Tanawin-st123975-auto-encoder

November 12, 2023

[1]:
```
!nvidia-smi
```

```
Sun Nov 12 17:04:55 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   44C    P8    10W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

[2]:
```
# # Import and creating some helper functions
# import numpy as np
# import tensorflow as tf
# import matplotlib.pyplot as plt
# import copy
# from tensorflow.keras import layers
# from tensorflow.keras.datasets import mnist
# from tensorflow.keras.models import Model


# def preprocess(array):
#     """
#     Normalizes the supplied array and reshapes it into the appropriate format.
#     """
```

```
#     array = array.astype("float32") / 255.0
#     array = np.reshape(array, (len(array), 28, 28, 1))
#     return array


# def occlude(array):
#     """
#     Adds occlusion.
#     """
#     new_array = copy.deepcopy( array )
#     print(new_array.shape)
#     new_array[:,10:13,:] = 1.0

#     return new_array


# def display(array1, array2):
#     """
#     Displays ten random images from each one of the supplied arrays.
#     """

#     n = 10

#     indices = np.random.randint(len(array1), size=n)
#     images1 = array1[indices, :]
#     images2 = array2[indices, :]

#     plt.figure(figsize=(20, 4))
#     for i, (image1, image2) in enumerate(zip(images1, images2)):
#         ax = plt.subplot(2, n, i + 1)
#         plt.imshow(image1.reshape(28, 28))
#         plt.gray()
#         ax.get_xaxis().set_visible(False)
#         ax.get_yaxis().set_visible(False)

#         ax = plt.subplot(2, n, i + 1 + n)
#         plt.imshow(image2.reshape(28, 28))
#         plt.gray()
#         ax.get_xaxis().set_visible(False)
#         ax.get_yaxis().set_visible(False)

#     plt.show()



# # Since we only need images from the dataset to encode and decode, we
# # won't use the labels.
```

```python
# (train_data, _), (test_data, _) = mnist.load_data()

# # Normalize and reshape the data
# train_data = preprocess(train_data)
# test_data = preprocess(test_data)

# # Create a copy of the data with added noise
# noisy_train_data = occlude(train_data)
# noisy_test_data = occlude(test_data)

# # Display the train data and a version of it with added noise
# display(train_data, noisy_train_data)
```

[3]:
```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import copy
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

[4]:
```python
def preprocess(array):
    array = array.astype("float32") / 255.0
    array = np.reshape(array, (len(array), 28, 28, 1))
    return array

def occlude(array):
    new_array = copy.deepcopy(array)
    new_array[:, 10:13, :] = 1.0
    return new_array

def display(array1, array2, array3, labels=['Actual', 'Occluded',␣
 ↪'Reconstructed']):
    n = 10
    indices = np.random.randint(len(array1), size=n)
    images1 = array1[indices, :]
    images2 = array2[indices, :]
    images3 = array3[indices, :]

    plt.figure(figsize=(20, 8))
    for i, (image1, image2, image3) in enumerate(zip(images1, images2,␣
 ↪images3)):
        ax = plt.subplot(3, n, i + 1)
        plt.imshow(image1.reshape(28, 28))
        plt.gray()
```

```
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        ax.set_title(labels[0])

        ax = plt.subplot(3, n, i + 1 + n)
        plt.imshow(image2.reshape(28, 28))
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        ax.set_title(labels[1])

        ax = plt.subplot(3, n, i + 1 + 2 * n)
        plt.imshow(image3.reshape(28, 28))
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        ax.set_title(labels[2])

    plt.show()


(train_data, _), (test_data, _) = mnist.load_data()

# Normalize and reshape the data
train_data = preprocess(train_data)
test_data = preprocess(test_data)

# Create a copy of the data with added noise
noisy_train_data = occlude(train_data)
noisy_test_data = occlude(test_data)

# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=False,
    vertical_flip=False,
    fill_mode='nearest'
)

# (train_data, _), (test_data, _) = mnist.load_data()

# # Normalize and reshape the data
# train_data = preprocess(train_data)
```

```
# test_data = preprocess(test_data)

# # Create a copy of the data with added noise
# noisy_train_data = occlude(train_data)
# noisy_test_data = occlude(test_data)

# # Define the autoencoder model
# input_img = layers.Input(shape=(28, 28, 1))
# encoded = layers.Conv2D(32, (3, 3), activation='relu',␣
↪padding='same')(input_img)
# encoded = layers.MaxPooling2D((2, 2), padding='same')(encoded)
# encoded = layers.Conv2D(64, (3, 3), activation='relu',␣
↪padding='same')(encoded)
# encoded = layers.MaxPooling2D((2, 2), padding='same')(encoded)

# decoded = layers.Conv2D(64, (3, 3), activation='relu',␣
↪padding='same')(encoded)
# decoded = layers.UpSampling2D((2, 2))(decoded)
# decoded = layers.Conv2D(32, (3, 3), activation='relu',␣
↪padding='same')(decoded)
# decoded = layers.UpSampling2D((2, 2))(decoded)
# decoded = layers.Conv2D(1, (3, 3), activation='sigmoid',␣
↪padding='same')(decoded)

# autoencoder = Model(input_img, decoded)
# autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# # Train the autoencoder on clean images
# autoencoder.fit(train_data, train_data, epochs=10, batch_size=128,␣
↪shuffle=True, validation_data=(test_data, test_data))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
```

