4s 115ms/step - loss: 4.0439 -
sparse_categorical_accuracy: 0.2010
```

[42]:
```
(4.043946266174316, 0.20100000500679016)
```

[43]:
```
model_TL.save_weights('EfficientNetB3_full_train.h5')
```

this is what we already had expected, by training the whole model at once we are actually not using it on it's full potential. Because our source domain is different from our destination domain we are ruining the pretrained weights. We can train our model longer and might get better results but is not our purpose. We want to use a pretrained model so our training time is shorter.

until now we only used the HDF5 files which provided us with 1000 training images. From the experiments above we learned that the best method was **EfficientNetB3 with different fine tuning approach**. We will now use that same approach but on a 80% of the whole dataset and use the last 20% as test images

We won't use the HDF5 files because the images available in them are cropped instead of squeezed, this makes the food very hard to recognize (even for humans like me). That's why we decided to use the original images and resize them ourselves

#### 1.2.5 loading our data from directory

[ ]:
```
dataset_root_path=Path("/project_ghent/raman/project/food41")
```

[ ]:
```
data_dir=dataset_root_path/"train"
test_dir=dataset_root_path/"test"
```

[ ]:
```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=47,
  image_size=(img_height, img_width),
  batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=47,
  image_size=(img_height, img_width),
  batch_size=batch_size)
test_ds = tf.keras.utils.image_dataset_from_directory(
  test_dir,
  seed=47,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 75750 files belonging to 101 classes.
Using 60600 files for training.
```

```
2022-12-16 20:11:16.805643: 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:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2022-12-16 20:11:19.554186: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1616] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 10041 MB memory:  -> device:
0, name: NVIDIA GeForce GTX 1080 Ti, pci bus id: 0000:03:00.0, compute
capability: 6.1
```

```
Found 75750 files belonging to 101 classes.
Using 15150 files for validation.
Found 25250 files belonging to 101 classes.
```

[ ]:
```python
def prepare_dataset(ds, batch_size=32, b_shuffle=True,augment=True):
    # transform input data into tf.data

    ds = ds.map(map_func = preprocessing ,num_parallel_calls = tf.data.
 ↪experimental.AUTOTUNE)
    # normally you only need to shuffle the training data
    if b_shuffle == True:
        ds = ds.shuffle(255)
    # normally you only need to shuffle the training data
    if augment:
        ds = ds.map(lambda x, y: (data_augmentation(x, training=True),␣
 ↪y),num_parallel_calls=tf.data.experimental.AUTOTUNE)
    # ds = ds.batch(batch_size)
    # ds = ds.cache()
    ds = ds.prefetch(buffer_size = 255)
    return ds



def preprocessing(image, label):
    image = resize(image)
    return image, label


train_ds = prepare_dataset(train_ds,augment=True)
val_ds = prepare_dataset(val_ds, b_shuffle = False,augment=False)
test_ds = prepare_dataset(test_ds, b_shuffle = False,augment=False)
```

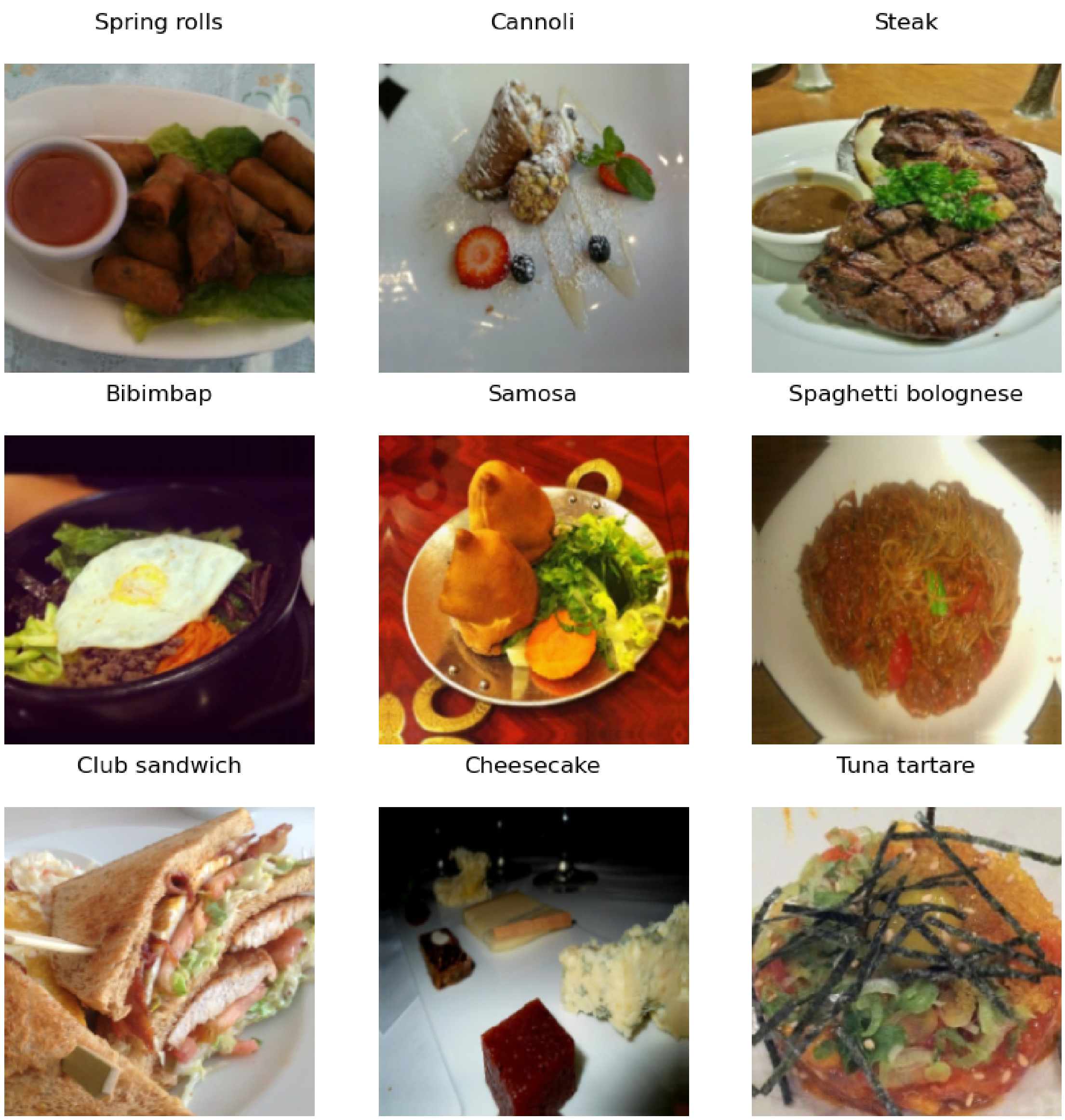
```
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there
is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there
is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3
```

