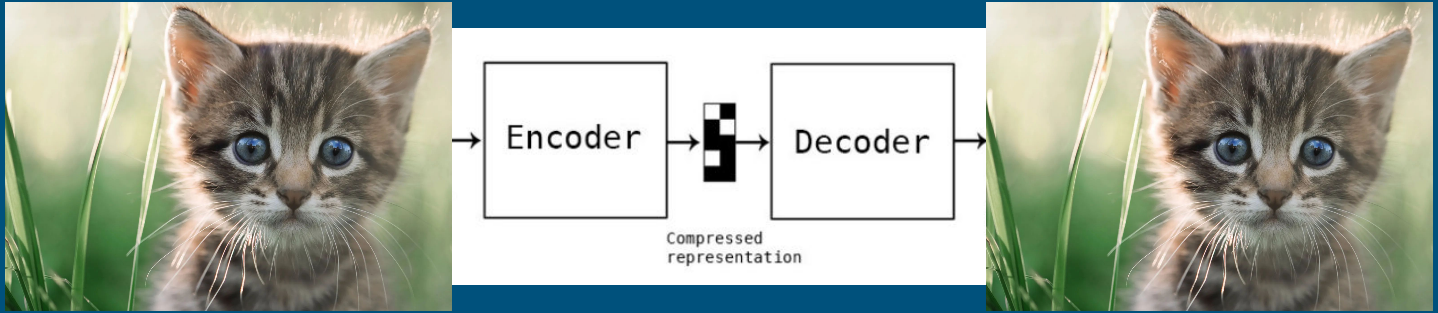
Convolutional Autoencoders for Dimensionality Reduction

By Luke Harwood | Kevin Paganini

Type of machine learning problem

Main Purpose: Reducing Complexity/Dimensionality of Feature Space

- Unsupervised learning technique
- Used in regression or classification
- Performs Feature Extraction (or Feature Transformation)
- Input and output are the same image



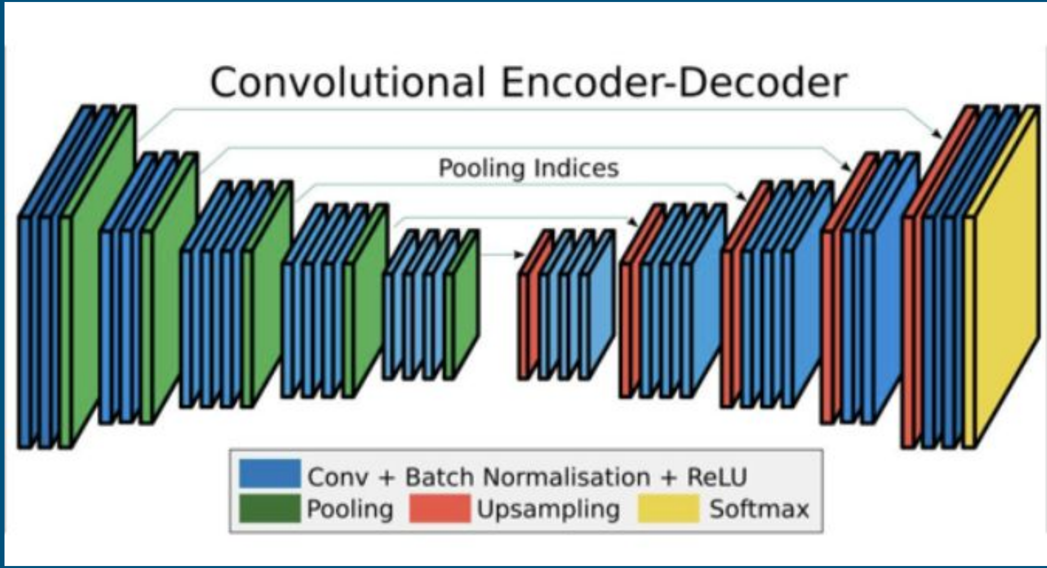
Model Architecture

Model consists of three main parts

- The Encoder
- Latent Layer
- The Decoder

Input Image:
128, 128, 3

Output Image (same):
128, 128, 3



Latent Space:
1024

Model implementation

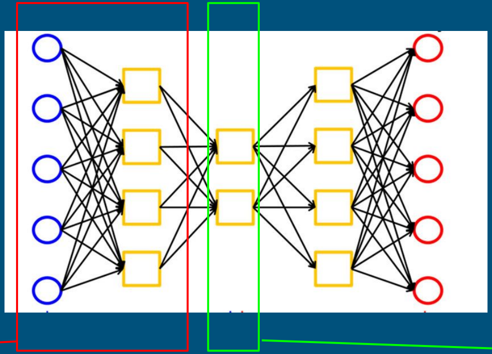
Used keras + tensorflow to implement the sequential model
Image preprocessing: CV2 → Resizing images to 128, 128, 3
Encoder: Conv2D, MaxPooling2D, GlobalMaxPooling2D, Dense
Decoder: Conv2D, UpSampling2D, Reshape
Image display: Matplotlib

Train Image Shapes:

```
(150, 150, 3)    13986
(113, 150, 3)        7
(111, 150, 3)        3
(135, 150, 3)        3
(144, 150, 3)        2
(123, 150, 3)        2
(142, 150, 3)        2
(146, 150, 3)        2
(143, 150, 3)        2
(134, 150, 3)        2
(136, 150, 3)        2
(108, 150, 3)        2
(105, 150, 3)        1
(97, 150, 3)         1
(131, 150, 3)        1
(147, 150, 3)        1
(81, 150, 3)         1
(145, 150, 3)        1
(141, 150, 3)        1
(100, 150, 3)        1
(103, 150, 3)        1
(76, 150, 3)         1
(120, 150, 3)        1
(102, 150, 3)        1
(119, 150, 3)        1
(133, 150, 3)        1
(115, 150, 3)        1
(124, 150, 3)        1
(110, 150, 3)        1
(149, 150, 3)        1
(140, 150, 3)        1
dtype: int64
```

Encoder Architecture

- Goes from high-dimensional space to low-dimensional space
- 5 Convolutional layers with increasing filter number
- 4 MaxPooling layers
- 1 GlobalMaxPooling
- Latent Layer: 1024 Dense layer

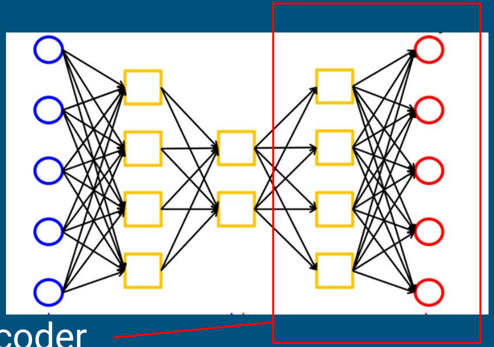



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 128, 128, 32)      896

max_pooling2d (MaxPooling2D  (None, 64, 64, 32)        0
)

conv2d_1 (Conv2D)            (None, 64, 64, 64)        18496

max_pooling2d_1 (MaxPooling  (None, 32, 32, 64)        0
2D)

conv2d_2 (Conv2D)            (None, 32, 32, 128)       73856

max_pooling2d_2 (MaxPooling  (None, 16, 16, 128)       0
2D)

conv2d_3 (Conv2D)            (None, 16, 16, 256)       295168

max_pooling2d_3 (MaxPooling  (None, 8, 8, 256)         0
2D)

conv2d_4 (Conv2D)            (None, 8, 8, 512)         1180160

global_max_pooling2d (Globa  (None, 512)               0
lMaxPooling2D)

dense_6 (Dense)              (None, 1024)              525312

=================================================================
Total params: 2,093,888
Trainable params: 2,093,888
Non-trainable params: 0
```

Decoder Architecture

- Goes from low-dimensional space to high-dimensional space
- Similar but in reverse
- 1 Reshape layer
- 6 Convolutional layers
- 3 UpSampling layers

Upsampling is the opposite of Pooling



Decoder

