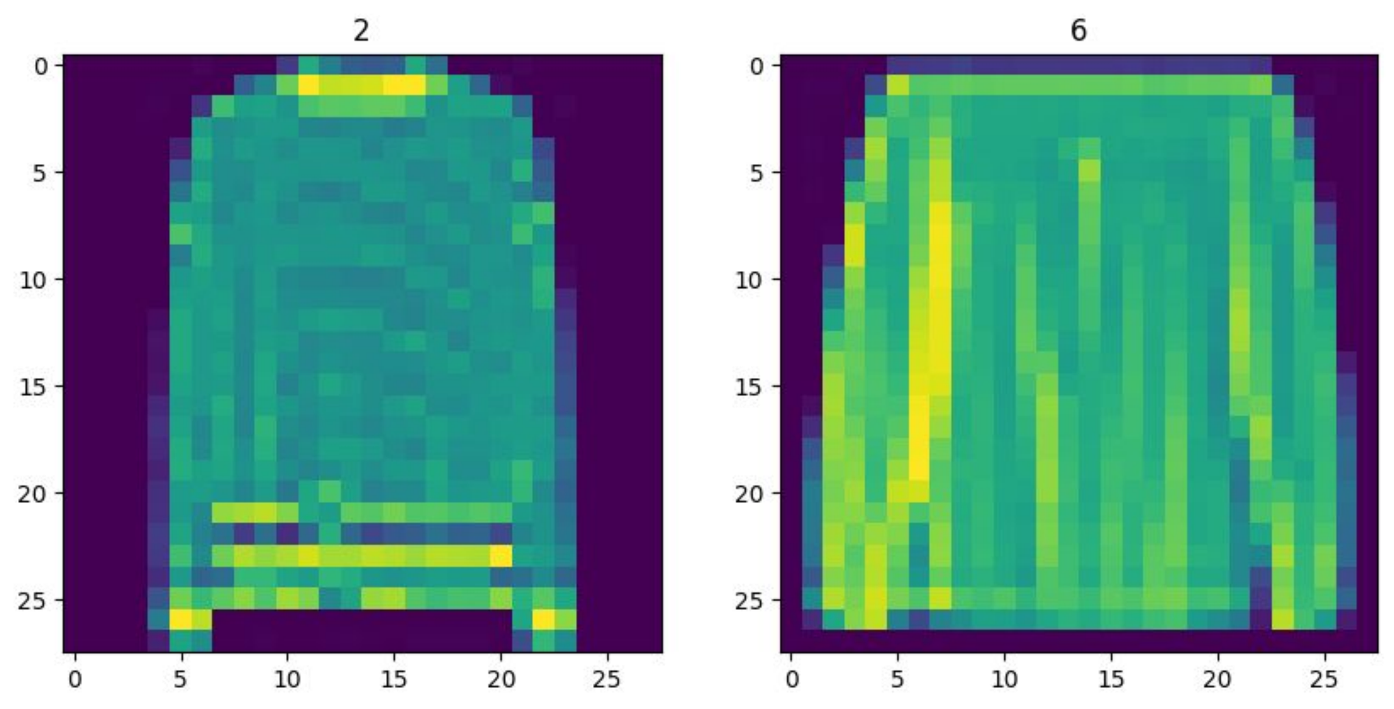
Deep Learning Final Exam

Kaggle : Image Generation Model

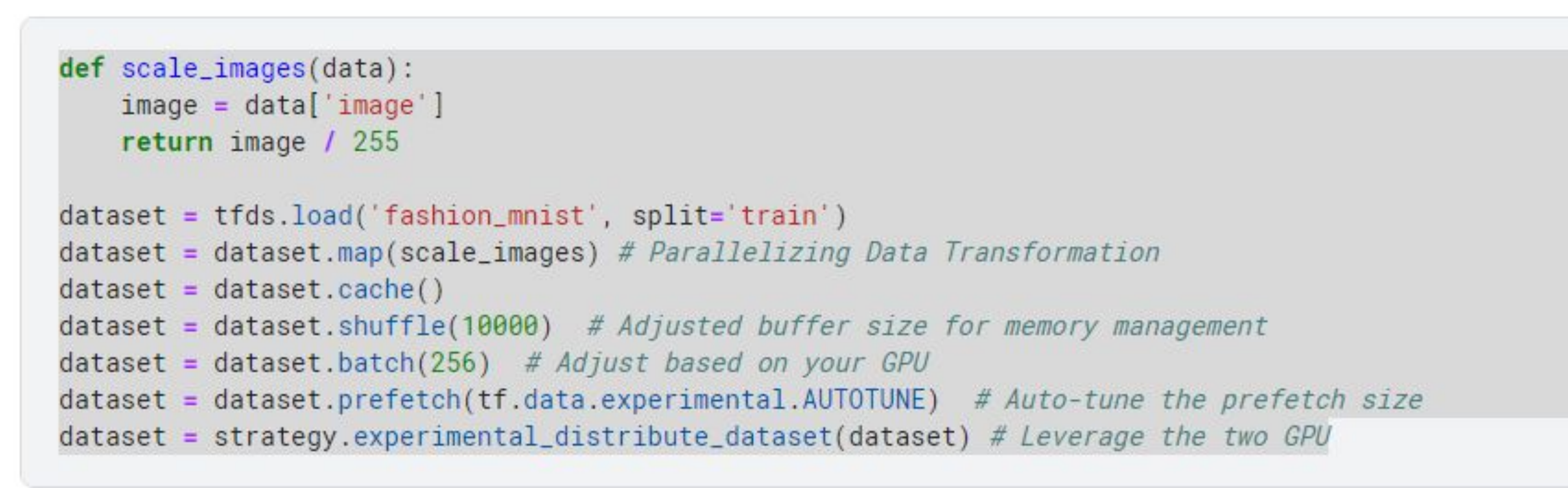
- Goal is to build a GAN that can generate/discriminate images from e-commerce clothing store.
- Dataset : fashion MNIST

- Challenges :
    - Limited amount of computing resources

What’s the data look like ?



Pre-processing



```python
def scale_images(data):
    image = data['image']
    return image / 255

dataset = tfds.load('fashion_mnist', split='train')
dataset = dataset.map(scale_images) # Parallelizing Data Transformation
dataset = dataset.cache()
dataset = dataset.shuffle(10000)  # Adjusted buffer size for memory management