```
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there
is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting
StatelessRandomUniformFullIntV2 cause there is no registered converter for this
op.
WARNING:tensorflow:Using a while_loop for converting
StatelessRandomGetKeyCounter cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting AdjustContrastv2 cause
Input "contrast_factor" of op 'AdjustContrastv2' expected to be loop invariant.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there
is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there
is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there
is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no
registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting
StatelessRandomUniformFullIntV2 cause there is no registered converter for this
op.
WARNING:tensorflow:Using a while_loop for converting
StatelessRandomGetKeyCounter cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2
```

```
cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting AdjustContrastv2 cause
Input "contrast_factor" of op 'AdjustContrastv2' expected to be loop invariant.
```

[ ]:

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))

for images, labels in train_ds.take(9):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```



as we can see here, the resize strategy was squeeze instead of crop, we can still recognise the different food classes

### 1.2.6 EfficientNetB3 different fine-tuning approach and using full dataset

[ ]:
```
EfficientNetB3=tf.keras.applications.EfficientNetB3(
    include_top=False,
    weights="imagenet",
    input_shape=(img_width, img_width, 3),
    pooling=None,
    classes=101,
)

base_out = EfficientNetB3.output
x = tf.keras.layers.GlobalAveragePooling2D()(base_out)
x = layers.Dense(256, activation='relu',kernel_regularizer=regularizers.l2(0.
 ↪001))(x)
x = layers.Dropout(0.5)(x)
output = layers.Dense(len(class_names), activation='softmax')(x)

model_TL_complex_finetune_full = models.Model(EfficientNetB3.input, output)

# freezing the base_model:
for layer in EfficientNetB3.layers[:]:
    layer.trainable = False

from tensorflow.keras.optimizers import Adam
model_TL_complex_finetune_full.compile(loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(),
                 optimizer="adam",
                 metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])

history = model_TL_complex_finetune_full.fit(train_ds,
                         validation_data=val_ds,
                         epochs=4,
                         verbose=1)
# unfreezing the base model but keeping the first 380 weights frozen:

for layer in EfficientNetB3.layers[:380]:
    layer.trainable = False

for layer in EfficientNetB3.layers[380:]:
    layer.trainable = True
```

```
model_TL_complex_finetune_full.compile(loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(),
                 optimizer=Adam(
                learning_rate=0.0002,
                beta_1=0.9,
                beta_2=0.999,
                epsilon=1e-08 ),
                 metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])

history = model_TL_complex_finetune_full.fit(train_ds,
                         validation_data=val_ds,
                         epochs=20,
                         callbacks=[early_stopping, plateau],
                         verbose=1)
```

```
Epoch 1/4
```

```
2022-12-16 20:14:57.381678: I tensorflow/stream_executor/cuda/cuda_dnn.cc:384]
Loaded cuDNN version 8500
```

