sprint3_autoencoder

December 23, 2022

1 Clustering through feature extraction with an autoencoder

In sprint2 we tried to build a recommendation system for users. A user could load an image and it'll recommend restaurant with similar images. An example is when we input a pasta image that it would recommend restaurants that sell pasta.

In the previous sprint we tried using HOG and SIFT as feature extraction, but that didn't work quite good sadly. Now we'll try to use an autoencoder for feature extraction.

[1]:
```python
import tensorflow as tf
from tensorflow import keras
import os
import numpy as np
from fastai.vision.all import PILImage
import matplotlib.pyplot as plt
```

```
2022-12-22 09:11:53.328672: 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 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2022-12-22 09:11:53.729778: E tensorflow/stream_executor/cuda/cuda_blas.cc:2981]
Unable to register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
2022-12-22 09:11:55.058855: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: cannot open shared
object file: No such file or directory
2022-12-22 09:11:55.058996: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libnvinfer_plugin.so.7'; dlerror: libnvinfer_plugin.so.7:
cannot open shared object file: No such file or directory
2022-12-22 09:11:55.059010: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot
dlopen some TensorRT libraries. If you would like to use Nvidia GPU with
TensorRT, please make sure the missing libraries mentioned above are installed
properly.
```

1.1 Checking if gpu is available

[2]:
```python
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 3971546288231065025
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 10910760960
locality {
  bus_id: 2
  numa_node: 1
  links {
  }
}
incarnation: 11866661542868266544
physical_device_desc: "device: 0, name: NVIDIA GeForce GTX 1080 Ti, pci bus id:
0000:b0:00.0, compute capability: 6.1"
xla_global_id: 416903419
]
```

```
2022-12-22 09:12:00.935320: 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 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2022-12-22 09:12:02.218620: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1616] Created device
/device:GPU:0 with 10405 MB memory:  -> device: 0, name: NVIDIA GeForce GTX 1080
Ti, pci bus id: 0000:b0:00.0, compute capability: 6.1
```

1.2 Loading dataset

[3]:
```python
IMG_HEIGHT = 128
IMG_WIDTH = 128
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]):
        # print(f"{i},{file}")
        img=PILImage.create(os.path.join(img_folder,file))
        img_resized=img.resize((IMG_HEIGHT,IMG_WIDTH))
        img_np = np.array(img_resized) #.reshape((IMG_HEIGHT,IMG_WIDTH, 3)) #.
↪astype(np.float32)
        images[i]=img_np/255
    return images

images = create_dataset(img_folder)
images_length = len(images)
X_train_full = images
X_train_full.shape
```

[3]: (15182, 128, 128, 3)

Before we continue, we want to filter out 'bad' images. 'bad' images are images that aren't food. In the notebook `differentiating-buildings-from-food-cnn.ipynb` we've made a model to seperate food from buildings.

[4]:
```
model = keras.models.load_model("../results")
results = model.predict(X_train_full)

indices = np.argwhere(results < 0.5)
indices = indices[..., 0]
test = X_train_full
X_train_full = X_train_full[indices]
X_train_full.shape
```

```
2022-12-22 09:13:17.544655: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1616] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 10405 MB memory:  -> device:
0, name: NVIDIA GeForce GTX 1080 Ti, pci bus id: 0000:b0:00.0, compute
capability: 6.1
2022-12-22 09:13:36.414850: I tensorflow/stream_executor/cuda/cuda_dnn.cc:384]
Loaded cuDNN version 8500

475/475 [==============================] - 14s 24ms/step
```

