

De La Salle University – Manila

Gokongwei College of Engineering

Department of Electronics and Computer Engineering

CpE Elective 3 Laboratory

LBYCPC4

Summative Laboratory Report

Prepared by:

Banal, Kenan A.

LBYCPC4 – EQ3

# TABLE OF CONTENTS

Table of Contents

[TABLE OF CONTENTS 2](#_Toc210069227)

[I. INTRODUCTION <Replace the guide text with your own introduction; Approx. 1-2 pages> 4](#_Toc210069228)

[II. METHODOLOGY, RESULTS, AND ANALYSIS 4](#_Toc210069229)

[Activity 1: Autoencoders 4](#_Toc210069230)

[1.1. Objectives 4](#_Toc210069231)

[1.2. Experimental Procedure 4](#_Toc210069232)

[1.3. Results and Analysis 5](#_Toc210069233)

[1.4. Answer to Guide Questions, Insights, and Reflections 24](#_Toc210069234)

[Activity 2: <Title> 25](#_Toc210069235)

[2.1. Objectives 25](#_Toc210069236)

[2.2. Experimental Procedure 25](#_Toc210069237)

[2.3. Results and Analysis 25](#_Toc210069238)

[2.4. Answer to Guide Questions, Insights, and Reflections 25](#_Toc210069239)

[<Add more activity sections as needed> 25](#_Toc210069240)

[Activity 3: <Title> 26](#_Toc210069241)

[1.1. Objectives 26](#_Toc210069242)

[1.2. Experimental Procedure 26](#_Toc210069243)

[1.3. Results and Analysis 26](#_Toc210069244)

[1.4. Answer to Guide Questions, Insights, and Reflections 26](#_Toc210069245)

[<Add more activity sections as needed> 26](#_Toc210069246)

[Activity 4: <Title> 27](#_Toc210069247)

[1.1. Objectives 27](#_Toc210069248)

[1.2. Experimental Procedure 27](#_Toc210069249)

[1.3. Results and Analysis 27](#_Toc210069250)

[1.4. Answer to Guide Questions, Insights, and Reflections 27](#_Toc210069251)

[<Add more activity sections as needed> 27](#_Toc210069252)

[Activity 5: <Title> 28](#_Toc210069253)

[1.1. Objectives 28](#_Toc210069254)

[1.2. Experimental Procedure 28](#_Toc210069255)

[1.3. Results and Analysis 28](#_Toc210069256)

[1.4. Answer to Guide Questions, Insights, and Reflections 28](#_Toc210069257)

[<Add more activity sections as needed> 28](#_Toc210069258)

[Activity 6: <Title> 29](#_Toc210069259)

[1.1. Objectives 29](#_Toc210069260)

[1.2. Experimental Procedure 29](#_Toc210069261)

[1.3. Results and Analysis 29](#_Toc210069262)

[1.4. Answer to Guide Questions, Insights, and Reflections 29](#_Toc210069263)

[<Add more activity sections as needed> 29](#_Toc210069264)

[II. CONCLUSION AND OVERALL UNDERSTANDING <Replace the guide text with your own conclusions. Approx. 1-2 pages> 30](#_Toc210069265)

[III. REFERENCES <Use APA Citation Style> 30](#_Toc210069266)

# INTRODUCTION <Replace the guide text with your own introduction; Approx. 1-2 pages>

<This report serves as a summative assessment of your understanding and application of the concepts and theory in the design and development of generative deep learning models and the deployment of deep learning models on a single-board computer. It should demonstrate your ability to analyze problems, design and implement working models, and critically reflect on your learning.

In this section, write a complete introduction with the following guidelines:

* State the purpose of this report – e.g., to demonstrate your understanding and application of concepts learned in all laboratory activities.
* Discuss all the concepts and theories you have learned. Ideally, there should be one paragraph that discusses the concepts covered for each laboratory activity. Provide in-text citations that support the statements you have written in relation to the concepts or theories learned.
* In-text citations should refer to reputable sources and should have a corresponding entry in the references section.