dataset = dataset.batch(256)  # Adjust based on your GPU
dataset = dataset.prefetch(tf.data.experimental.AUTOTUNE)  # Auto-tune the prefetch size
dataset = strategy.experimental_distribute_dataset(dataset) # Leverage the two GPU
```

Hardware challenges

- Started with google Colab but quickly limited

- Managed to have access to NVIDIA Tesla P4 GPU but the cooling fan was nto strong enough to prevent overheating during training

- Found out that Kaggle allow for 30hrs of usage of 2 T4 freely

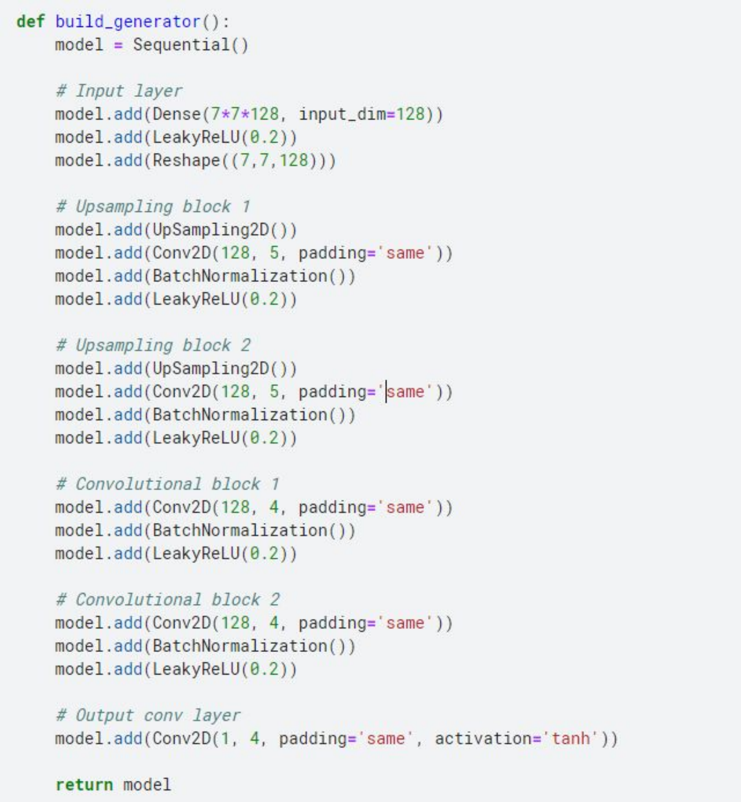
Training challenge

- Not owning the hardware and large training time make it more difficult perform some trial & errors approach