```
1894/1894 [==============================] - 478s 238ms/step - loss: 2.8847 -
sparse_categorical_accuracy: 0.3930 - val_loss: 2.0496 -
val_sparse_categorical_accuracy: 0.5786
Epoch 2/4
1894/1894 [==============================] - 457s 236ms/step - loss: 2.4611 -
sparse_categorical_accuracy: 0.4744 - val_loss: 1.9922 -
val_sparse_categorical_accuracy: 0.5861
Epoch 3/4
1894/1894 [==============================] - 456s 235ms/step - loss: 2.4193 -
sparse_categorical_accuracy: 0.4884 - val_loss: 1.9879 -
val_sparse_categorical_accuracy: 0.5896
Epoch 4/4
1894/1894 [==============================] - 457s 236ms/step - loss: 2.3987 -
sparse_categorical_accuracy: 0.4913 - val_loss: 1.9858 -
val_sparse_categorical_accuracy: 0.5929
Epoch 1/20
1894/1894 [==============================] - 476s 239ms/step - loss: 2.1633 -
sparse_categorical_accuracy: 0.5386 - val_loss: 1.7121 -
val_sparse_categorical_accuracy: 0.6389 - lr: 2.0000e-04
Epoch 2/20
1894/1894 [==============================] - 458s 236ms/step - loss: 1.9244 -
sparse_categorical_accuracy: 0.5762 - val_loss: 1.5963 -
val_sparse_categorical_accuracy: 0.6560 - lr: 2.0000e-04
Epoch 3/20
1894/1894 [==============================] - 462s 238ms/step - loss: 1.8031 -
sparse_categorical_accuracy: 0.5967 - val_loss: 1.5378 -
val_sparse_categorical_accuracy: 0.6625 - lr: 2.0000e-04
Epoch 4/20
```

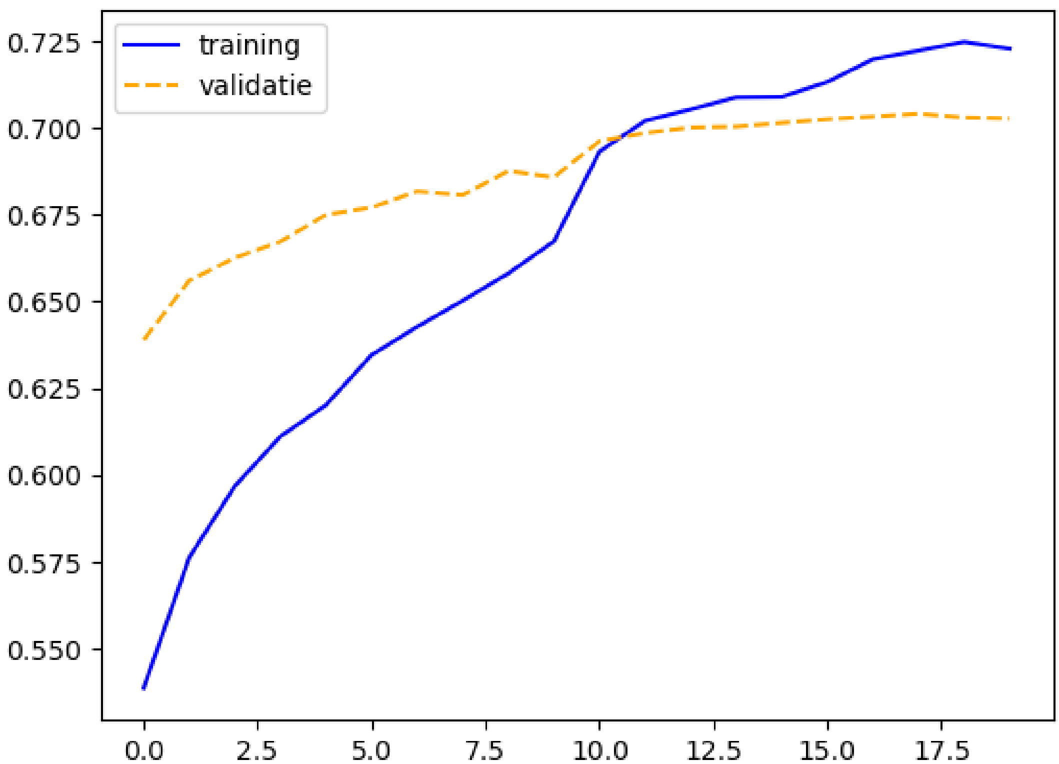
```
1894/1894 [==============================] - 457s 236ms/step - loss: 1.7142 -
sparse_categorical_accuracy: 0.6111 - val_loss: 1.4936 -
val_sparse_categorical_accuracy: 0.6672 - lr: 2.0000e-04
Epoch 5/20
1894/1894 [==============================] - 458s 236ms/step - loss: 1.6507 -
sparse_categorical_accuracy: 0.6201 - val_loss: 1.4626 -
val_sparse_categorical_accuracy: 0.6749 - lr: 2.0000e-04
Epoch 6/20
1894/1894 [==============================] - 456s 236ms/step - loss: 1.5878 -
sparse_categorical_accuracy: 0.6345 - val_loss: 1.4365 -
val_sparse_categorical_accuracy: 0.6770 - lr: 2.0000e-04
Epoch 7/20
1894/1894 [==============================] - 457s 236ms/step - loss: 1.5385 -
sparse_categorical_accuracy: 0.6426 - val_loss: 1.4143 -
val_sparse_categorical_accuracy: 0.6817 - lr: 2.0000e-04
Epoch 8/20
1894/1894 [==============================] - 459s 237ms/step - loss: 1.5026 -
sparse_categorical_accuracy: 0.6502 - val_loss: 1.4003 -
val_sparse_categorical_accuracy: 0.6806 - lr: 2.0000e-04
Epoch 9/20
1894/1894 [==============================] - 457s 236ms/step - loss: 1.4608 -
sparse_categorical_accuracy: 0.6580 - val_loss: 1.3944 -
val_sparse_categorical_accuracy: 0.6875 - lr: 2.0000e-04
Epoch 10/20
1894/1894 [==============================] - ETA: 0s - loss: 1.4356 -
sparse_categorical_accuracy: 0.6672
Epoch 10: ReduceLROnPlateau reducing learning rate to 3.9999998989515007e-05.
1894/1894 [==============================] - 459s 237ms/step - loss: 1.4356 -
sparse_categorical_accuracy: 0.6672 - val_loss: 1.3932 -
val_sparse_categorical_accuracy: 0.6857 - lr: 2.0000e-04
Epoch 11/20
1894/1894 [==============================] - 456s 236ms/step - loss: 1.3234 -
sparse_categorical_accuracy: 0.6931 - val_loss: 1.3486 -
val_sparse_categorical_accuracy: 0.6961 - lr: 4.0000e-05
Epoch 12/20
1894/1894 [==============================] - 458s 237ms/step - loss: 1.2896 -
sparse_categorical_accuracy: 0.7020 - val_loss: 1.3380 -
val_sparse_categorical_accuracy: 0.6985 - lr: 4.0000e-05
Epoch 13/20
1894/1894 [==============================] - 459s 237ms/step - loss: 1.2733 -
sparse_categorical_accuracy: 0.7053 - val_loss: 1.3311 -
val_sparse_categorical_accuracy: 0.7001 - lr: 4.0000e-05
Epoch 14/20
```

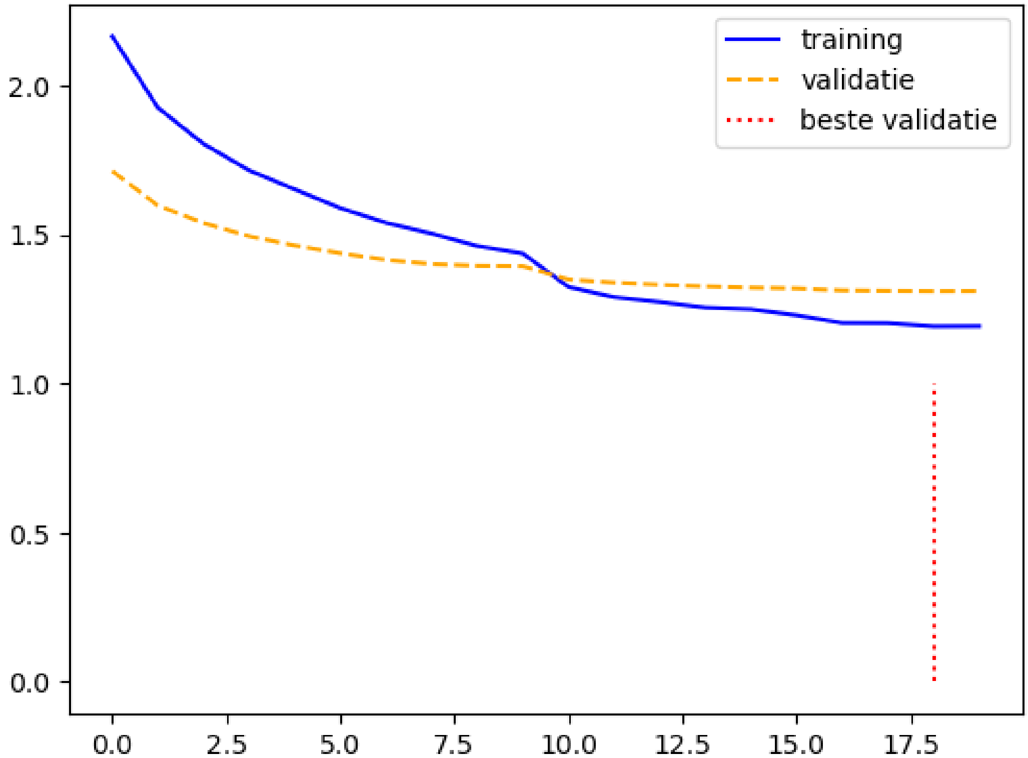
```
2022-12-16 22:25:23.872810: I
tensorflow/core/kernels/data/shuffle_dataset_op.cc:390] Filling up shuffle
buffer (this may take a while): 227 of 255
2022-12-16 22:25:25.034836: I
```