>

# METHODOLOGY, RESULTS, AND ANALYSIS

## Activity 1: Autoencoders

### Objectives

1. Understand the working principle and architecture of autoencoders.
2. Build and train an autoencoder using deep learning framework.
3. Visualize the output of an autoencoder
4. Assess the performance of the implemented autoencoder

### Experimental Procedure

**A. Dimensionality Reduction with Autoencoders**

1. Download the Wine dataset from the UCI ML Repository and load it using Scikit-Learn.
2. Normalize the dataset features using MinMaxScaler.
3. Split the dataset into training (80%) and testing (20%) sets.
4. Build an autoencoder model using Keras Sequential API based on the given architecture.
5. Compile the model with Adam optimizer and mean squared error loss.
6. Train the autoencoder with a batch size of 10, using early stopping until validation loss ≤ 0.02.
7. Plot the training and validation loss per epoch with proper labels and title.
8. Extract the encoder subnetwork from the trained autoencoder.
9. Obtain the encoded representation of both training and testing samples.
10. Generate a 3D scatter plot of the encoder output and analyze the clusters.
11. Perform Principal Component Analysis (PCA) with 3 components and plot in 3D. Compare the PCA results with the encoder output.
12. Build another autoencoder with only 2 hidden nodes in the middle layer.
13. Repeat steps 5–9 for the 2-node autoencoder, this time generating 2D plots.
14. Compare the validation loss and clustering results with the first autoencoder.

**B. Building a Denoising Convolutional Autoencoder**

1. Download and load the Fashion MNIST dataset. Normalize pixel values to [0,1].
2. Display at least ten sample training images in a single row.
3. Generate noisy versions of training and testing images by adding Gaussian noise.
4. Display at least ten noisy images from the training set.
5. Build a convolutional autoencoder model using Keras based on the given architecture.
6. Compile the model with an appropriate optimizer and loss function.
7. Train the model with a batch size of 100, using early stopping until validation loss ≤ 0.01.
8. Plot the training and validation loss per epoch with labels and title.
9. Obtain the denoised outputs for at least ten noisy test images.
10. Display the noisy inputs (first row) and corresponding denoised outputs (second row).
11. Measure the reconstruction error between original clean images and denoised outputs using a suitable error metric.
12. Compute and report the average reconstruction error across all test images.
13. Collect at least five external clothing images (normalized grayscale, same size as dataset).
14. Apply noise to these external images in the same manner as step 3.
15. Use the trained denoising autoencoder to predict denoised versions.
16. Display the noisy inputs and their denoised outputs in two rows, then record observations.

### Results and Analysis

|  |
| --- |
| # Load the Wine dataset as a Numpy array  # Perform feature normalization using MinMaxScaler  # Perform train-test split with 20% test data  # Save the training data to variable X\_train  # Save the testing data to variable X\_test  # Save the target to variable y  ### --YOUR CODE HERE-- ###  wine = load\_wine()  X = wine.data  y = wine.target  scaler = MinMaxScaler()  X\_normalized = scaler.fit\_transform(X)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_normalized,                                    y, test\_size=0.2, random\_state=42)  y = y  print("X\_train shape:", X\_train.shape)  print("X\_test shape:", X\_test.shape)  print("y shape:", y.shape) |

Figure 1.1. Loading the Wine Dataset and Splitting the Dataset

As seen on Figure 1.1, the wine dataset is imported and the X is set as the data and y is set as the target. Furthermore, normalization is used using the MinMaxScaler which was imported. This was used to normalize the wide X data. Likewise, the normalized data is split to 80-20 where 20% is the test size while the 80% is the train size. Noted that the X train shape is 142 by 13 and the X test shape is 36 by 13 while the y shape is 178.

|  |
| --- |
| # Create the autoencoder model based from the illustrated model plot  # Save the model as autoencoder\_ann variable  ### --YOUR CODE HERE-- ###  input\_dim = X\_train.shape[1]  autoencoder\_ann = Sequential([Input(shape=(input\_dim,)),                                Dense(3, activation="sigmoid"),                                Dense(input\_dim, activation="sigmoid")]) |

Figure 1.2 Creating the Autoencoder

A diagram of a code

AI-generated content may be incorrect.

Figure 1.2.1 Autoencoder Architecture

The coded autoencoder architecture of Figure 1.2.1 is show on Figure 1.2. The encoder’s architecture contains 2 activation sigmoid with an input shape of 13 which is the input dim or the X train other shape.

|  |
| --- |
| # Configure the network for training using the compile method  # Set the optimizer to Adam and determine the appropriate loss function  ### --YOUR CODE HERE-- ###  autoencoder\_ann.compile(optimizer='adam', loss='mse')  # Train the model using X\_train. Use validation with X\_test  # Set the batch size to 10 and the verbosity to 2.  # Ensure the validation loss to be no greater than 0.02  # Use early stopping to determine the appropriate training epochs  # Assign the output to hist\_autoencoder\_ann variable  ### --YOUR CODE HERE-- ###  from keras.callbacks import EarlyStopping  early\_stopping = EarlyStopping(monitor='val\_loss', mode='min', baseline=0.02)  hist\_autoencoder\_ann = autoencoder\_ann.fit(      X\_train, X\_train,      validation\_data=(X\_test, X\_test),      epochs=1000,      batch\_size=10,      verbose=2,      callbacks=[early\_stopping]  )  eval\_base = autoencoder\_ann.evaluate(X\_test, X\_test, verbose=0) |

