VQVAE on the OASIS dataset

We're here trying to train a generative VQVAE to generate images from the OASIS dataset with an SSIM of at least 0.6.

The VQVAE does the encoding and the quantising, but also the decoding. A PixelCNN is used alongside it to perform prior training for random image generation (with a quantised input such as the VQVAE would produce)

I. Data

The datasets I used were those in the link on blackboard. I intentionally discarded the segmentation datasets. The data is loaded thanks the load_data() function which also normalises the images so we don't have to

II. Models implemented

Most of the code was adapted from the corresponding Keras tutorial (https://keras.io/examples/generative/vq_vae/).

a) VQ-VAE

The model

Class VQ : the vector quantiser class, which, as its name may suggest, implements the codebook. Used 512 codebook vectors here.

Function encoder : implements the encoder. Opted for 4 convolutional layers with, in that order, 256, 128, 64 and 128 filters (the last number corresponds to the dimension of the codebook vectors)

Function decoder : implements the decoder, very much just the symmetric to the encoder

Function vqvae : builds the VQVAE with calls to VQ, encoder, and decoder. The end product had over 955,000 parameters:

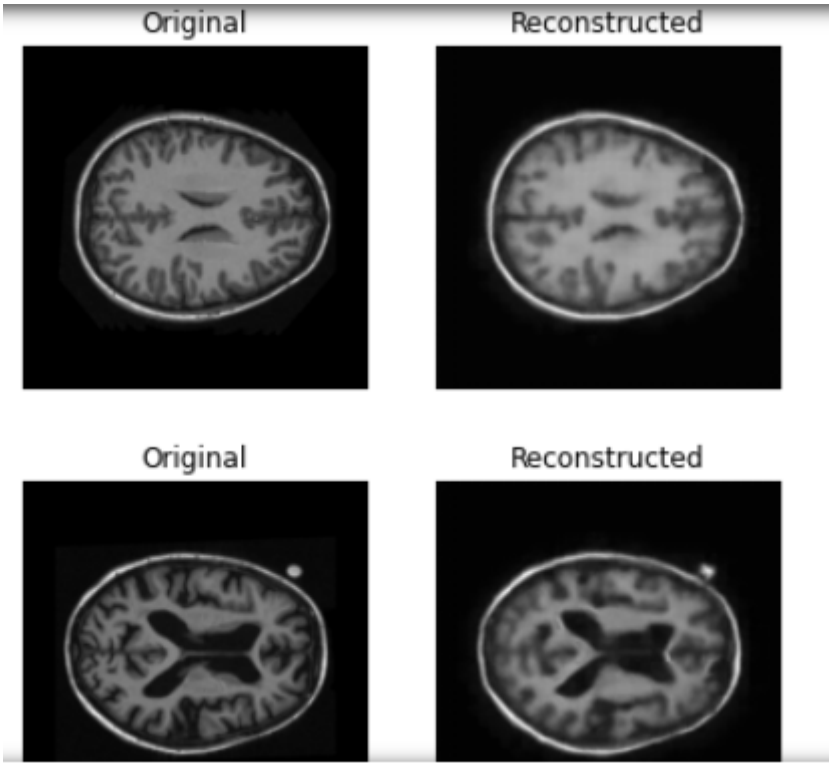
```
Model: "VQVAE"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 256, 256, 1)]     0
_________________________________________________________________
encoder (Functional)         (None, 32, 32, 128)       445248
_________________________________________________________________
Quantiser (VQ)               (None, 32, 32, 128)       65536
_________________________________________________________________
decoder (Functional)         (None, 256, 256, 1)       445121
=================================================================
Total params: 955,905
Trainable params: 955,905
Non-trainable params: 0
_________________________________________________________________
```