[5]:

```
# Define the autoencoder model
# input_img = layers.Input(shape=(28, 28, 1))
# encoded = layers.Conv2D(32, (3, 3), activation='relu',␣
↪padding='same')(input_img)
# encoded = layers.MaxPooling2D((2, 2), padding='same')(encoded)
# encoded = layers.Conv2D(64, (3, 3), activation='relu',␣
↪padding='same')(encoded)
# encoded = layers.MaxPooling2D((2, 2), padding='same')(encoded)

# decoded = layers.Conv2D(64, (3, 3), activation='relu',␣
↪padding='same')(encoded)
```

```
# decoded = layers.UpSampling2D((2, 2))(decoded)
# decoded = layers.Conv2D(32, (3, 3), activation='relu',␣
↪padding='same')(decoded)
# decoded = layers.UpSampling2D((2, 2))(decoded)
# decoded = layers.Conv2D(1, (3, 3), activation='sigmoid',␣
↪padding='same')(decoded)

input_img = layers.Input(shape=(28, 28, 1))
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)


x = layers.Dropout(0.25)(x)

encoded = layers.Flatten()(x)
decoded = layers.Reshape((7, 7, 64))(encoded)

decoded = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(decoded)
decoded = layers.UpSampling2D((2, 2))(decoded)
decoded = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(decoded)
decoded = layers.UpSampling2D((2, 2))(decoded)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid',␣
↪padding='same')(decoded)

autoencoder = Model(input_img, decoded)
sgd_optimizer = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
# autoencoder.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),␣
↪loss='binary_crossentropy')

autoencoder.compile(optimizer=sgd_optimizer, loss='binary_crossentropy',␣
↪metrics=['accuracy'])



early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',␣
↪patience=3, restore_best_weights=True)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.
↪2, patience=2, min_lr=1e-6)


history = autoencoder.fit(
    datagen.flow(train_data, train_data, batch_size=128),
```

```
    epochs=2000,
    steps_per_epoch=len(train_data) // 128,
    validation_data=(test_data, test_data),
    callbacks=[early_stopping, reduce_lr]
)


autoencoder.save_weights('autoencoder_weights.h5')
```