```
tensorflow/core/kernels/data/shuffle_dataset_op.cc:415] Shuffle buffer filled.
1894/1894 [==============================] - 463s 238ms/step - loss: 1.2544 -
sparse_categorical_accuracy: 0.7088 - val_loss: 1.3255 -
val_sparse_categorical_accuracy: 0.7003 - lr: 4.0000e-05
Epoch 15/20
1894/1894 [==============================] - 459s 237ms/step - loss: 1.2489 -
sparse_categorical_accuracy: 0.7089 - val_loss: 1.3211 -
val_sparse_categorical_accuracy: 0.7014 - lr: 4.0000e-05
Epoch 16/20
1894/1894 [==============================] - ETA: 0s - loss: 1.2286 -
sparse_categorical_accuracy: 0.7132
Epoch 16: ReduceLROnPlateau reducing learning rate to 7.999999797903002e-06.
1894/1894 [==============================] - 466s 241ms/step - loss: 1.2286 -
sparse_categorical_accuracy: 0.7132 - val_loss: 1.3185 -
val_sparse_categorical_accuracy: 0.7024 - lr: 4.0000e-05
Epoch 17/20
1894/1894 [==============================] - 462s 238ms/step - loss: 1.2029 -
sparse_categorical_accuracy: 0.7197 - val_loss: 1.3118 -
val_sparse_categorical_accuracy: 0.7032 - lr: 8.0000e-06
Epoch 18/20
1894/1894 [==============================] - 458s 236ms/step - loss: 1.2024 -
sparse_categorical_accuracy: 0.7222 - val_loss: 1.3104 -
val_sparse_categorical_accuracy: 0.7040 - lr: 8.0000e-06
Epoch 19/20
1894/1894 [==============================] - ETA: 0s - loss: 1.1915 -
sparse_categorical_accuracy: 0.7247
Epoch 19: ReduceLROnPlateau reducing learning rate to 1.5999999959603884e-06.
1894/1894 [==============================] - 458s 236ms/step - loss: 1.1915 -
sparse_categorical_accuracy: 0.7247 - val_loss: 1.3095 -
val_sparse_categorical_accuracy: 0.7029 - lr: 8.0000e-06
Epoch 20/20
1894/1894 [==============================] - 460s 238ms/step - loss: 1.1920 -
sparse_categorical_accuracy: 0.7227 - val_loss: 1.3099 -
val_sparse_categorical_accuracy: 0.7026 - lr: 1.6000e-06
```

[ ]:
```
train_loss_values = history.history['loss']
val_loss_values = history.history['val_loss']
best_val_idx = np.argmin(val_loss_values)
num_epochs = range(len(train_loss_values))