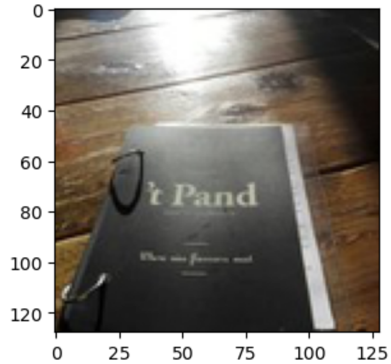
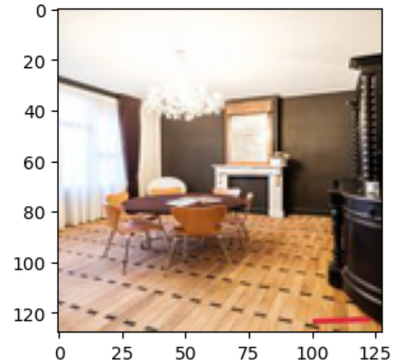
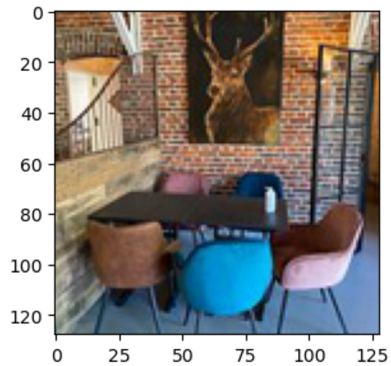
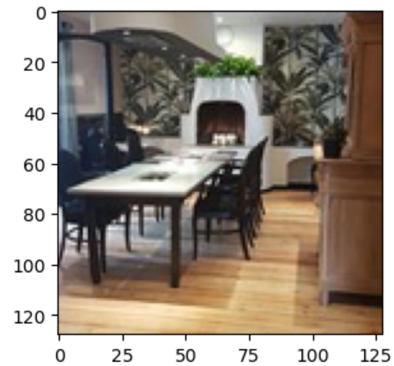
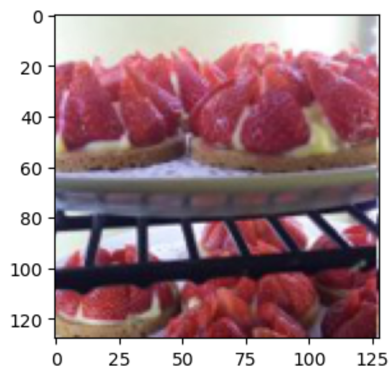
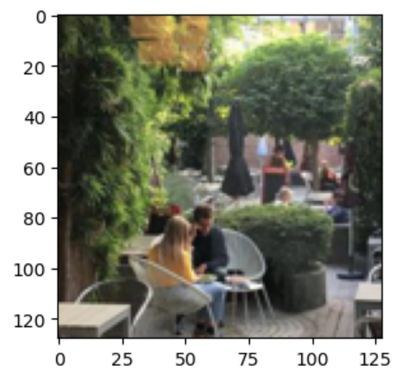
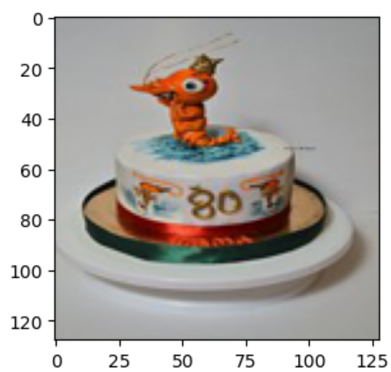
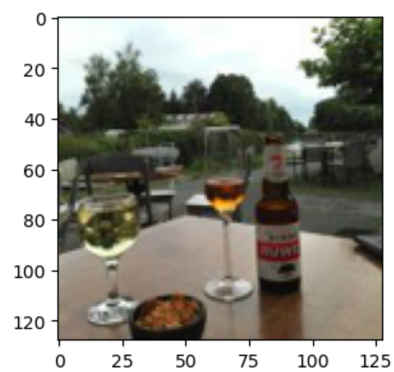
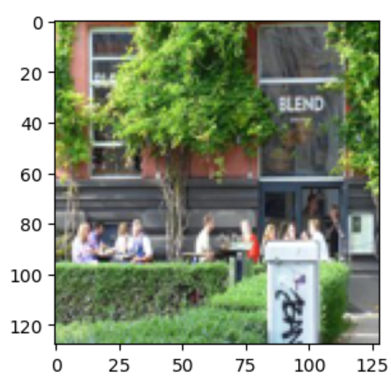
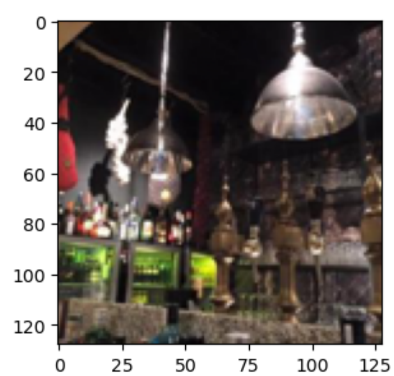
[4]: (10886, 128, 128, 3)

Now we have filtered out all bad images. We take a look at the bad images to see what the model filtered out, it should normally be all images without food.

[5]:
```
indices_buildings = np.argwhere(results >= 0.5)
indices_buildings = indices_buildings[..., 0]
```

```
buildings = test[indices_buildings]

fig=plt.figure(figsize=(10,15))
for i in range(0,10):
    plt.subplot(5,2,i+1)
    img = buildings[i]
    # restaurant name get be obtained in file_names[i]
    plt.imshow(img)
fig.tight_layout()
plt.show()
```


't Pand








BLEND


That looks good, two images with food are filtered out but the other images are correctly filtered out.