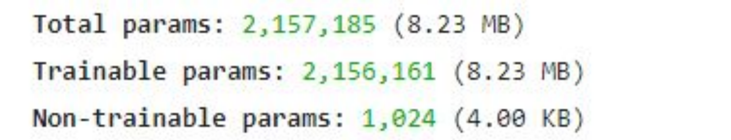
- Even after 12 hrs of training our 1st model, the results were not satisfying

- Decided to change the approach and to finetune an existing model (ResNet50) instead of creating it from scratch

Initial generator

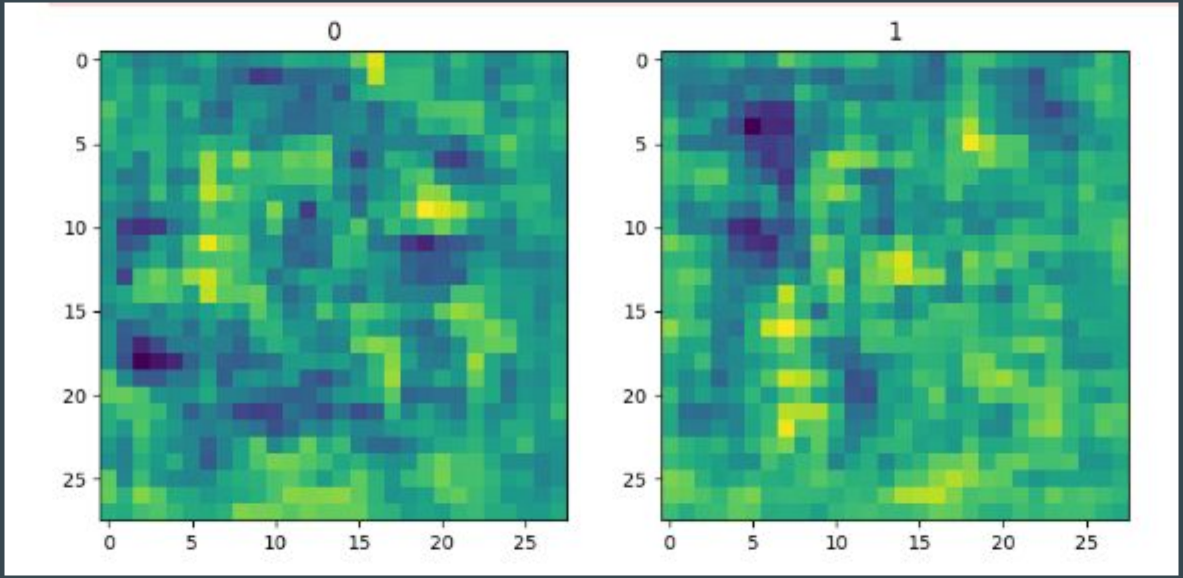


```python
def build_generator():
    model = Sequential()

    # Input layer
    model.add(Dense(7*7*128, input_dim=128))
    model.add(LeakyReLU(0.2))
    model.add(Reshape((7,7,128)))

    # Upsampling block 1
    model.add(UpSampling2D())
    model.add(Conv2D(128, 5, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(0.2))

    # Upsampling block 2
    model.add(UpSampling2D())
    model.add(Conv2D(128, 5, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(0.2))

    # Convolutional block 1
    model.add(Conv2D(128, 4, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(0.2))

    # Convolutional block 2
    model.add(Conv2D(128, 4, padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(0.2))

    # Output conv layer
    model.add(Conv2D(1, 4, padding='same', activation='tanh'))

    return model
```