```
Layer (type)                 Output Shape              Param #
=================================================================
reshape_1 (Reshape)          (None, 16, 16, 4)         0

conv2d_5 (Conv2D)            (None, 16, 16, 512)       18944

up_sampling2d (UpSampling2D  (None, 32, 32, 512)       0
)

conv2d_6 (Conv2D)            (None, 32, 32, 256)       1179904

up_sampling2d_1 (UpSampling  (None, 64, 64, 256)       0
2D)

conv2d_7 (Conv2D)            (None, 64, 64, 128)       295040

up_sampling2d_2 (UpSampling  (None, 128, 128, 128)     0
2D)

conv2d_8 (Conv2D)            (None, 128, 128, 64)      73792

conv2d_9 (Conv2D)            (None, 128, 128, 32)      18464

conv2d_10 (Conv2D)           (None, 128, 128, 3)       867

=================================================================
Total params: 1,587,011
Trainable params: 1,587,011
Non-trainable params: 0
```

Hyperparameters

- Number and makeup of hidden layers
- Cost Function
- Optimizer
- Bottleneck layer size
- Data Augmentation
- Image Size

```
Conv2D(filters=32,kernel_size = (3, 3),strides=1,padding='same', activation='selu', kernel_initializer='lecun_normal'))
```

```
deep_ae.compile(loss=tf.keras.losses.MeanSquaredError(), optimizer=tf.keras.optimizers.Adam(), metrics=[rounded_accuracy])
```

```
deep_e.add(keras.layers.Dense(1024,activation='selu', kernel_initializer='lecun_normal'))
```

```
resample_x = 128
resample_y = 128
resample_z = 3
```

Cost function and Model fitting

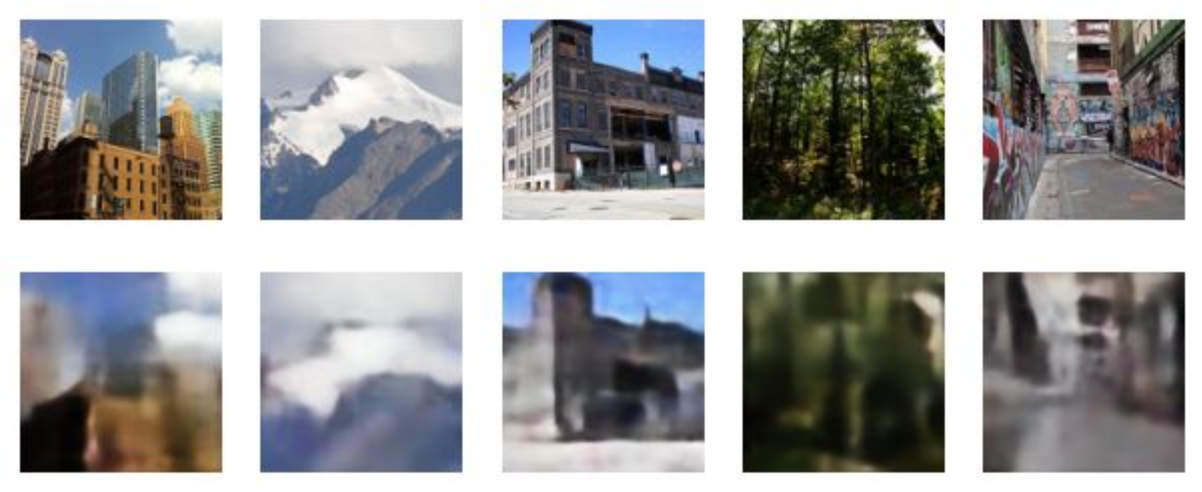
MSE for one image:

$$\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

Cost function: Pixel by Pixel - Mean squared error
Result: ~ 0.015 MSE (1 is bad, 0 is good)
Optimizer: Adam optimizer (form of gradient descent)

Input

Output

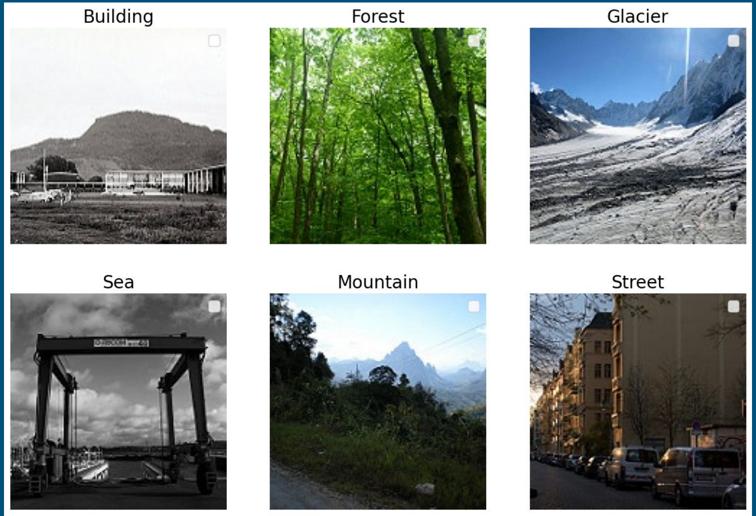


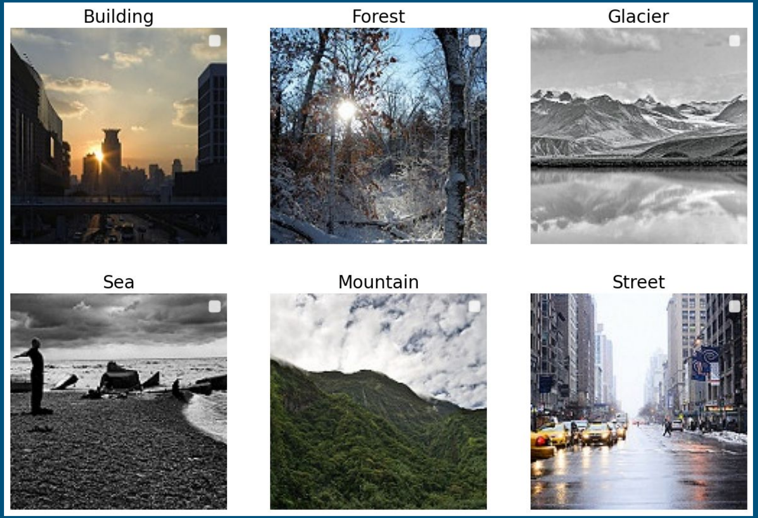
Intel Image Dataset

~14000 training instances
~3000 test instances
Balanced dataset
Image size: 150, 150, 3
6 classes of images