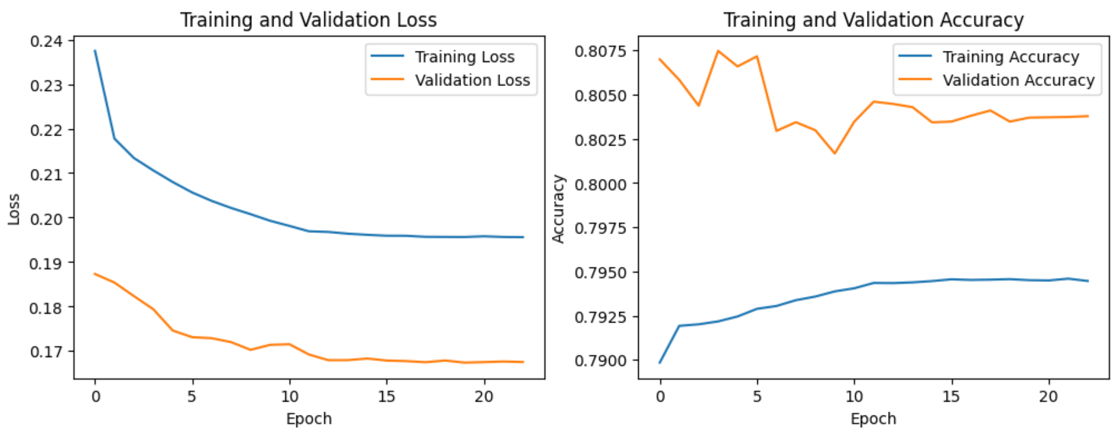
```
Epoch 1/2000
468/468 [==============================] - 29s 36ms/step - loss: 0.2375 -
accuracy: 0.7898 - val_loss: 0.1873 - val_accuracy: 0.8070 - lr: 0.0100
Epoch 2/2000
468/468 [==============================] - 16s 35ms/step - loss: 0.2178 -
accuracy: 0.7919 - val_loss: 0.1854 - val_accuracy: 0.8058 - lr: 0.0100
Epoch 3/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.2134 -
accuracy: 0.7920 - val_loss: 0.1824 - val_accuracy: 0.8044 - lr: 0.0100
Epoch 4/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.2106 -
accuracy: 0.7922 - val_loss: 0.1794 - val_accuracy: 0.8075 - lr: 0.0100
Epoch 5/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.2080 -
accuracy: 0.7925 - val_loss: 0.1746 - val_accuracy: 0.8066 - lr: 0.0100
Epoch 6/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.2056 -
accuracy: 0.7929 - val_loss: 0.1731 - val_accuracy: 0.8072 - lr: 0.0100
Epoch 7/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.2037 -
accuracy: 0.7930 - val_loss: 0.1729 - val_accuracy: 0.8029 - lr: 0.0100
Epoch 8/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.2022 -
accuracy: 0.7934 - val_loss: 0.1720 - val_accuracy: 0.8034 - lr: 0.0100
Epoch 9/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.2008 -
accuracy: 0.7936 - val_loss: 0.1703 - val_accuracy: 0.8030 - lr: 0.0100
Epoch 10/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.1993 -
accuracy: 0.7939 - val_loss: 0.1714 - val_accuracy: 0.8017 - lr: 0.0100
Epoch 11/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.1981 -
accuracy: 0.7940 - val_loss: 0.1715 - val_accuracy: 0.8035 - lr: 0.0100
Epoch 12/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.1969 -
accuracy: 0.7943 - val_loss: 0.1692 - val_accuracy: 0.8046 - lr: 0.0020
Epoch 13/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.1968 -
```

```
accuracy: 0.7943 - val_loss: 0.1680 - val_accuracy: 0.8045 - lr: 0.0020
Epoch 14/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.1964 -
accuracy: 0.7944 - val_loss: 0.1680 - val_accuracy: 0.8043 - lr: 0.0020
Epoch 15/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.1961 -
accuracy: 0.7945 - val_loss: 0.1683 - val_accuracy: 0.8034 - lr: 0.0020
Epoch 16/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.1959 -
accuracy: 0.7946 - val_loss: 0.1679 - val_accuracy: 0.8035 - lr: 4.0000e-04
Epoch 17/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.1959 -
accuracy: 0.7945 - val_loss: 0.1678 - val_accuracy: 0.8038 - lr: 4.0000e-04
Epoch 18/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.1957 -
accuracy: 0.7945 - val_loss: 0.1675 - val_accuracy: 0.8041 - lr: 4.0000e-04
Epoch 19/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.1957 -
accuracy: 0.7946 - val_loss: 0.1679 - val_accuracy: 0.8035 - lr: 4.0000e-04
Epoch 20/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.1956 -
accuracy: 0.7945 - val_loss: 0.1674 - val_accuracy: 0.8037 - lr: 4.0000e-04
Epoch 21/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.1958 -
accuracy: 0.7945 - val_loss: 0.1675 - val_accuracy: 0.8037 - lr: 8.0000e-05
Epoch 22/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.1956 -
accuracy: 0.7946 - val_loss: 0.1677 - val_accuracy: 0.8037 - lr: 8.0000e-05
Epoch 23/2000
468/468 [==============================] - 16s 34ms/step - loss: 0.1956 -
accuracy: 0.7945 - val_loss: 0.1675 - val_accuracy: 0.8038 - lr: 1.6000e-05
```