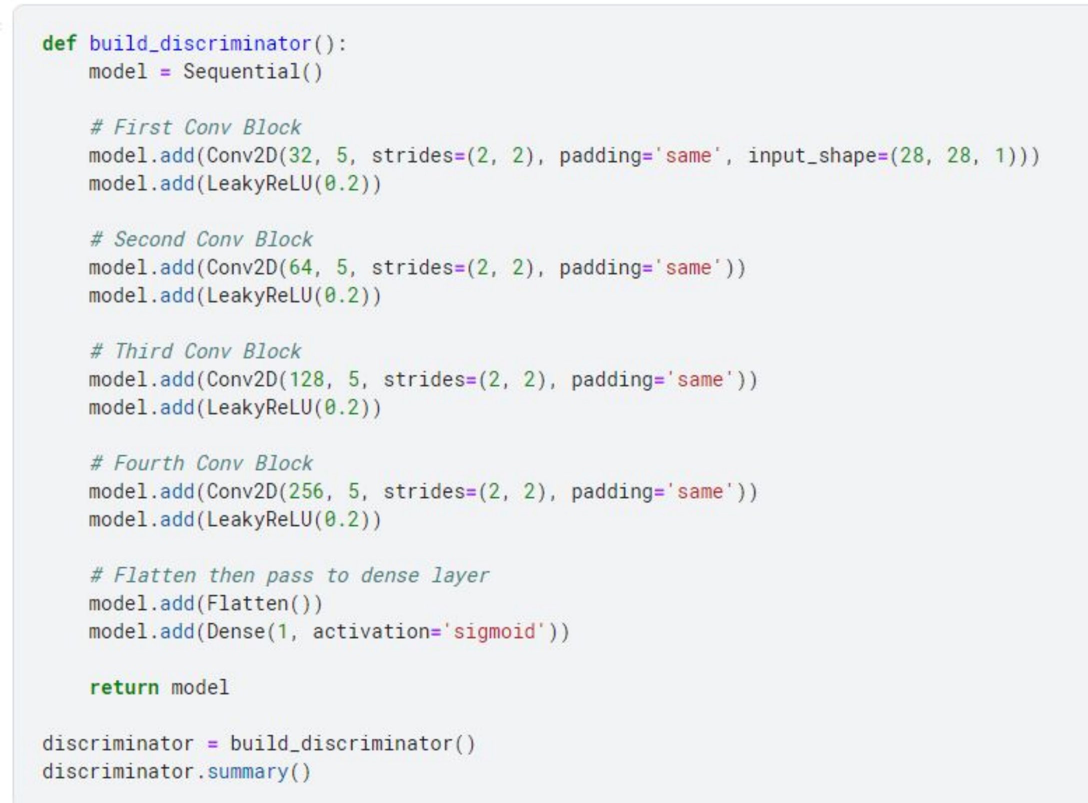


```
Total params: 2,157,185 (8.23 MB)
Trainable params: 2,156,161 (8.23 MB)
Non-trainable params: 1,024 (4.00 KB)
```

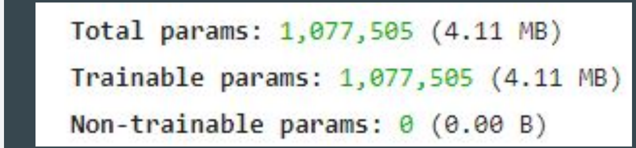
Generator output (no training)