Training

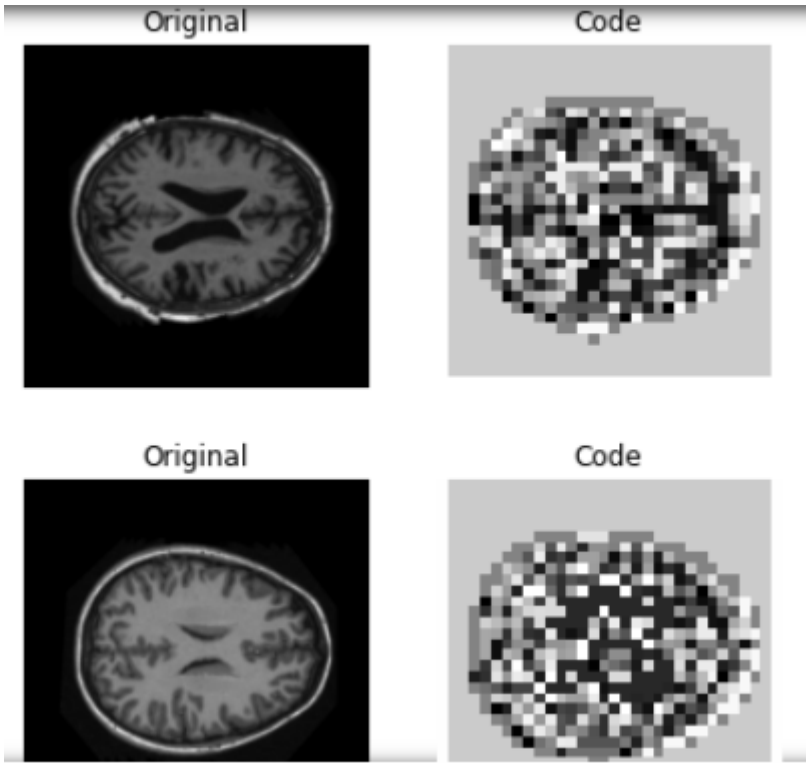
Trained for 20 epochs with batch size of 32. The weights were saved under the name "VQVAE".

Results

If we take a look a a few encoded and decoded test images, that's what we get for 2 of them :



Not too bad, but still sort of blurry, the details are lacking. As for the corresponding codes :



Obtained a mean SSIM of 0.8682193400121998 on the test set for the, as shown on the picture below:

```
#Reconstructing the images in the test set to calculate SSIM
reconstructed_test_images = trained_vqvae.predict(X_test)

from skimage.metrics import structural_similarity as ssim

SSIM=np.zeros(shape=(reconstructed_test_images.shape[0],))

for i in range(len(SSIM)):
    SSIM[i]=ssim(X_test[i,:,:,0],reconstructed_test_images[i,:,:,0])

total_ssim=np.sum(SSIM)
mean_ssim = total_ssim/X_test.shape[0]

print("The mean SSIM on the test dataset is",mean_ssim)
```

```
<ipython-input-26-d5c350ca123d>:9: UserWarning: Inputs have mismatched
  SSIM[i]=ssim(X_test[i,:,:,0],reconstructed_test_images[i,:,:,0])
```

```
The mean SSIM on the test dataset is 0.8682193400121998
```

b) PixelCNN

The model

This one was far deeper than the VQ-VAE. It would take too long to sum up its characteristics, but I built it with 16 residual blocks and 16 convolutional layers. Here's the bottom of its summary:

```
_________________________________________________________________
pixel_cnn_layer_893 (PixelCN (None, 32, 32, 128)       16512
_________________________________________________________________
pixel_cnn_layer_894 (PixelCN (None, 32, 32, 128)       16512
_________________________________________________________________
pixel_cnn_layer_895 (PixelCN (None, 32, 32, 128)       16512
_________________________________________________________________
pixel_cnn_layer_896 (PixelCN (None, 32, 32, 128)       16512
_________________________________________________________________
conv2d_1809 (Conv2D)         (None, 32, 32, 512)       66048
=================================================================
Total params: 5,119,616
Trainable params: 5,119,616
Non-trainable params: 0
_________________________________________________________________
```