Figure 1.3. Configuring the Autoencoder and Making of Early Stopper

As seen on Figure 1.3, adam optimizer with an MSE loss is set for the training network. Furthermore, the validation loss is set to be no more than 0.02 in the early stopper and the batch size of 10 with verbose of 2.

|  |
| --- |
| # Extract the losses during training and validation  losses = hist\_autoencoder\_ann.history["loss"]  validation\_losses = hist\_autoencoder\_ann.history["val\_loss"]  epochs = range(1, len(losses) + 1)  # Plot the history of training and validation losses  plt.figure(figsize=(10, 5), constrained\_layout=True)  plt.subplot(1, 2, 1)  ### --YOUR CODE HERE-- ###  plt.plot(epochs, losses)  plt.title('Training Loss per Epoch')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.grid(True)  plt.subplot(1, 2, 2)  plt.plot(epochs, validation\_losses)  plt.title('Validation Loss per Epoch')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.grid(True)  plt.show() |

Figure 1.4. Plotting the Training and Validation loss

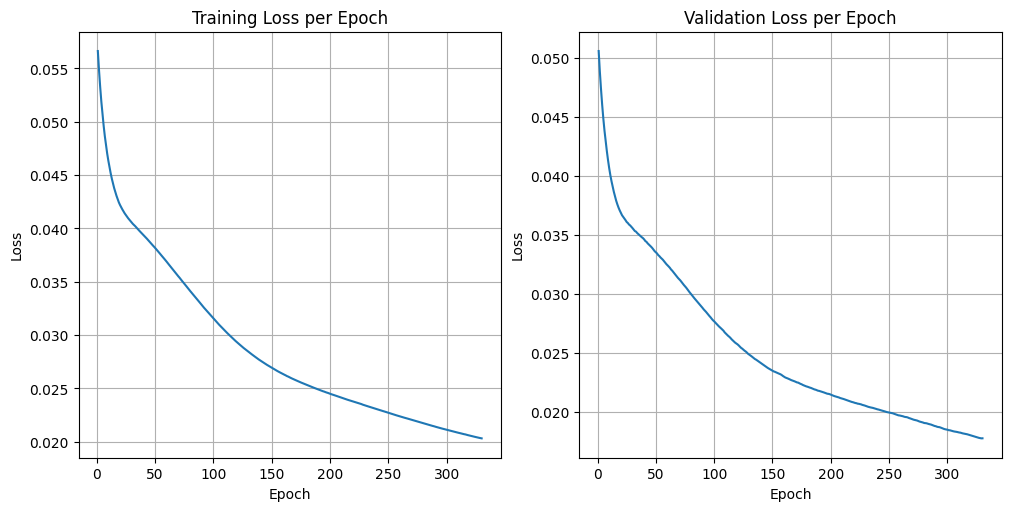


Figure 1.4.1. Training and Validation Loss per Epoch

As seen on Figure 1.4.1, the validation loss is no more than 0.020 above the 250 epoch. Furthermore, the code to plot the training and validation loss is shown in Figure 1.4.

|  |
| --- |
| # Extract the encoder subnetwork  encoder\_ann = Sequential(autoencoder\_ann.layers[:1])  # Obtain the output of the encoder for each dataset sample  # Save the results to X\_encoded variable  # The results should include both training and testing data  ### --YOUR CODE HERE-- ###  X\_encoded = encoder\_ann.predict(X\_normalized) |

Figure 1.5. Getting of Encoder Prediction

Figure 1.5 shows the extraction of the encoder’s output and the getting the normalized encoder’s output.

|  |
| --- |
| # Import PCA from Scikit-Learn  from sklearn.decomposition import PCA  X\_PCA = PCA(n\_components=3).fit\_transform(np.vstack((X\_train, X\_test)))  # Create a 3D plot of the PCA components for each dataset point  # Put appropriate title and axis labels  ### --YOUR CODE HERE-- ###  fig = plt.figure(1, figsize=(8, 6))  ax = fig.add\_subplot(111, projection="3d", elev=45, azim=-45)  ax.scatter(      X\_PCA[:, 0],      X\_PCA[:, 1],      X\_PCA[:, 2],      c=y,      s=20  )  # Set titles and axis labels  ax.set\_title("PCA output")  ax.set\_xlabel("Component 1")  ax.set\_ylabel("Component 2")  ax.set\_zlabel("Component 3")  plt.show() |

Figure 1.6. 3D Plotting Code the PCA

A graph of a function

AI-generated content may be incorrect.