Initial discriminator

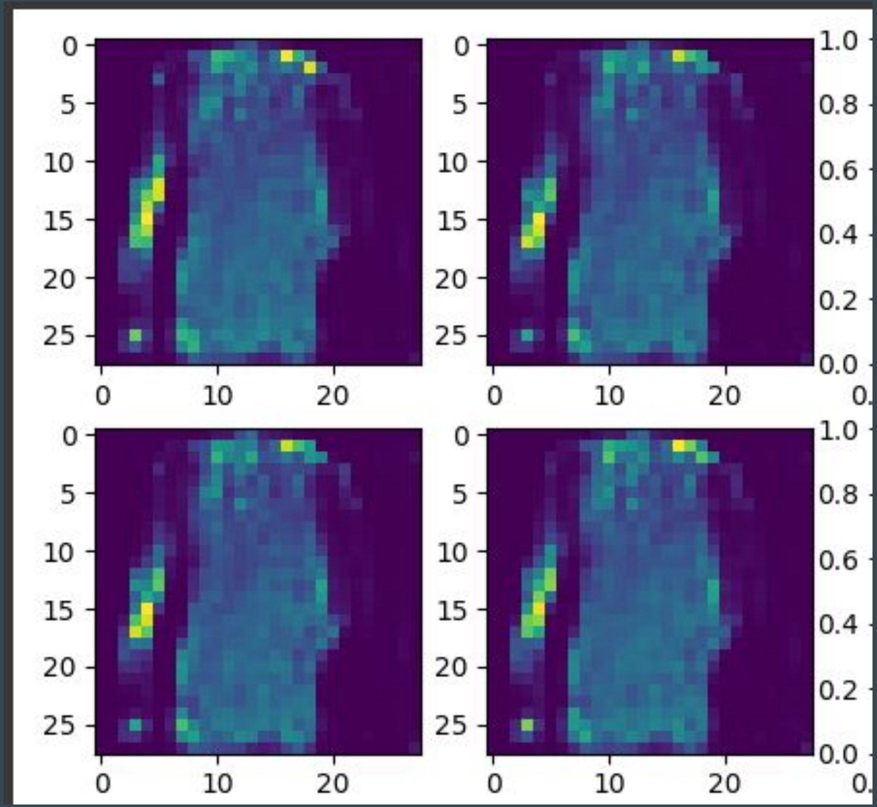


```python
def build_discriminator():
    model = Sequential()

    # First Conv Block
    model.add(Conv2D(32, 5, strides=(2, 2), padding='same', input_shape=(28, 28, 1)))
    model.add(LeakyReLU(0.2))

    # Second Conv Block
    model.add(Conv2D(64, 5, strides=(2, 2), padding='same'))
    model.add(LeakyReLU(0.2))

    # Third Conv Block
    model.add(Conv2D(128, 5, strides=(2, 2), padding='same'))
    model.add(LeakyReLU(0.2))

    # Fourth Conv Block
    model.add(Conv2D(256, 5, strides=(2, 2), padding='same'))
    model.add(LeakyReLU(0.2))

    # Flatten then pass to dense layer
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))

    return model

discriminator = build_discriminator()
discriminator.summary()
```

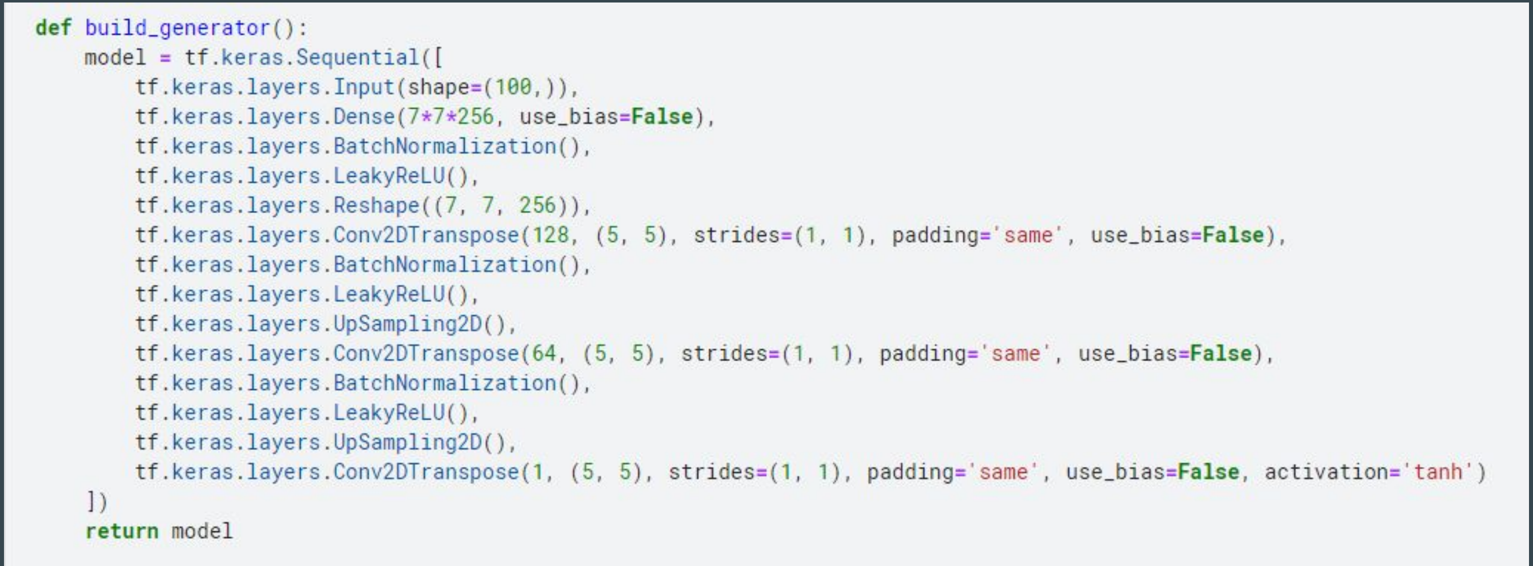


```
Total params: 1,077,505 (4.11 MB)
Trainable params: 1,077,505 (4.11 MB)
Non-trainable params: 0 (0.00 B)
```