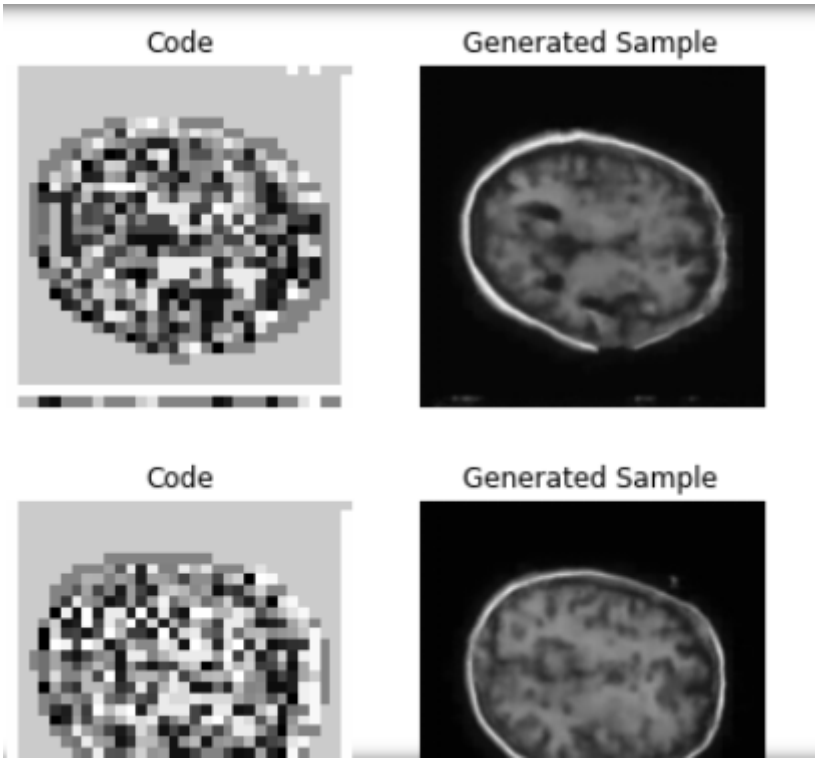
Training

I trained it on 100 epochs, with batch size 32 and validation split 0.1 :

```
Epoch 99/100
272/272 [==============================] - 17s 63ms/step - loss: 0.6653 - accuracy: 0.7797 - val_loss: 0.8908 - val_accu
0.7307
Epoch 100/100
272/272 [==============================] - 17s 63ms/step - loss: 0.6611 - accuracy: 0.7810 - val_loss: 0.8912 - val_accu
0.7301
```