Splitting the dataset in a very small (5 images) test dataset, just to see how the autoencoder will work on new images.

[6]:
```
from sklearn.model_selection import train_test_split

positie = len(X_train_full) - 5
test_set = X_train_full[positie:len(X_train_full)]
X_train_full = X_train_full[:positie]

print(test_set.shape)
print(X_train_full.shape)
```

```
(5, 128, 128, 3)
(10881, 128, 128, 3)
```

1.3 Training the model

The bottleneck of our autoencoder has a shape of (16, 16, 128). This is the shape of the output of the encoder. The ratio between the input and the output of the encoder is 16x16x128 / 128x128x3 = 66.67% (The output features consist of 66.67% of the total amount of pixel values)

[7]:
```
def rounded_accuracy(y_true, y_pred):
    return keras.metrics.binary_accuracy(tf.round(y_true), tf.round(y_pred))



conv_encoder = keras.models.Sequential([
    # keras.layers.Reshape([128, 128, 3], input_shape=[128, 128, 3]),
    keras.layers.Conv2D(32, kernel_size=3, padding="SAME", activation="selu"),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Conv2D(64, kernel_size=3, padding="SAME", activation="selu"),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Conv2D(128, kernel_size=3, padding="SAME", activation="selu"),
    keras.layers.MaxPool2D(pool_size=2)
])
conv_decoder = keras.models.Sequential([
    keras.layers.Conv2DTranspose(64, kernel_size=3, strides=2, padding="SAME",␣
 ↪activation="selu", input_shape=[16, 16, 128]),
    keras.layers.Conv2DTranspose(32, kernel_size=3, strides=2, padding="SAME",␣
 ↪activation="selu"),
    keras.layers.Conv2DTranspose(3, kernel_size=3, strides=2, padding="SAME",␣
 ↪activation="sigmoid"),
])
```

```
conv_ae = keras.models.Sequential([conv_encoder, conv_decoder])


conv_ae.compile(loss="binary_crossentropy", optimizer=keras.optimizers.
 ↪SGD(learning_rate=0.1),
                metrics=[rounded_accuracy])
history = conv_ae.fit(X_train_full, X_train_full, epochs=10, validation_split=0.
 ↪2)

# print(conv_encoder.summary())
# conv_decoder.summary()
```

```
Epoch 1/10
272/272 [==============================] - 16s 48ms/step - loss: 0.5767 -
rounded_accuracy: 0.8325 - val_loss: 0.5447 - val_rounded_accuracy: 0.8937
Epoch 2/10
272/272 [==============================] - 10s 35ms/step - loss: 0.5409 -
rounded_accuracy: 0.8986 - val_loss: 0.5403 - val_rounded_accuracy: 0.9028
Epoch 3/10
272/272 [==============================] - 10s 36ms/step - loss: 0.5366 -
rounded_accuracy: 0.9060 - val_loss: 0.5368 - val_rounded_accuracy: 0.9058
Epoch 4/10
272/272 [==============================] - 10s 36ms/step - loss: 0.5344 -
rounded_accuracy: 0.9099 - val_loss: 0.5325 - val_rounded_accuracy: 0.9176
Epoch 5/10
272/272 [==============================] - 9s 34ms/step - loss: 0.5323 -
rounded_accuracy: 0.9143 - val_loss: 0.5327 - val_rounded_accuracy: 0.9167
Epoch 6/10
272/272 [==============================] - 10s 36ms/step - loss: 0.5311 -
rounded_accuracy: 0.9166 - val_loss: 0.5387 - val_rounded_accuracy: 0.9064
Epoch 7/10
272/272 [==============================] - 9s 34ms/step - loss: 0.5300 -
rounded_accuracy: 0.9195 - val_loss: 0.5390 - val_rounded_accuracy: 0.9004
Epoch 8/10
272/272 [==============================] - 10s 36ms/step - loss: 0.5289 -
rounded_accuracy: 0.9217 - val_loss: 0.5309 - val_rounded_accuracy: 0.9172
Epoch 9/10
272/272 [==============================] - 9s 35ms/step - loss: 0.5285 -
rounded_accuracy: 0.9224 - val_loss: 0.5298 - val_rounded_accuracy: 0.9230
Epoch 10/10
272/272 [==============================] - 10s 35ms/step - loss: 0.5277 -
rounded_accuracy: 0.9238 - val_loss: 0.5289 - val_rounded_accuracy: 0.9264
```