Figure 1.6.1 3D Plot of Given PCA

A diagram of a computer generated graph

AI-generated content may be incorrect.

Figure 1.6.2 3D Plot of Our PCA

Figure 1.6.2 as seen above shows the PCA or the Principal Component Analysis using the X train and X test data. The respective code to plot Figure 1.6.2 is shown in Figure 1.6. For the observations of Figure 1.6.1 and Figure 1.6.2, Figure 1.6.1 is clearly more separated than Figure 1.6.2 for all the targets in the clusters.

|  |
| --- |
| # Define another autoencoder with 2 nodes at hidden layer  ### --YOUR CODE HERE-- ###  autoencoder\_2\_nodes = Sequential([      Input(shape=(13,)),      Dense(2, activation='sigmoid'),      Dense(13, activation='sigmoid')  ])  autoencoder\_2\_nodes.summary()  # Configure and train the network  ### --YOUR CODE HERE-- ###  autoencoder\_2\_nodes.compile(      optimizer='adam',      loss='mse'  )  es = EarlyStopping(monitor='val\_loss', mode='min', baseline=0.02)  history\_2\_nodes = autoencoder\_2\_nodes.fit(      X\_train, X\_train,      epochs=1000,      validation\_data=(X\_test, X\_test),      verbose=2,      batch\_size=10,      callbacks=[es]  )  # Plot the history of training and validation losses  ### --YOUR CODE HERE-- ###  train\_losses = history\_2\_nodes.history["loss"]  val\_losses = history\_2\_nodes.history["val\_loss"]  epochs = range(1, len(train\_losses) + 1)  plt.figure(figsize=(10, 5), constrained\_layout=True)  plt.subplot(1, 2, 1)  plt.plot(epochs, train\_losses)  plt.xlabel("Epochs")  plt.ylabel("Training Loss")  plt.title("Training Loss per Epoch")  plt.subplot(1, 2, 2)  plt.plot(epochs, val\_losses)  plt.xlabel("Epochs")  plt.ylabel("Validation Loss")  plt.title("Validation Loss per Epoch")  plt.show()  # Obtain the output of the encoder for each dataset sample  ### --YOUR CODE HERE-- ###  encoder\_2\_nodes = Sequential(autoencoder\_2\_nodes.layers[:1])  X\_encoded\_2\_nodes = encoder\_2\_nodes.predict(X\_normalized)  plt.figure(figsize=(8, 6))  plt.scatter(      X\_encoded\_2\_nodes[:, 0],      X\_encoded\_2\_nodes[:, 1],      c=y,      s=20  )  plt.title("Encoder Output")  plt.xlabel("Component 1")  plt.ylabel("Component 2")  plt.show() |

Figure 1.7 Making Another Autoencoder but with 2D Plots

A graph of loss and loss

AI-generated content may be incorrect.

Figure 1.7.1 Training and Validation Loss per Epoch

A graph of colored dots

AI-generated content may be incorrect.

Figure 1.7.2 2D Plots of the Encoder

As seen on Figure 1.7.1 and Figure 1.7.2, the training and validation loss plot per epoch shown and the 2D encoder output. Likewise, the code for these two figures is shown in Figure 1.7. The code inside Figure 1.7 contains the defining of autoencoder together with the training and configuration of the autoencoder together with the early stopping and its parameters for training it and plotting both the training and validation loss together with the 2D Encoder plot.

|  |
| --- |
| # Import functions and classes  from keras.datasets import fashion\_mnist  from keras.utils import to\_categorical  # Load the Fashion MNIST dataset  # Rescale the pixel values to be between 0 and 1  # Save the training images and labels to x\_train and y\_train variables  # Save the testing images and labels to x\_test and y\_test variables  # Plot ten (10) images from training set in a single row  ### --YOUR CODE HERE-- ###  (x\_train, y\_train), (x\_test, y\_test) = fashion\_mnist.load\_data()  x\_train = x\_train.astype('float32') / 255.0  x\_test = x\_test.astype('float32') / 255.0  class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',                 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']  fig, axes = plt.subplots(1, 10, figsize=(15, 2),                           subplot\_kw={'xticks':[], 'yticks':[]})  for i, ax in enumerate(axes.flat):      ax.imshow(x\_train[i], cmap='binary')      ax.set\_title(f'{class\_names[y\_train[i]]}', fontsize=10)  plt.suptitle('Original Fashion MNIST Images')  plt.tight\_layout()  plt.show() |

Figure 1.8. Loading and Splitting the Fashion MNIST dataset

A black and white picture of a chair

AI-generated content may be incorrect.