```
Total number of train buildings instances: 2191
Total number of train forest instances: 2271
Total number of train glacier instances: 2404
Total number of train mountain instances: 2512
Total number of train sea instances: 2274
Total number of train street instances: 2382
```

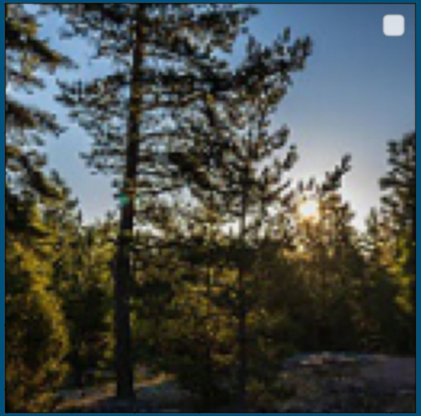
```
Total number of test buildings instances: 437
Total number of test forest instances: 474
Total number of test glacier instances: 553
Total number of test mountain instances: 525
Total number of test sea instances: 510
Total number of test street instances: 501
```

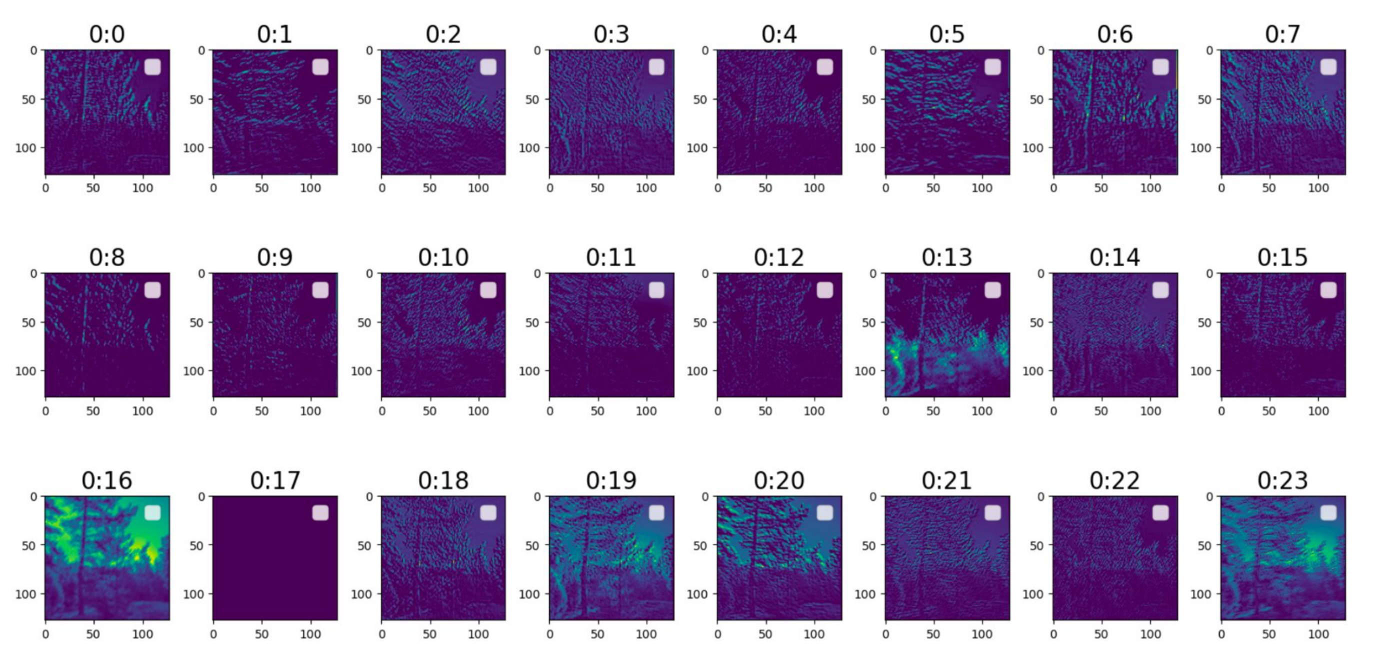




1st Convolution

32 Filters

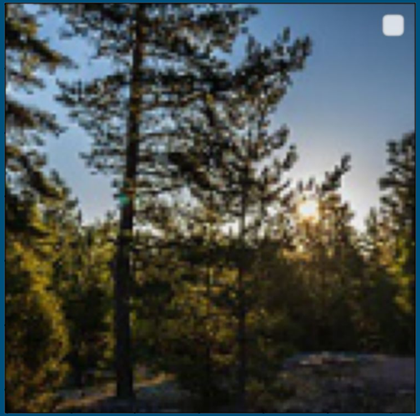


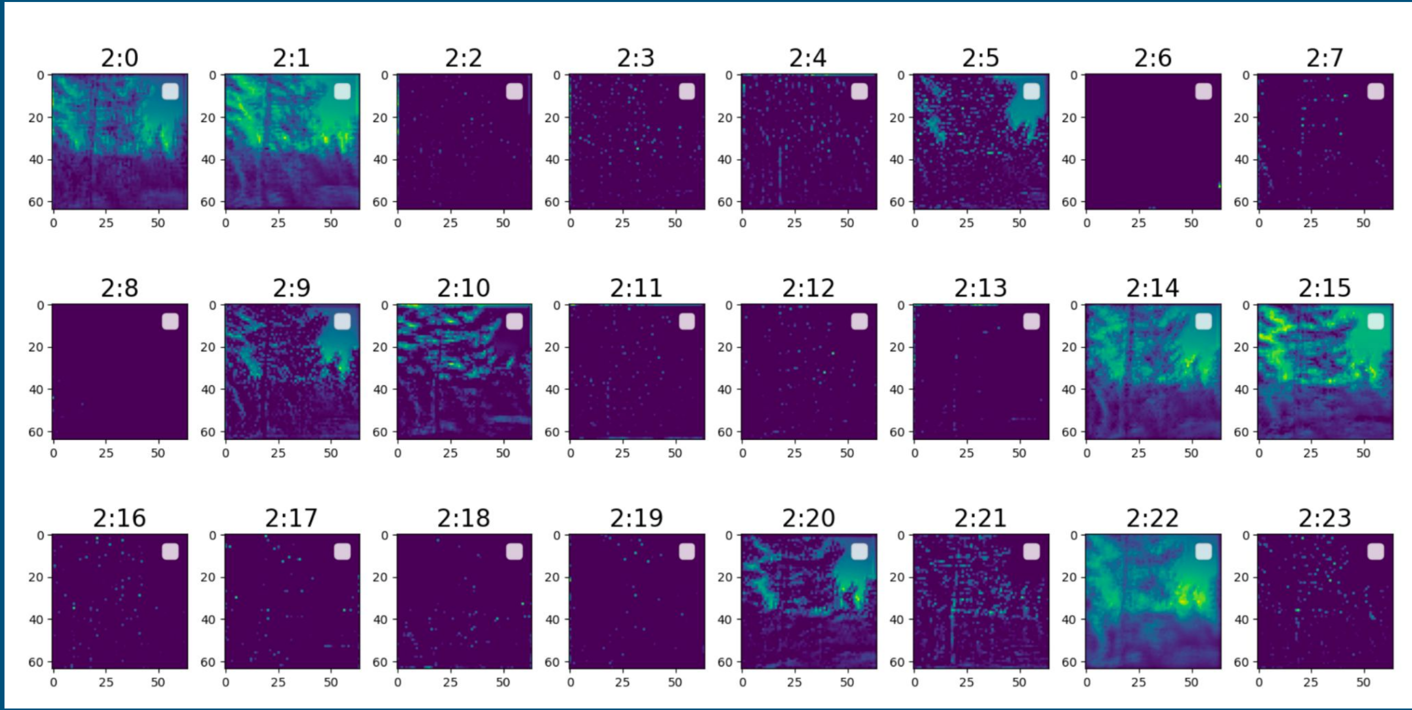