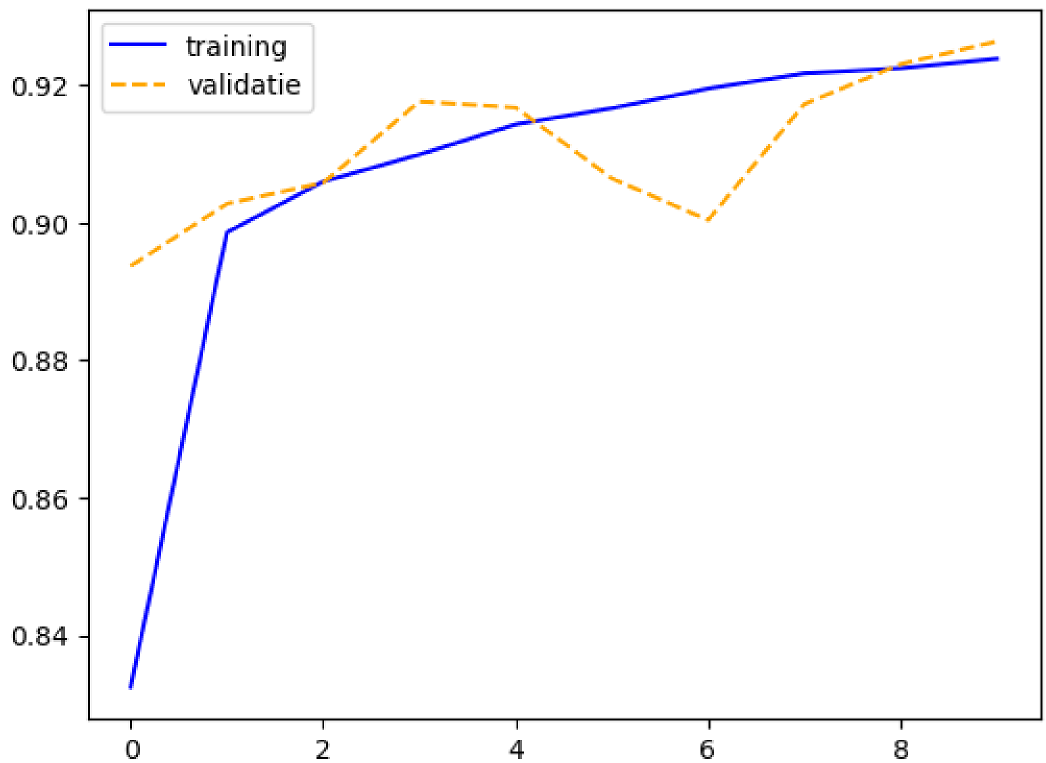
Plotting training and loss curve

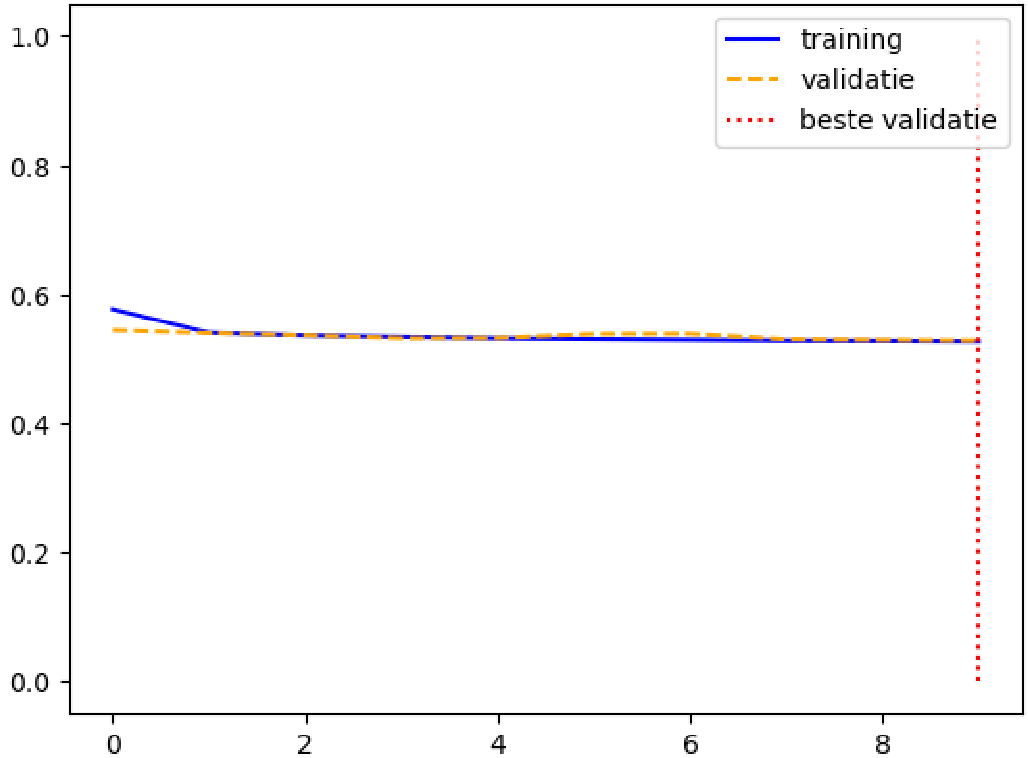
[8]:
```
train_loss_values = history.history['loss']
val_loss_values = history.history['val_loss']
best_val_idx = np.argmin(val_loss_values)
```

```
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['rounded_accuracy']
val_loss_values = history.history['val_rounded_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()
```