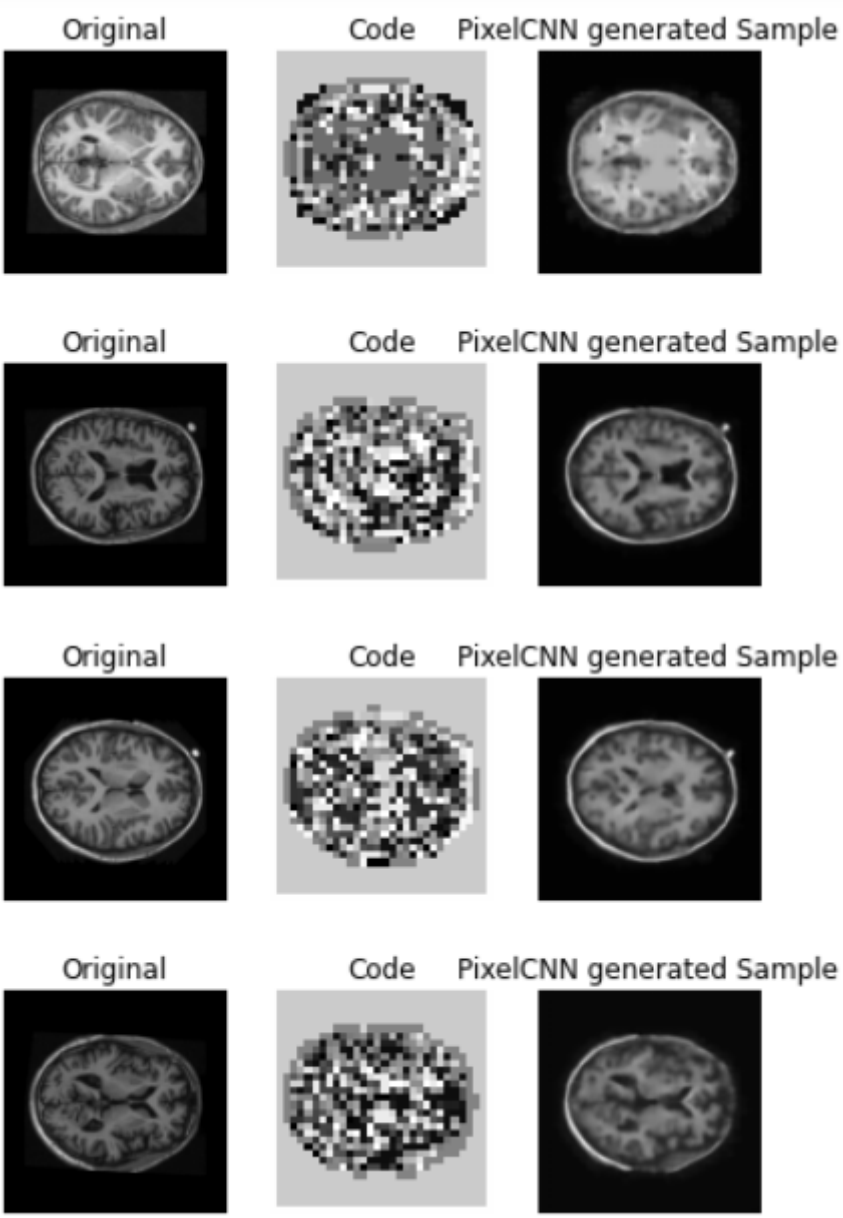
Results

Random image generation with priors



As we can see, a few generated samples actually look like brains (I wish I found a way to show more on here but hopefully you can find everything on the notebook).

Generating test images with the test set encodings yielded the following (I randomly printed out 10 and below are 4 of those 10):



As for the SSIM, it is as follows :

```
test_SSIM=np.zeros(shape=(X_test.shape[0],))

for i in range(len(test_SSIM)):
    test_SSIM[i]=ssim(X_test[i,:,:,0],generated_test_samples[i,:,:,0])

test_total_ssim=np.sum(test_SSIM)
test_mean_ssim = test_total_ssim/X_test.shape[0]

print("The mean SSIM on the test dataset is",test_mean_ssim)
```

```
<ipython-input-172-6cb974e955ef>:4: UserWarning: Inputs have mismatched
  test_SSIM[i]=ssim(X_test[i,:,:,0],generated_test_samples[i,:,:,0])
```

```
The mean SSIM on the test dataset is 0.8682193400545757
```

Not how it looks extremely similar to the VQ-VAE decoded test image. The two are actually different, but the difference only starts at the 11th decimal place.

III. Dependencies

tensorflow 2.4

tensorflow-probability 0.12.1

numpy (any version that's compatible with tensorflow 2.4)

imageio (data loading)

skimage.metrics (for ssim)

os (used the listdir function to load the images)

IV. Additional Notes

I imported both the model weights and pushed them on the repository (they're named "VQVAE" and "PixelCNN"). But you're more than welcome to rerun the training if you wish to.

I also included the notebook. The raw code in the Code.py file, and the notebook in the Code.ipynb file.