```
Conv2D(filters=32,kernel_size = (3, 3),strides=1,padding='same', activation='selu', kernel_initializer='lecun_normal')
```

2nd Convolution

64 Filters

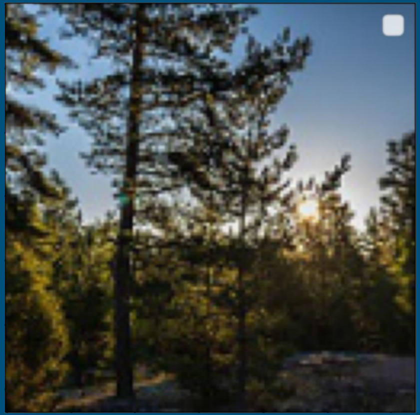


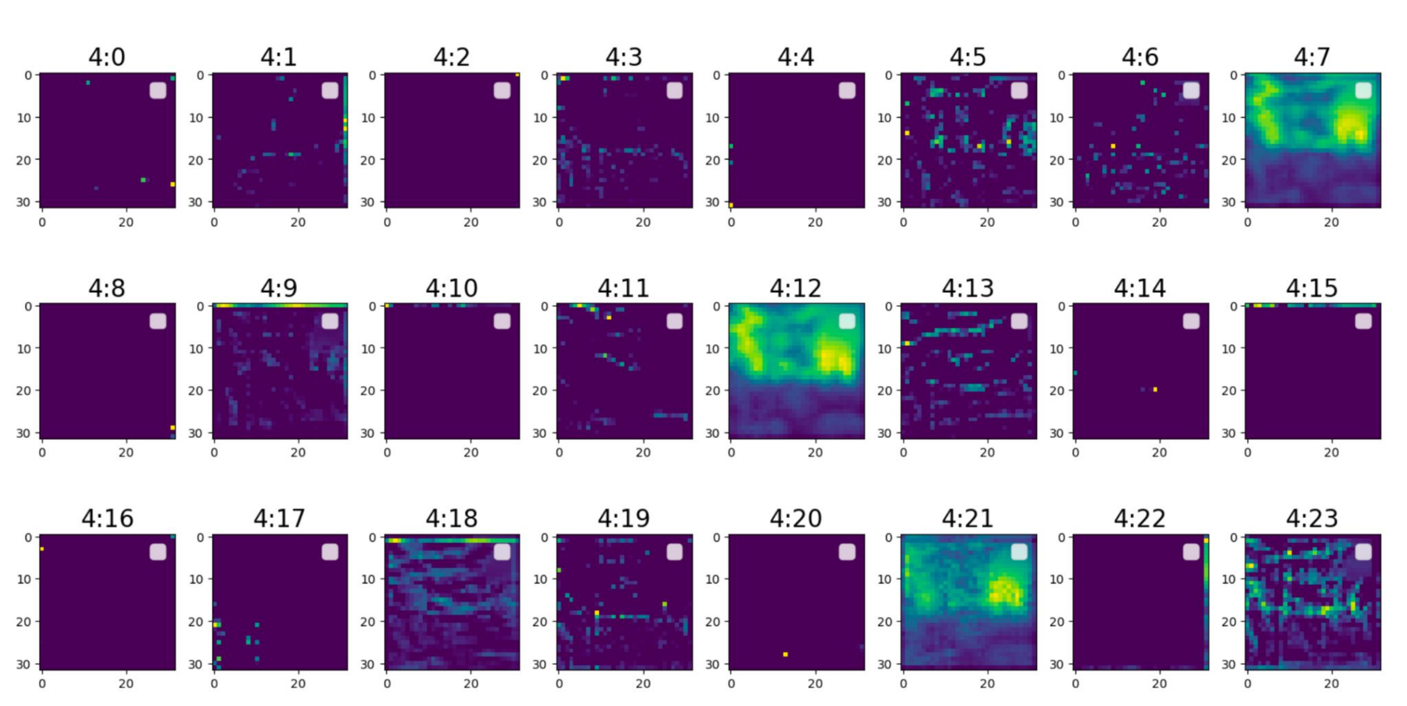



```
Conv2D(filters=64,kernel_size = (3, 3),strides=1,padding='same', activation='selu', kernel_initializer='lecun_normal')
```

3rd Convolution

128 Filters

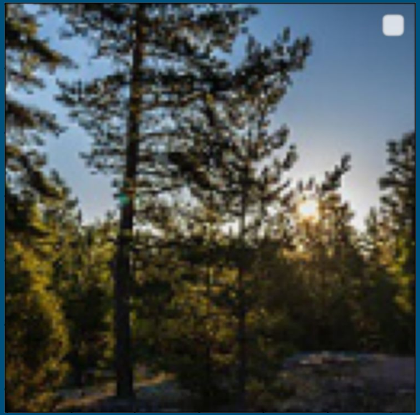


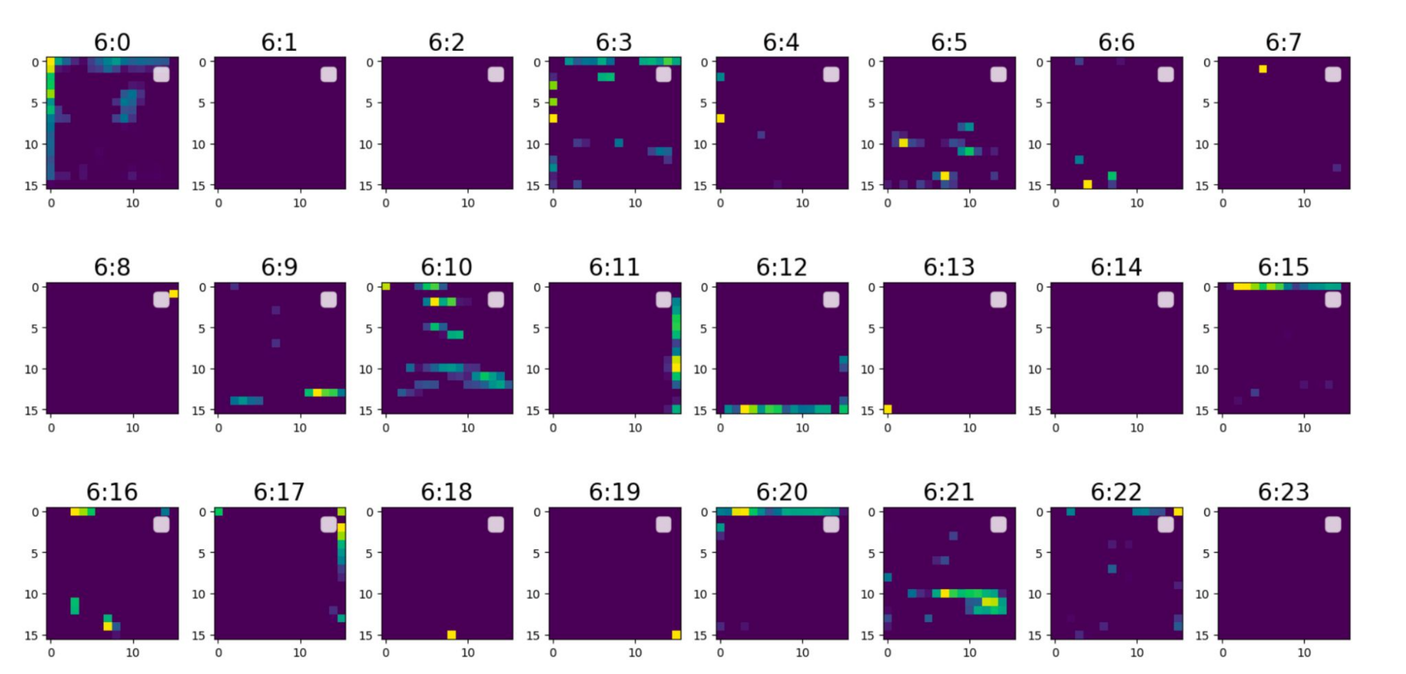