plt.plot(num_epochs, train_loss_values, label='training', color='blue', ls='-')
plt.plot(num_epochs, val_loss_values, label='validatie', color='orange',␣
 ↪ls='--')
plt.vlines(x=best_val_idx, ymin=0, ymax=1, label='beste validatie',␣
 ↪color='red', ls=':')
plt.legend()
plt.figure(0)
```

```
train_loss_values = history.history['sparse_categorical_accuracy']
val_loss_values = history.history['val_sparse_categorical_accuracy']
num_epochs = range(len(train_loss_values))

plt.plot(num_epochs, train_loss_values, label='training', color='blue', ls='-')
plt.plot(num_epochs, val_loss_values, label='validatie', color='orange',␣
 ↪ls='--')
plt.legend()
plt.figure(1)
plt.show()
```






[4]:
```
loss, categorical_accuracy = model_TL_complex_finetune_full.evaluate(test_ds,␣
    ↪verbose=1)
loss, categorical_accuracy
```

```
790/790 [==============================] - 62s 78ms/step - loss: 1.0837 -
sparse_categorical_accuracy: 0.7479
(1.0836948156356812, 0.7478811740875244)
```

[ ]:
```
model_TL_complex_finetune_full.
    ↪save_weights('EfficientNetB3_complex_finetuned_full.h5')
```

### 1.2.7 feature map visualisation

like we were told in our lab

```
Visualizing the feature maps of a trained model helps us to better interpret the results and un
```

[56]:
```
import matplotlib.cm as cm
from IPython.display import Image, display

def get_img_array(img_path, size):
    # `img` is a PIL image of size 299x299
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=size)
```

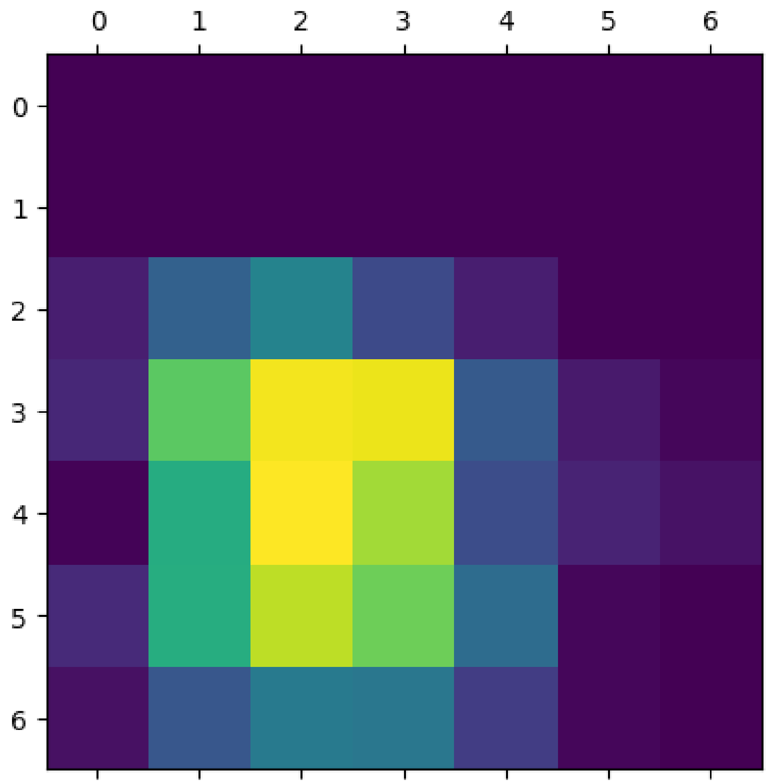
```
    # `array` is a float32 Numpy array of shape (299, 299, 3)
    array = tf.keras.preprocessing.image.img_to_array(img)
    # We add a dimension to transform our array into a "batch"
    # of size (1, 299, 299, 3)
    array = np.expand_dims(array, axis=0)
    return array


def make_gradcam_heatmap(img_array, model, last_conv_layer_name,␣
↪pred_index=None):
    # First, we create a model that maps the input image to the activations
    # of the last conv layer as well as the output predictions
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output, model.
↪output]
    )

    # Then, we compute the gradient of the top predicted class for our input␣
↪image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    # This is the gradient of the output neuron (top predicted or chosen)
    # with regard to the output feature map of the last conv layer
    grads = tape.gradient(class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the top predicted class
    # then sum all the channels to obtain the heatmap class activation
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)

    # For visualization purpose, we will also normalize the heatmap between 0 &␣
↪1
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()
def save_and_display_gradcam(img_path, heatmap, cam_path="cam.jpg", alpha=0.4):
    # Load the original image
```

```
    img = tf.keras.preprocessing.image.load_img(img_path)
    img = tf.keras.preprocessing.image.img_to_array(img)

