p4

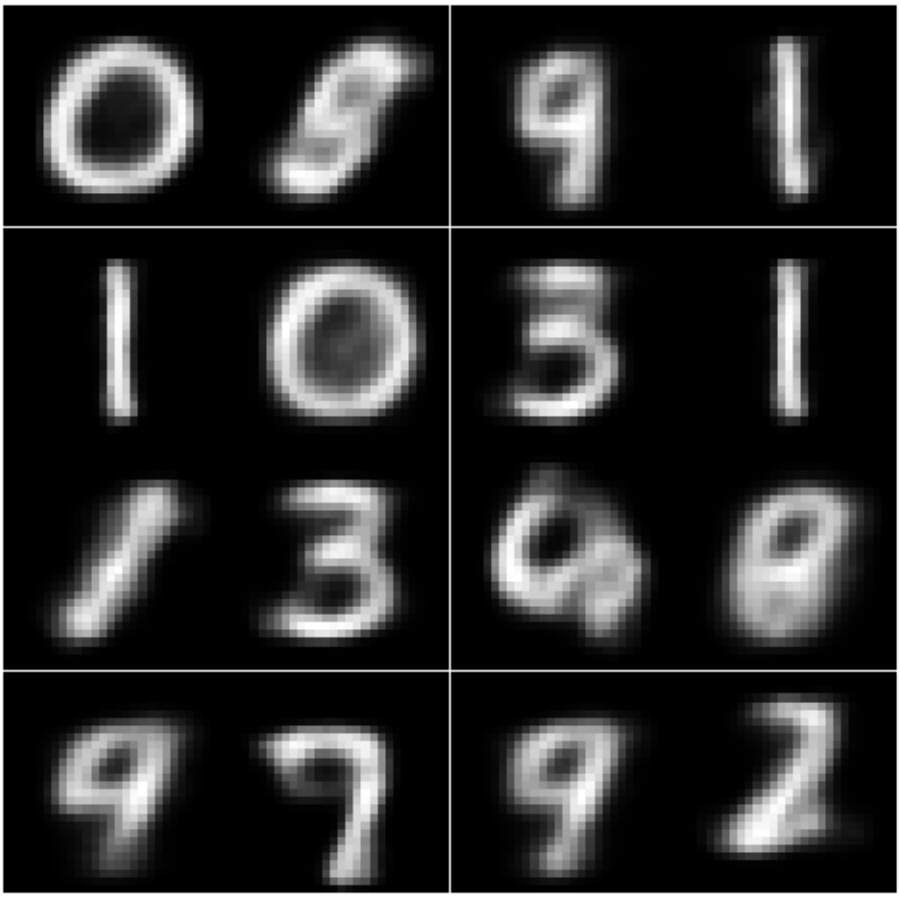
November 7, 2024

[1]:
```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Dense, Input, Flatten, Reshape, LeakyReLU,␣
 ↪Activation, Dropout
from tensorflow.keras.models import Model, Sequential
from matplotlib import pyplot as plt
from IPython import display # If using IPython, Colab, or Jupyter
import numpy as np

# Load MNIST data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0

# Plot an example image from x_train
plt.imshow(x_train[0], cmap="gray")
plt.show()

LATENT_SIZE = 32

# Define encoder model
encoder = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(512),
    LeakyReLU(),
    Dropout(0.5),
    Dense(256),
    LeakyReLU(),
    Dropout(0.5),
    Dense(128),
    LeakyReLU(),
    Dropout(0.5),
    Dense(64),
    LeakyReLU(),
    Dropout(0.5),
    Dense(LATENT_SIZE),
    LeakyReLU()
])
```

```
# Define decoder model
decoder = Sequential([
    Dense(64, input_shape=(LATENT_SIZE,)),
    LeakyReLU(),
    Dropout(0.5),
    Dense(128),
    LeakyReLU(),
    Dropout(0.5),
    Dense(256),
    LeakyReLU(),
    Dropout(0.5),
    Dense(512),
    LeakyReLU(),
    Dropout(0.5),
    Dense(784),
    Activation("sigmoid"),
    Reshape((28, 28))
])

# Build autoencoder model
img = Input(shape=(28, 28))
latent_vector = encoder(img)
output = decoder(latent_vector)
model = Model(inputs=img, outputs=output)

# Compile the model
model.compile(optimizer="nadam", loss="binary_crossentropy")

EPOCHS = 60

# Training and visualization loop
for epoch in range(EPOCHS):
    fig, axs = plt.subplots(4, 4, figsize=(6, 6))
    rand = x_test[np.random.randint(0, 10000, 16)].reshape((16, 28, 28))
    display.clear_output(wait=True)  # If you imported display from IPython

    for i in range(4):
        for j in range(4):
            axs[i, j].imshow(model.predict(rand[i * 4 + j].reshape(1, 28,␣
 ↪28))[0], cmap="gray")
            axs[i, j].axis("off")

    plt.subplots_adjust(wspace=0, hspace=0)
    plt.show()

    print("-----------", "EPOCH", epoch + 1, "-----------")
```

```
# Train the model
model.fit(x_train, x_train, epochs=1, verbose=1)
```

```
1/1             0s 33ms/step
1/1             0s 32ms/step
1/1             0s 31ms/step
1/1             0s 29ms/step
1/1             0s 30ms/step
1/1             0s 29ms/step
1/1             0s 35ms/step
1/1             0s 35ms/step
1/1             0s 35ms/step
1/1             0s 31ms/step
1/1             0s 32ms/step
1/1             0s 42ms/step
1/1             0s 90ms/step
1/1             0s 75ms/step
1/1             0s 35ms/step
1/1             0s 31ms/step
```



```
---------- EPOCH 3 ----------
 308/1875            31s 20ms/step -
loss: 0.2057
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-1-e57ad80f86c2> in <cell line: 69>()
     83
     84     # Train the model
---> 85     model.fit(x_train, x_train, epochs=1, verbose=1)

/usr/local/lib/python3.10/dist-packages/keras/src/utils/traceback_utils.py in␣
 ↪error_handler(*args, **kwargs)
    115         filtered_tb = None
    116         try:
--> 117             return fn(*args, **kwargs)
    118         except Exception as e:
    119             filtered_tb = _process_traceback_frames(e.__traceback__)

/usr/local/lib/python3.10/dist-packages/keras/src/backend/tensorflow/trainer.py␣
 ↪in fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split,␣
 ↪validation_data, shuffle, class_weight, sample_weight, initial_epoch,␣
 ↪steps_per_epoch, validation_steps, validation_batch_size, validation_freq)
    316                 for step, iterator in epoch_iterator.enumerate_epoch():
    317                     callbacks.on_train_batch_begin(step)