```
Conv2D(filters=128,kernel_size = (3, 3),strides=1,padding='same', activation='selu’, kernel_initializer=’lecun_normal’)
```

4th Convolution

256 Filters

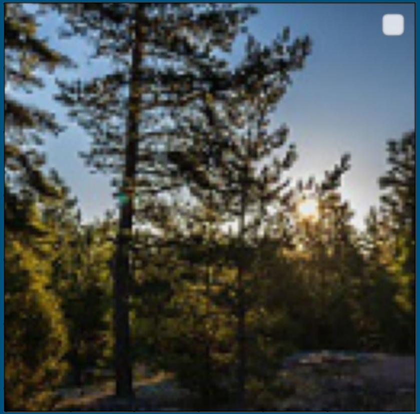


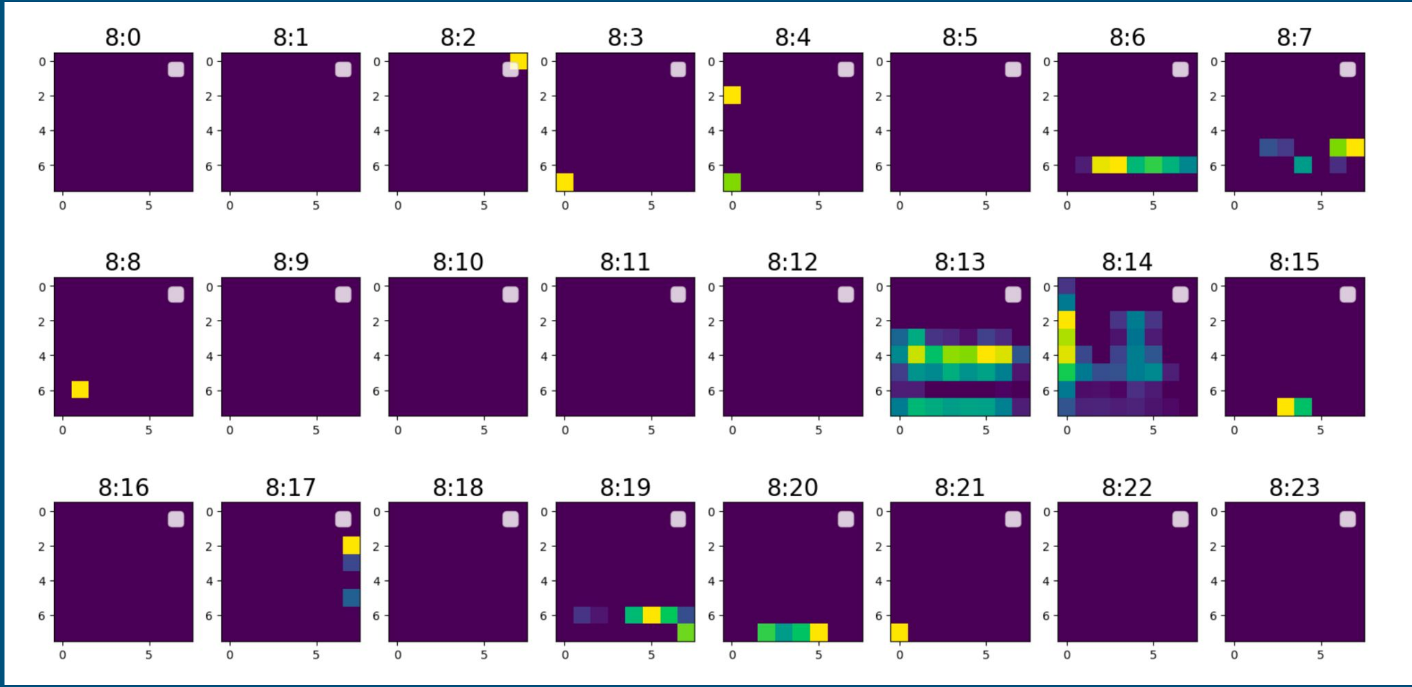



```
Conv2D(filters=256,kernel_size = (3, 3),strides=1,padding='same', activation='selu', kernel_initializer='lecun_normal)
```

Last Convolution

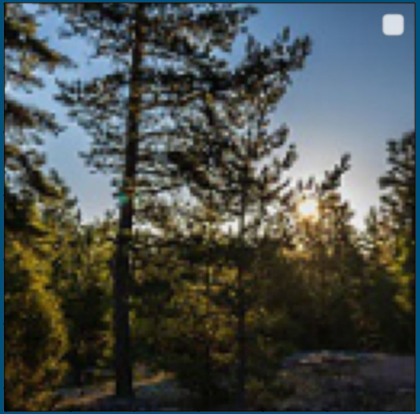
512 Filters





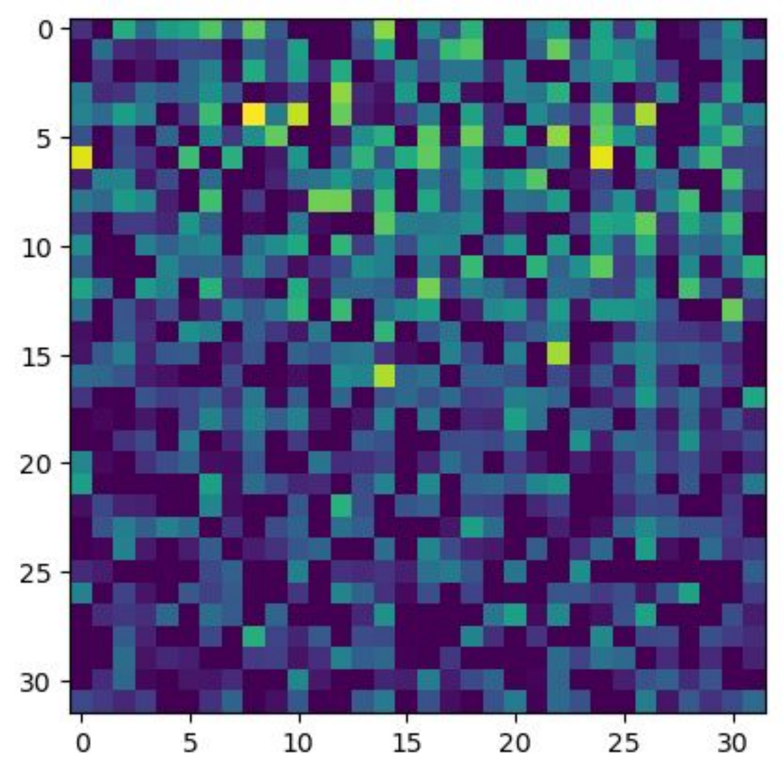