    # Rescale heatmap to a range 0-255
    heatmap = np.uint8(255 * heatmap)

    # Use jet colormap to colorize heatmap
    jet = cm.get_cmap("jet")

    # Use RGB values of the colormap
    jet_colors = jet(np.arange(256))[:, :3]
    jet_heatmap = jet_colors[heatmap]

    # Create an image with RGB colorized heatmap
    jet_heatmap = tf.keras.preprocessing.image.array_to_img(jet_heatmap)
    jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
    jet_heatmap = tf.keras.preprocessing.image.img_to_array(jet_heatmap)

    # Superimpose the heatmap on original image
    superimposed_img = jet_heatmap * alpha + img
    superimposed_img = tf.keras.preprocessing.image.
 ↪array_to_img(superimposed_img)

    # Save the superimposed image
    superimposed_img.save(cam_path)

    # Display Grad CAM
    display(Image(cam_path))
```

[ ]:
```
# Prepare image
img_size=(img_height,img_width,3)
img_path="/project_ghent/raman/project/food41/test/chocolate_mousse/1379570.jpg"
img_array = get_img_array(img_path, size=img_size)



# Print what the top predicted class is
preds = model_TL_complex_finetune_full.predict(img_array)
print("Predicted:", class_names[np.argmax(preds)])

# Generate class activation heatmap
heatmap = make_gradcam_heatmap(img_array, model_TL_complex_finetune_full,␣
 ↪"top_conv")

# Display heatmap
plt.matshow(heatmap)
plt.show()
```

```
save_and_display_gradcam(img_path, heatmap)
```

```
1/1 [==============================] - 0s 395ms/step
Predicted: Chocolate mousse
```



we can see how the correct region in our image is activated, our model does not look at the background but focusses on the food, the prediction is also correct. We will now look at some other examples and see if this wasn't just pure luck

[ ]:
```
# Prepare image
img_size=(img_height,img_width,3)
img_path="/project_ghent/raman/project/food41/test/dumplings/146377.jpg"
img_array = get_img_array(img_path, size=img_size)



# Print what the top predicted class is
preds = model_TL_complex_finetune_full.predict(img_array)
print("Predicted:", class_names[np.argmax(preds)])

# Generate class activation heatmap
heatmap = make_gradcam_heatmap(img_array, model_TL_complex_finetune_full,␣
 ↪"top_conv")

# Display heatmap
# plt.matshow(heatmap)
# plt.show()
save_and_display_gradcam(img_path, heatmap)
```

```
1/1 [==============================] - 0s 289ms/step
Predicted: Dumplings
```

again, the correct prediction and our corner regions that don't have food in them are not activated, now let's take a more challenging picture

```
[ ]: # Prepare image
img_size=(img_height,img_width,3)
img_path="/project_ghent/raman/project/food41/test/lobster_bisque/1889467.jpg"
img_array = get_img_array(img_path, size=img_size)



# Print what the top predicted class is
preds = model_TL_complex_finetune_full.predict(img_array)
print("Predicted:", class_names[np.argmax(preds)])

# Generate class activation heatmap
heatmap = make_gradcam_heatmap(img_array, model_TL_complex_finetune_full,␣
↪"top_conv")
```

```python
# Display heatmap
# plt.matshow(heatmap)
# plt.show()
save_and_display_gradcam(img_path, heatmap)
```

```
1/1 [==============================] - 0s 329ms/step
Predicted: French onion soup
```

our model is looking at the wrong region in the image, that's also the reason wh the prediction is wrong, the model is more focusen on the person holding the spoon, eventhough I would also count this as "French onion soup" but maybe a thick version ;)

So what was quite interesting about these feature maps is how these can actually be used to uncover a little bit about what the model was focussing on to make certain predictions. For example if I find multiple pictures where the model is predicting the wrong food and in the feature map I see a man holding a spoon and the model focussing on that part, That would give me a sign that I have to focus training my model on those kinds of pictures. So we can actually get a lot of information out of these feature maps when used correctly

### 1.3 using our model to make recommendations

[2]:
```
EfficientNetB3=tf.keras.applications.EfficientNetB3(
    include_top=False,
    weights="imagenet",
    input_shape=(img_width, img_width, 3),
    pooling=None,
    classes=101,
)

base_out = EfficientNetB3.output
x = tf.keras.layers.GlobalAveragePooling2D()(base_out)
x = layers.Dense(256, activation='relu',kernel_regularizer=regularizers.l2(0.
 ↪001))(x)
x = layers.Dropout(0.5)(x)
output = layers.Dense(101, activation='softmax')(x)

model_TL_complex_finetune_full = tf.keras.models.Model(EfficientNetB3.input,␣
 ↪output)
```

[3]:
```
model_TL_complex_finetune_full.load_weights("./
 ↪EfficientNetB3_complex_finetuned_full.h5")
```

we have to remember that our model has a rescaling layer included, the only preprocssing we have to do is resize our images.

We've also just realized we've worked with 2 different shapes, which is conflicting with using the different models. To not do the training again we do a trick

[39]:
```
# img_folder = "D:/industrieel_ingenieur/4de_jaar/ML/labo/lab1/
 ↪tripadvisor_dataset/tripadvisor_images_small"
img_height = 224
img_width =224

img_folder = "../tripadvisor_dataset/tripadvisor_images_small"
def create_dataset(img_folder, n=None):
    # n = amount of images
    image_files=os.listdir(os.path.join(img_folder))
    if n==None:
        n=len(image_files)
    images = np.zeros((n, img_height, img_width, 3))
    for i,file in enumerate(image_files[:n]):
        img=PILImage.create(os.path.join(img_folder,file))
        img_resized=img.resize((img_width,img_height))
        images[i]=img_resized
    return images