Final image generation after ~18hrs of training

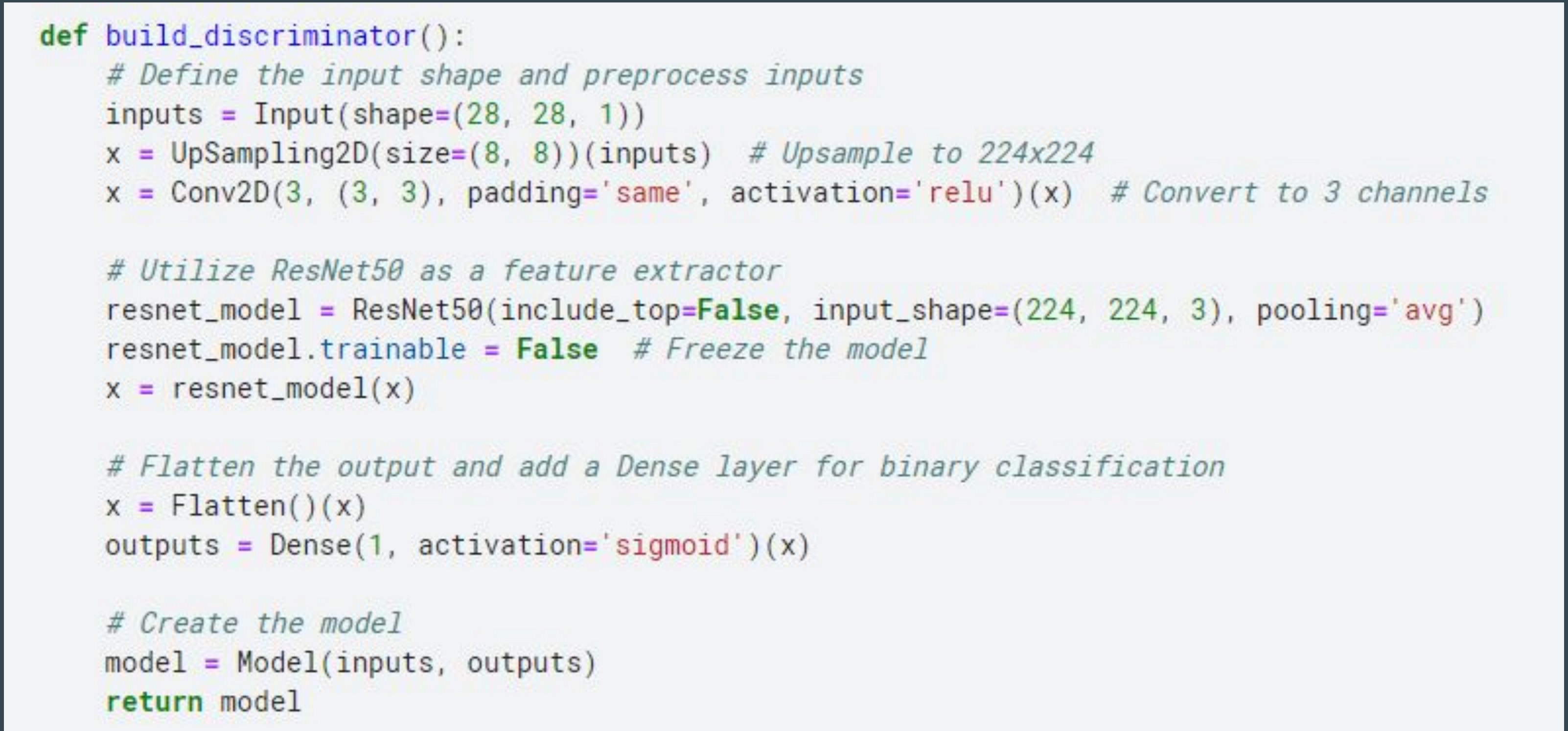


Second generator



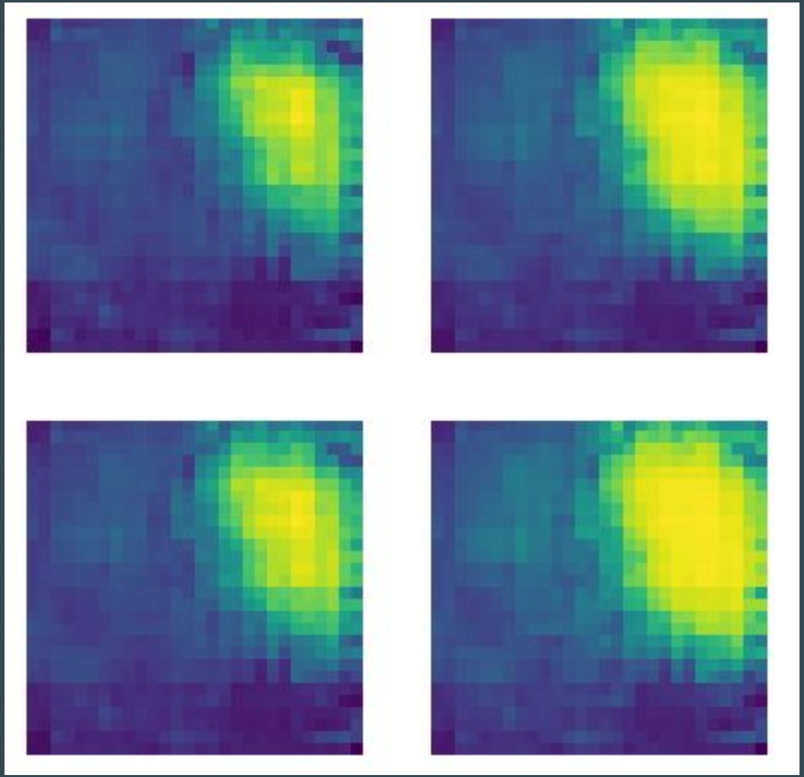
```python
def build_generator():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(100,)),
        tf.keras.layers.Dense(7*7*256, use_bias=False),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.LeakyReLU(),
        tf.keras.layers.Reshape((7, 7, 256)),
        tf.keras.layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.LeakyReLU(),
        tf.keras.layers.UpSampling2D(),
        tf.keras.layers.Conv2DTranspose(64, (5, 5), strides=(1, 1), padding='same', use_bias=False),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.LeakyReLU(),
        tf.keras.layers.UpSampling2D(),
        tf.keras.layers.Conv2DTranspose(1, (5, 5), strides=(1, 1), padding='same', use_bias=False, activation='tanh')
    ])
    return model
```

Second discriminator



```
def build_discriminator():
    # Define the input shape and preprocess inputs
    inputs = Input(shape=(28, 28, 1))
    x = UpSampling2D(size=(8, 8))(inputs)  # Upsample to 224x224
    x = Conv2D(3, (3, 3), padding='same', activation='relu')(x)  # Convert to 3 channels

    # Utilize ResNet50 as a feature extractor
    resnet_model = ResNet50(include_top=False, input_shape=(224, 224, 3), pooling='avg')
    resnet_model.trainable = False  # Freeze the model
    x = resnet_model(x)

    # Flatten the output and add a Dense layer for binary classification
    x = Flatten()(x)
    outputs = Dense(1, activation='sigmoid')(x)

    # Create the model
    model = Model(inputs, outputs)
    return model
```

Results after ~4hrs of training



Conclusion

- Hardware/Computing power and training time are a major hurdle in training and deploying deep learning model

Areas of improvement

- Go back to initial architecture without using pre-trained model
- Setup a more powerful NN training lab
- Give more training time to the model
- Experiment with different architecture for generator and discriminator