[6]:

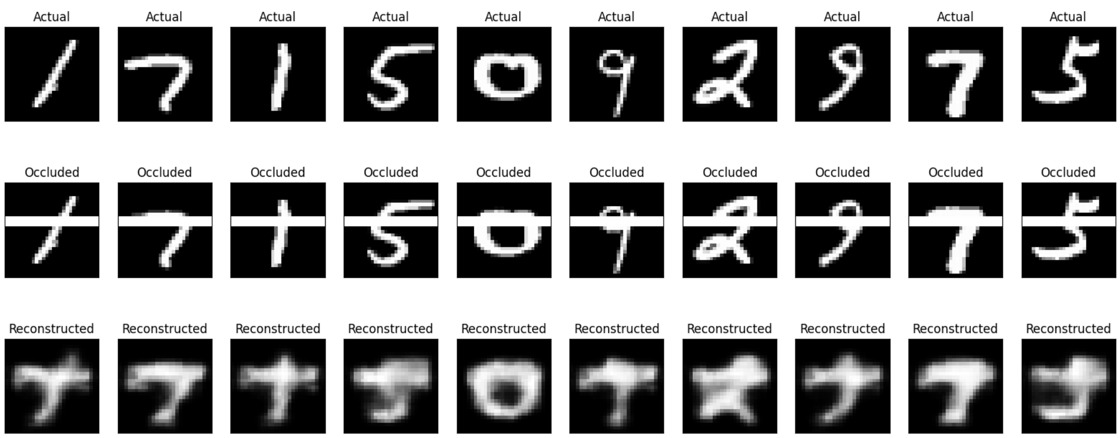
```python
training_loss = history.history['loss']
training_accuracy = history.history['accuracy']

validation_loss = history.history['val_loss']
validation_accuracy = history.history['val_accuracy']

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(training_loss, label='Training Loss')
plt.plot(validation_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

```
plt.subplot(1, 2, 2)
plt.plot(training_accuracy, label='Training Accuracy')
plt.plot(validation_accuracy, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



[7]:
```
decoded_images = autoencoder.predict(noisy_test_data)

display(test_data, noisy_test_data, decoded_images, labels=['Actual',␣
↪'Occluded', 'Reconstructed'])
```

```
313/313 [==============================] - 1s 2ms/step
```



[8]:
```
# fine_tune_autoencoder = Model(input_img, decoded)
# fine_tune_autoencoder.compile(optimizer=sgd_optimizer,␣
↪loss='binary_crossentropy')
# fine_tune_autoencoder.load_weights('autoencoder_weights.h5')

# # Freeze layers up to a specific layer for fine-tuning
# freeze_until_layer = 'name_of_layer_to_freeze'  # Replace with the actual␣
↪layer name
# for layer in fine_tune_autoencoder.layers:
#     layer.trainable = False
#     if layer.name == freeze_until_layer:
#         break


# fine_tune_autoencoder.compile(optimizer=sgd_optimizer,␣
↪loss='binary_crossentropy')
```

[9]:
```
# # Train the fine-tuned autoencoder
# fine_tune_history = fine_tune_autoencoder.fit(
#     datagen.flow(train_data, train_data, batch_size=128),
#     epochs=2000,
#     steps_per_epoch=len(train_data) // 128,
#     validation_data=(test_data, test_data),
#     callbacks=[early_stopping, reduce_lr]
# )
```

[10]:
```
# training_loss = fine_tune_history.history['loss']
# validation_loss = fine_tune_history.history['val_loss']

# plt.plot(training_loss, label='Training Loss')
# plt.plot(validation_loss, label='Validation Loss')
# plt.title('Training and Validation Loss')
# plt.xlabel('Epoch')
# plt.ylabel('Loss')
# plt.legend()
# plt.show()
```

[11]:
```
# decoded_images_fine_tuned = fine_tune_autoencoder.predict(noisy_test_data)

# display(test_data, noisy_test_data, decoded_images_fine_tuned,␣
↪labels=['Actual', 'Occluded', 'Reconstructed'])
```