images_224 = create_dataset(img_folder,1000)
```

[40]:
```
WIDTH = 128
HEIGHT = 128
def create_dataset(img_folder, n=None):
    # n = amount of images
    image_files=os.listdir(os.path.join(img_folder))
    if n==None:
        n=len(image_files)
    images = np.zeros((n, HEIGHT, WIDTH, 3))
    for i,file in enumerate(image_files[:n]):
        img=PILImage.create(os.path.join(img_folder,file))
        img_resized=img.resize((WIDTH,HEIGHT))
        images[i]=img_resized
    return images

images_128 = create_dataset(img_folder,1000)
```

Only allowing food images. We apply the model ( from here: differentiating-buildings-from-food-cnn.ipynb )on the images_128 and apply the results on images_224

[59]:
```
import keras
model = keras.models.load_model("../results")
results = model.predict(images_128)

indices = np.argwhere(results < 0.5)
indices = indices[..., 0]

images_224 = images_224[indices]
```

[42]:
```
preds=model_TL_complex_finetune_full.predict(images_224)
```

```
32/32 [==============================] - 156s 5s/step
```

[43]:
```
class_names = []
fo = open("./labels_food101.txt")
for line in fo:
    class_names.append(line)
fo.close()
```

[44]:
```
preds=np.argmax(preds,axis=1)
```

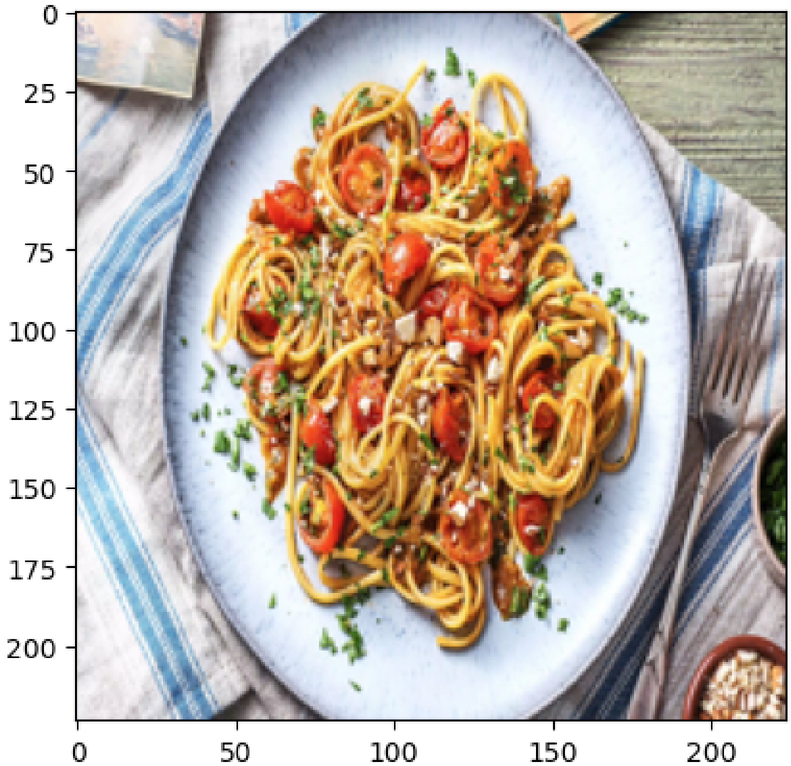
We predict in which class our pasta image is classified

[45]:
```
img = PILImage.create('../pasta.png')
img_resized = img.resize((img_height,img_width))
img_resized = np.array(img_resized)
img_resized_model = img_resized.reshape(1, img_height, img_width, 3)
img_resized_model.shape
```

[45]: (1, 224, 224, 3)

[47]:
```
plt.imshow(img_resized)
```

[47]: <matplotlib.image.AxesImage at 0x24f9b55c190>




[48]:
```
pred_image=model_TL_complex_finetune_full.predict(img_resized_model)
```

```
1/1 [==============================] - 0s 489ms/step
```

[49]:
```
pred_image=np.argmax(pred_image,axis=1)
pred_image
```

[49]: array([90], dtype=int64)

[50]:
```
print("The input image is classified as", class_names[pred_image[0]])
```

```
The input image is classified as Spaghetti bolognese
```

Getting other images classified as Spaghetti bolognese and the restaurants of that image

[52]:
```
image_files=os.listdir(img_folder)
indices = np.where(preds == pred_image[0])[0]
print(len(indices))
file_names = [image_files[i] for i in list(indices)]
unique_restaurants = set()
for file in file_names:
    unique_restaurants.add(int(file.split("_")[0]))
unique_restaurants
```