```
Conv2D(filters=512,kernel_size = (3, 3),strides=1,padding='same', activation='selu', kernel_initializer='lecun_normal)
```

Encoding

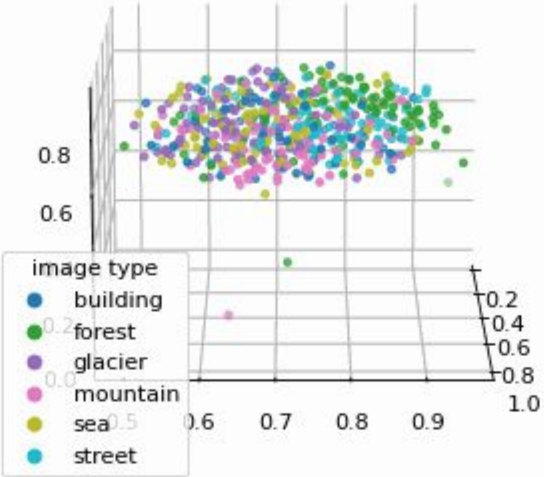


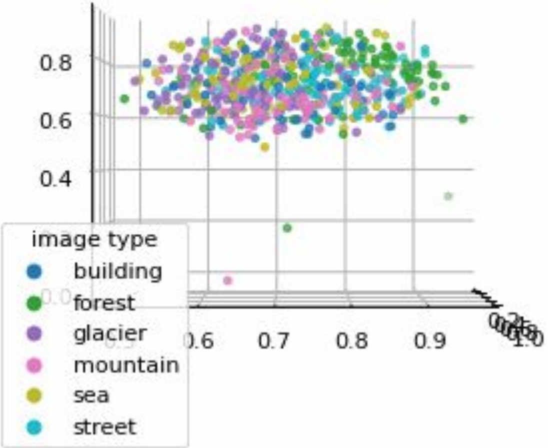
128 x 128 x 3

32 x 32 x 1



Latent Space on Validation Data

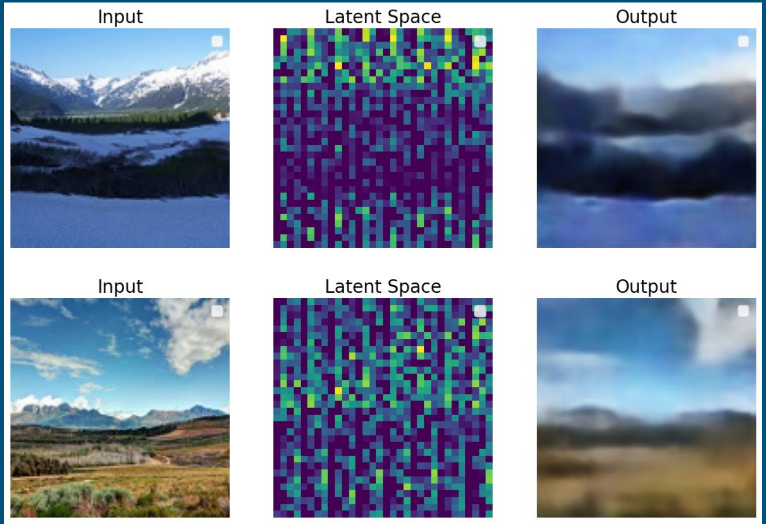




Reduced from 1024 to 3 dimensions with TSNE

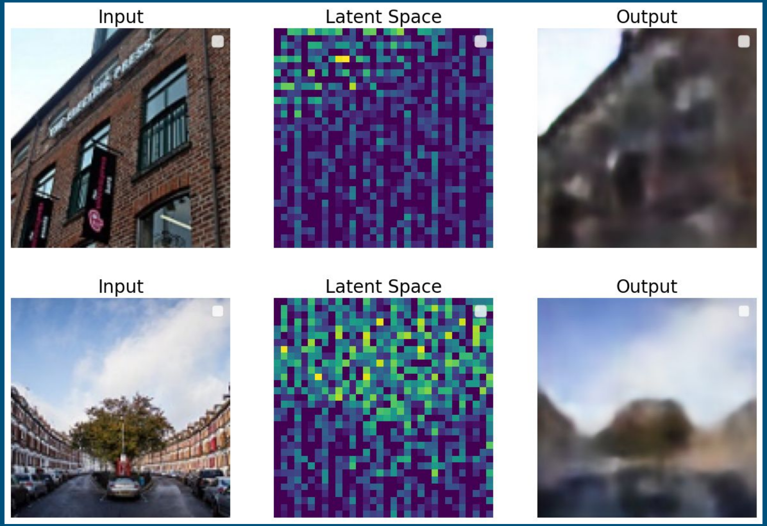
Visualisation of latent space of four samples

0.01



0.007

0.016



0.01

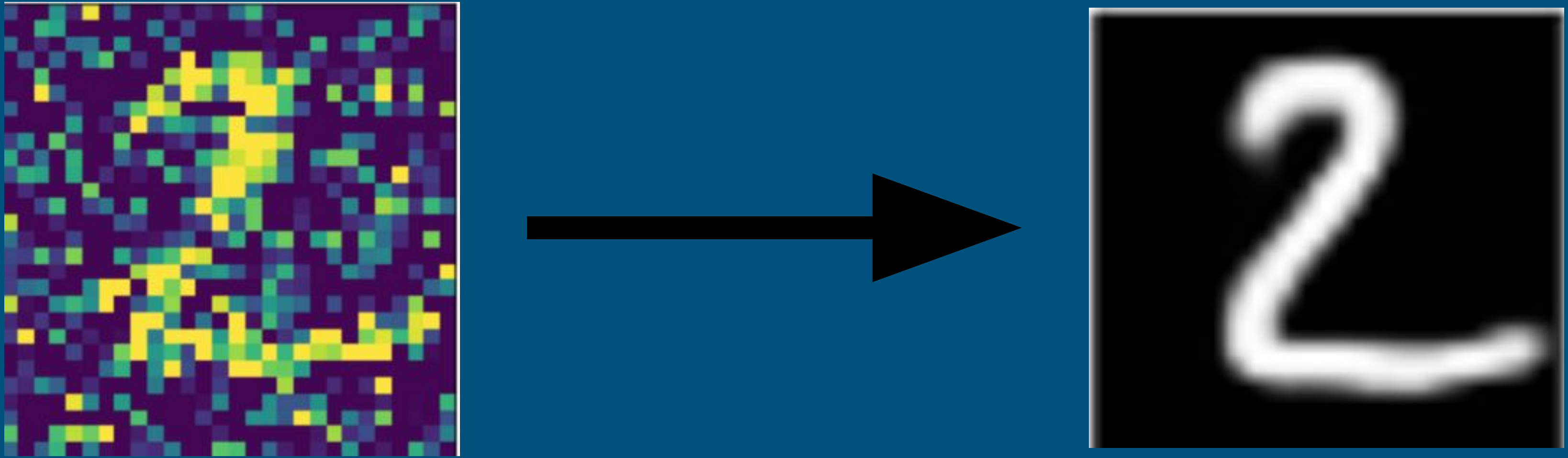
Notice: ordering of pixels in latent space does not matter it was reshaped into this format for display purposes

Applications

- Preprocessing Data (Dimensionality Reduction)
- Image denoising
- Anomaly Detection
- Generation of new images

Image Denoising

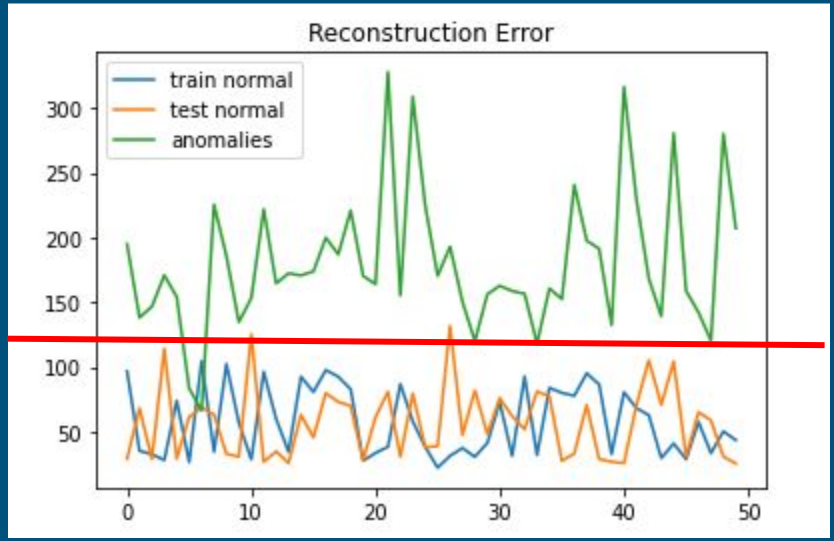
- Example: Trained model on the MNIST handwritten digits dataset



https://miro.medium.com/max/1100/1*GFd9K-88w06YKOEqphGMng.png

Anomaly Detection

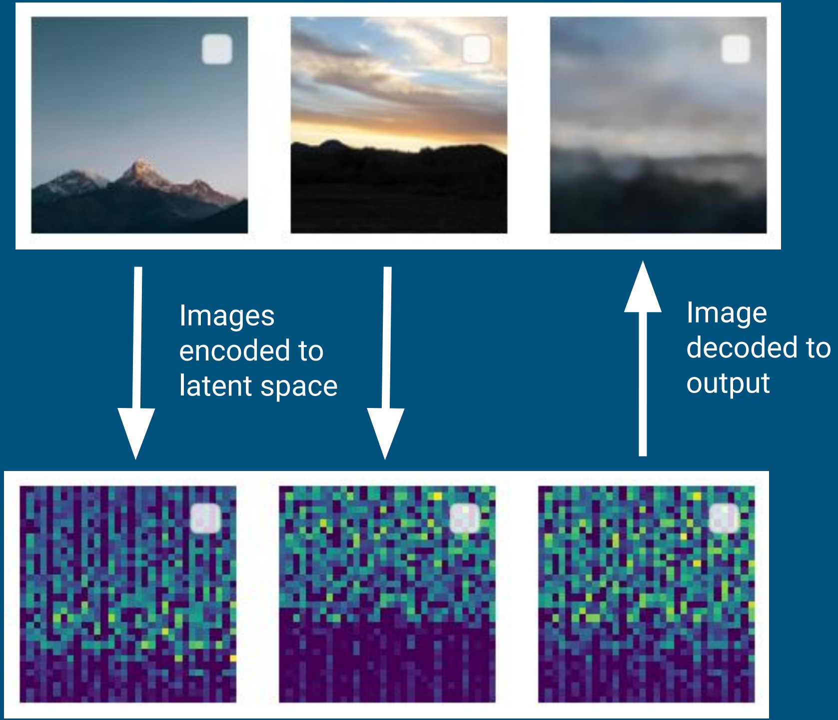