We immediatly start with a high accuracy and it keeps increasing. Our loss however almost doesn't decrease.

We check how the test images look when passing through the autoencoder.

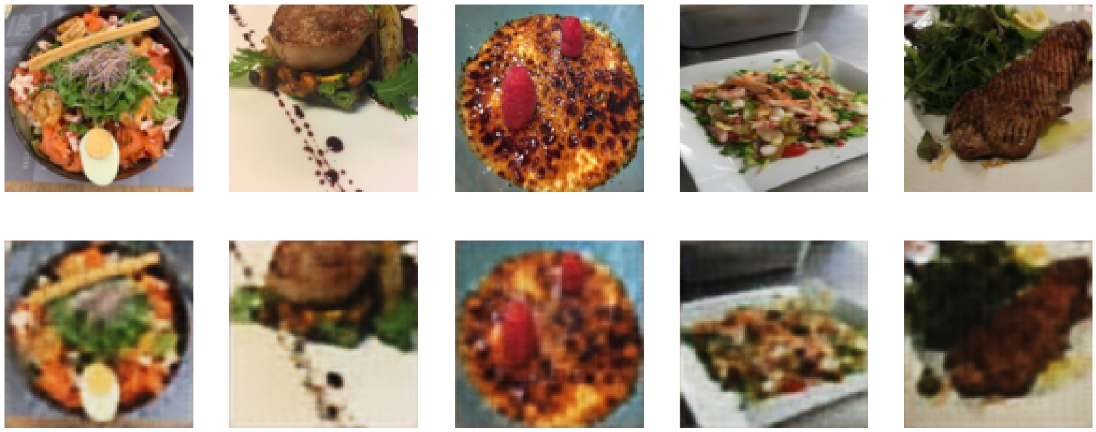
[9]:
```
def plot_image(image):
    plt.imshow(image, cmap="binary")
    plt.axis("off")


def show_reconstructions(model, images=test_set, n_images=5):
    reconstructions = model.predict(images[:n_images])
    fig = plt.figure(figsize=(n_images * 1.5, 3))
    for image_index in range(n_images):
        plt.subplot(2, n_images, 1 + image_index)
        plot_image(images[image_index])
        plt.subplot(2, n_images, 1 + n_images + image_index)
        plot_image(reconstructions[image_index])


show_reconstructions(conv_ae)

plt.show()
```

```
1/1 [==============================] - 0s 233ms/step
```



The reconstructed images look pretty good, that means that the feature extraction works. If the images wouldn't look like the original ones, that would mean that the features after encoding aren't representative. In this case the features after encoding are representative because the original image can be almost reconstructed.

[10]:
```
X_train_full[:1].shape
```

[10]:
```
(1, 128, 128, 3)
```

1.4 Save model

We only save the encoder, we don't need the decoder for feature extraction. The decoder was only needed when training.

[11]:
```
conv_encoder.save('./model_last_version/')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op while saving (showing
3 of 3). These functions will not be directly callable after loading.
```

```
INFO:tensorflow:Assets written to: ./model_last_version/assets
```

```
INFO:tensorflow:Assets written to: ./model_last_version/assets
```

1.5 Getting features of all the images

These features will be used in another notebook.

[12]:
```
features = conv_encoder.predict(X_train_full)
```

```
341/341 [==============================] - 3s 9ms/step
```

[13]:
```
import pickle
with open('features_encoder_pictures_with_no_buildings_last_version.pkl','wb')␣
 ↪as f:
    pickle.dump(features, f)
```

In notebook `sprint3_autoencoder_features.ipynb` we will use these features saved to the pickle file to cluster

1.6 References

https://github.com/ageron/handson-ml2