```
5
```

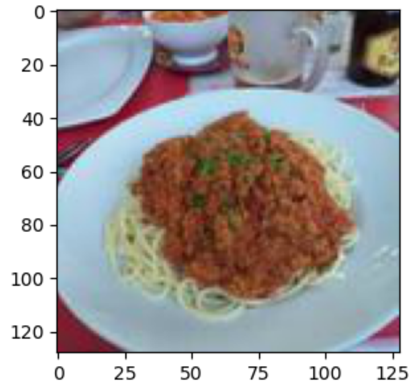
[52]:
```
{1072034, 10157303, 10501019, 10554019, 10731148}
```

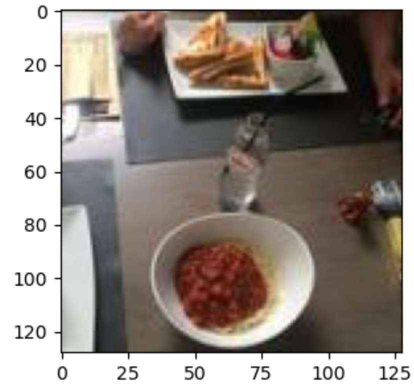
[61]:
```
original_df = pd.read_csv("../tripadvisor_dataset/restaurant_listings.csv")
related_restaurants = original_df[original_df.id.isin(unique_restaurants)]
related_restaurants["restaurant name"]
```

[61]:
```
417        't Ateljeeken
429     Giardino Di Roma
595     Bistrot Ma Tu Vu
1602        Cafe Sisaket
2052       Hedera Deinze
Name: restaurant name, dtype: object
```

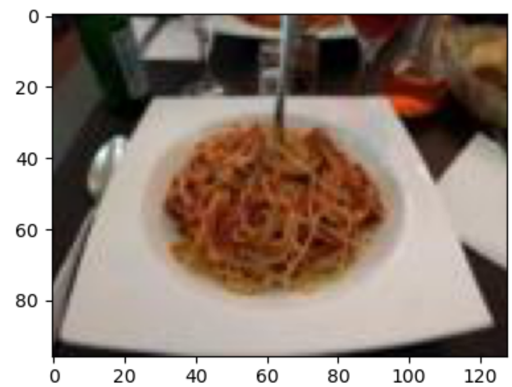
these will be the recommended restaurants

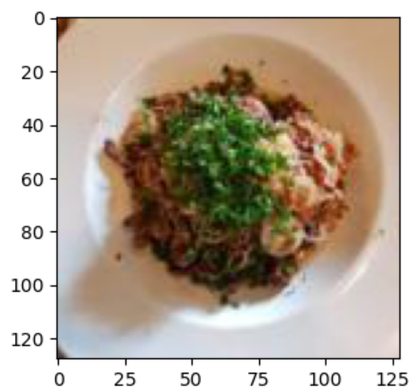
[54]:
```
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(10,15))

for i in range(0,len(file_names)):
    if i > 10:
        break
    plt.subplot(5,2,i+1)
    img = PILImage.create(img_folder + '/' + file_names[i])
    # restaurant name get be obtained in file_names[i]
    plt.imshow(img)
fig.tight_layout()
plt.show()
```







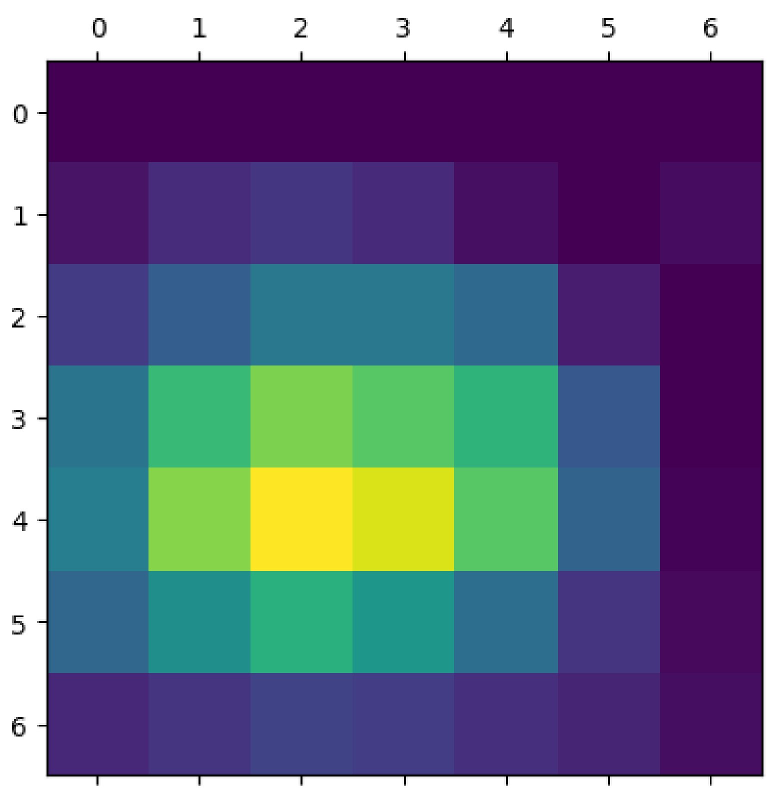

It seems like our model is finally working!! First we tried this with normal clustering and feature extraction methods. Then we tried a deep learning autoencoder as feature extractor for better clustering but that failed. Now this method with a food clissifier is finally giving good results

We can also look at the feature map visualisation below

[57]:
```
img_size=(img_height,img_width,3)
img_path= img_folder + '/' + file_names[0]
img_array = get_img_array(img_path, size=img_size)
```

```
# Print what the top predicted class is
predictions = model_TL_complex_finetune_full.predict(img_array)
print("Predicted:", class_names[np.argmax(predictions, axis=1)[0]])

# Generate class activation heatmap
heatmap = make_gradcam_heatmap(img_array, model_TL_complex_finetune_full,␣
 ↪"top_conv")

# Display heatmap
plt.matshow(heatmap)
plt.show()
save_and_display_gradcam(img_path, heatmap)
```

```
1/1 [==============================] - 0s 233ms/step
Predicted: Spaghetti bolognese
```



We can really see that our model is really able to find the pasta and it's working really well. It's not because of a coincidense that this image was predicted as pasta.