--> 318                     logs = self.train_function(iterator)
    319                     logs = self._pythonify_logs(logs)
    320                     callbacks.on_train_batch_end(step, logs)

/usr/local/lib/python3.10/dist-packages/tensorflow/python/util/traceback_utils.
 ↪py in error_handler(*args, **kwargs)
    148     filtered_tb = None
    149     try:
--> 150       return fn(*args, **kwargs)
    151     except Exception as e:
    152       filtered_tb = _process_traceback_frames(e.__traceback__)

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/polymorphic_function.py in __call__(self, *args, **kwds)
    831
    832       with OptionalXlaContext(self._jit_compile):
--> 833         result = self._call(*args, **kwds)
    834
    835       new_tracing_count = self.experimental_get_tracing_count()
```

```
/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/polymorphic_function.py in _call(self, *args, **kwds)
    876       # In this case we have not created variables on the first call. S␣
 ↪we can
    877       # run the first trace but we should fail if variables are created
--> 878       results = tracing_compilation.call_function(
    879           args, kwds, self._variable_creation_config
    880       )

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/tracing_compilation.py in call_function(args, kwargs,␣
 ↪tracing_options)
    137   bound_args = function.function_type.bind(*args, **kwargs)
    138   flat_inputs = function.function_type.unpack_inputs(bound_args)
--> 139   return function._call_flat(  # pylint: disable=protected-access
    140       flat_inputs, captured_inputs=function.captured_inputs
    141   )

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/concrete_function.py in _call_flat(self, tensor_inputs,␣
 ↪captured_inputs)
   1320         and executing_eagerly):
   1321       # No tape is watching; skip to running the function.
-> 1322       return self._inference_function.call_preflattened(args)
   1323     forward_backward = self._select_forward_and_backward_functions(
   1324         args,

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/atomic_function.py in call_preflattened(self, args)
    214   def call_preflattened(self, args: Sequence[core.Tensor]) -> Any:
    215     """Calls with flattened tensor inputs and returns the structured␣
 ↪output."""
--> 216     flat_outputs = self.call_flat(*args)
    217     return self.function_type.pack_output(flat_outputs)
    218

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/
 ↪polymorphic_function/atomic_function.py in call_flat(self, *args)
    249         with record.stop_recording():
    250           if self._bound_context.executing_eagerly():
--> 251             outputs = self._bound_context.call_function(
    252                 self.name,
    253                 list(args),

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/context.py in␣
 ↪call_function(self, name, tensor_inputs, num_outputs)
   1550     cancellation_context = cancellation.context()
```

```
   1551     if cancellation_context is None:
-> 1552       outputs = execute.execute(

   1553           name.decode("utf-8"),
   1554           num_outputs=num_outputs,

/usr/local/lib/python3.10/dist-packages/tensorflow/python/eager/execute.py in␣
 ↪quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     51   try:
     52     ctx.ensure_initialized()
---> 53     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name,␣
 ↪op_name,

     54                                         inputs, attrs, num_outputs)
     55   except core._NotOkStatusException as e:

KeyboardInterrupt:
```