- Detect outliers in dataset
- Based on reconstruction error
- Can be passed to another model
  - (e.g. Linear Classifier, etc.)

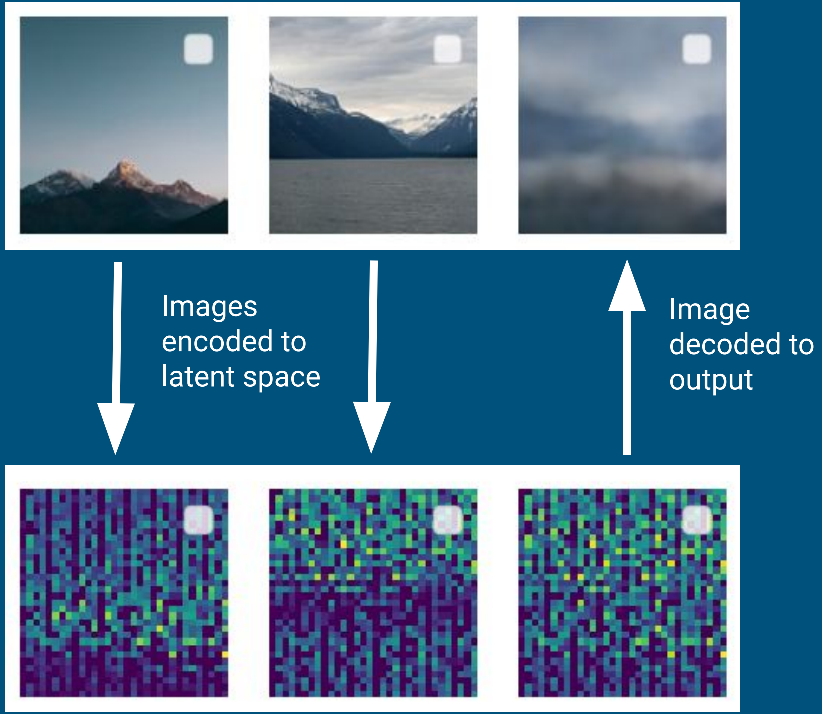


https://aws.amazon.com/blogs/machine-learning/deploying-variational-autoencoders-for-anomaly-detection-with-tensorflow-serving-on-amazon-sagemaker/

Decision Boundary

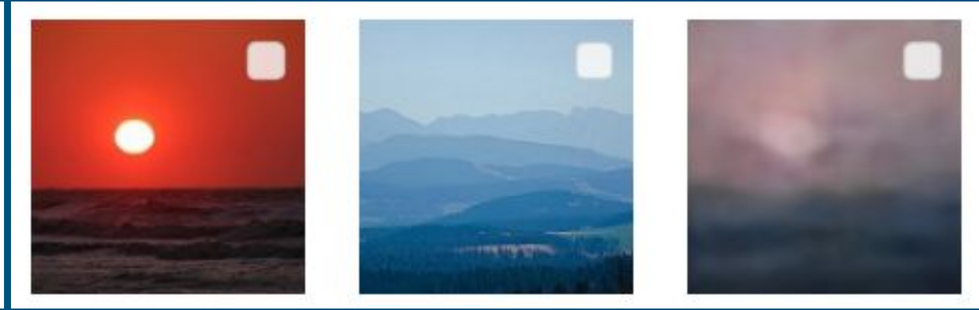
Averaging two latent spaces together





Notice: ordering of pixels in latent space does not matter it was reshaped into this format for display purposes

Average of two pictures

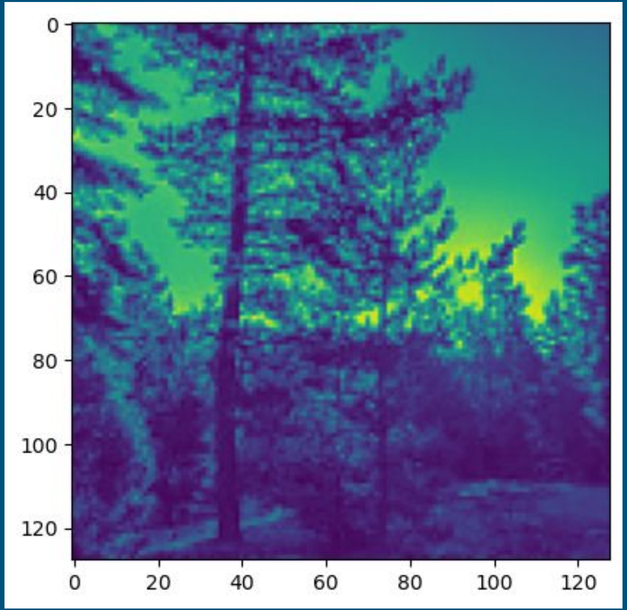


Advantages / Disadvantages

- Advantages
    - Able to handle more complex relationships (non-linear)
    - Not as ‘lossy’ as PCA
    - Clear metric to evaluate the model
    - Less computationally expensive compared to a dense network
- Disadvantages
    - Slow training process (Our autoencoder took about 90 minutes for 50 epochs)
    - A lot of model and hyperparameter tuning
    - Encoding into very low dimensions (PCA or TSNE is better at lower dimensions <50)
    - Requires lots of data

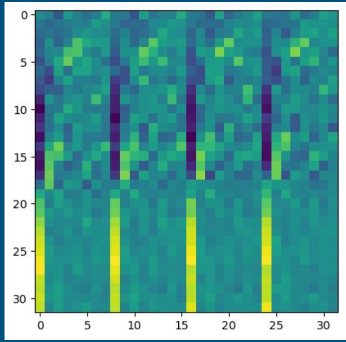
PCA compression

PCA Input Image



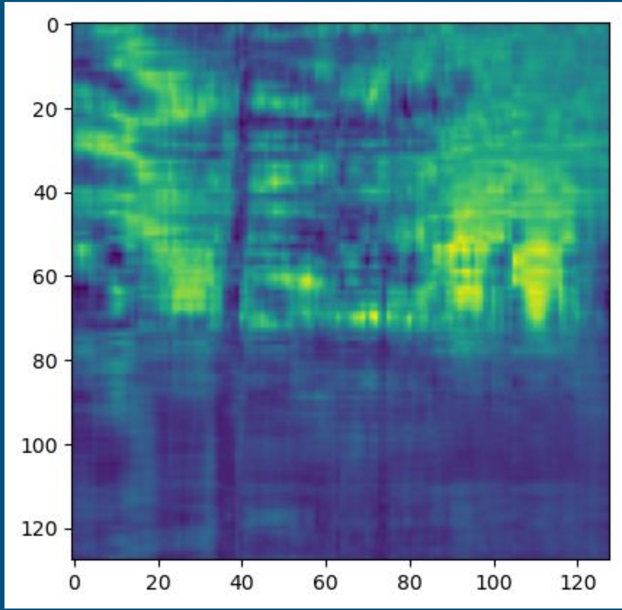
128 x 128 x 1

Encoding



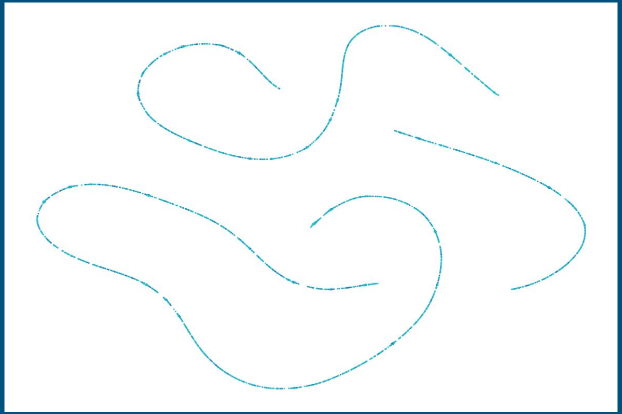
32 x 32 x 1

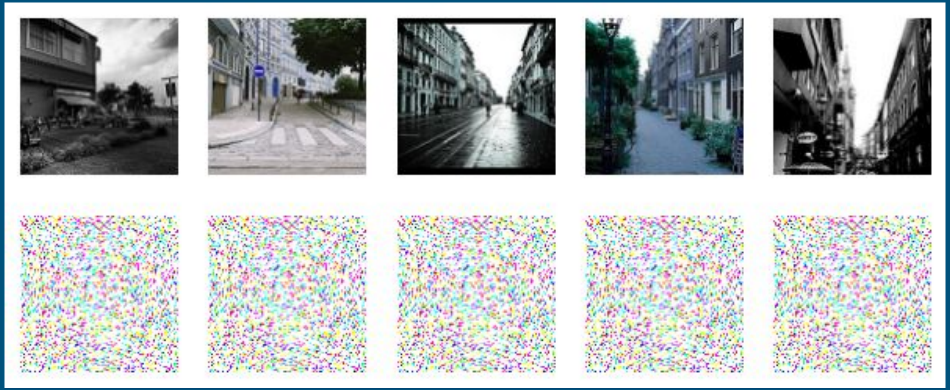
PCA Inverse Transform



128 x 128 x 1

Fails while making this project





Sources

- https://www.v7labs.com/blog/autoencoders-guide
- https://www.kaggle.com/datasets/puneet6060/intel-image-classification
- https://towardsdatascience.com/dimensionality-reduction-pca-versus-autoencoders-338fcaf3297d
- https://keras.io/guides/transfer_learning/
- https://aws.amazon.com/blogs/machine-learning/deploying-variational-autoencoders-for-anomaly-detection-with-tensorflow-serving-on-amazon-sagemaker/
- https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726
- https://www.kaggle.com/code/abdelrhamanfakhry/cnn-data-augmentation-early-stop-lr-reduction
-
- MSOE AI Club