Figure 1.8.1 Loading and Splitting the Fashion MNIST dataset

The code shown on Figure 1.8 shows that loading of the Fashion MNIST dataset while assigning the training and testing images and labels together with the plotting of the 10 images from the dataset. The sample images of the Fashion MNIST dataset which was plotted can be seen on the Figure 1.8.1.

|  |
| --- |
| # Apply noise to the image  x\_train\_noisy = x\_train + 0.2\*np.random.normal(loc=0, scale=1,                                                 size=x\_train.shape)  x\_test\_noisy = x\_test + 0.2\*np.random.normal(loc=0, scale=1,                                               size=x\_test.shape)  x\_train\_noisy = np.clip(x\_train\_noisy, 0, 1)  x\_test\_noisy = np.clip(x\_test\_noisy, 0, 1)  fig, axes = plt.subplots(1, 10, figsize=(10, 10),                           subplot\_kw={'xticks':[], 'yticks':[]},                           gridspec\_kw=dict(hspace=0.1, wspace=0.1))  for i, ax in enumerate(axes.flat):    ax.imshow(x\_train\_noisy[i], cmap='binary', interpolation='none') |

Figure 1.9. Adding Noise to the Images and Plotting it

Figure 1.9 shows the code for adding noise to the training and testing image dataset which was also plotted as seen on the Figure 1.9.1.

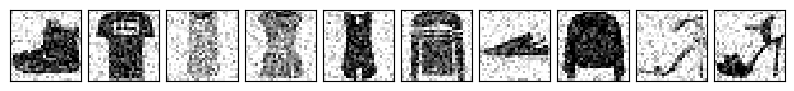


Figure 1.9.1 Noise Image Dataset Plot

|  |
| --- |
| # Import functions and classes  from keras.layers import Conv2D, MaxPooling2D, UpSampling2D  # Create the autoencoder model based from the illustrated model plot  # Save the model as autoencoder\_cnn variable  ### --YOUR CODE HERE-- ###  autoencoder\_cnn = Sequential([      Input(shape=(28, 28, 1)),      Conv2D(32, (3, 3), activation='relu', padding='same', name='conv2d\_1'),      MaxPooling2D((2, 2), padding='same', name='max\_pooling2d'),      Conv2D(32, (3, 3), activation='relu', padding='same', name='conv2d\_2'),      MaxPooling2D((2, 2), padding='same', name='code'),      Conv2D(32, (3, 3), activation='relu', padding='same', name='conv2d\_3'),      UpSampling2D((2, 2), name='up\_sampling2d'),      Conv2D(32, (3, 3), activation='relu', padding='same', name='conv2d\_4'),      UpSampling2D((2, 2), name='up\_sampling2d\_1'),      Conv2D(1, (3, 3), activation='sigmoid', padding='same', name='decode')  ])  autoencoder\_cnn.summary() |

Figure 1.10 Making a Denoising Autoencoder

A black and white diagram

AI-generated content may be incorrect.

Figure 1.10.1 Denoising Autoencoder Architecture

The coding of the denoising autoencoder architecture which is shown on Figure 1.10.1 is seen in Figure 1.10. The architecture of the denoising autoencoder contains four 2D convolution with activation function of relu, two up 2D sampling, and two 2D max pooling.

|  |
| --- |
| # Configure the network for training  ### --YOUR CODE HERE-- ###  autoencoder\_cnn.compile(optimizer='adam', loss='mse')  # Train the model. Perform validation as well  # Set the batch size to 100 and the verbosity to 2  # Ensure the validation loss to be no greater than 0.01  # Use early stopping to determine the appropriate training epochs  # Assign the output to hist\_autoencoder\_cnn variable  ### --YOUR CODE HERE-- ###  early\_stopping = EarlyStopping(      monitor='val\_loss',      baseline = 0.01,      mode = 'min'  )  hist\_autoencoder\_cnn = autoencoder\_cnn.fit(      x\_train\_noisy, x\_train\_noisy,      validation\_data=(x\_test\_noisy, x\_test\_noisy),      epochs=900,      batch\_size=100,      verbose=2,      callbacks=[early\_stopping]  ) |

Figure 1.11 Configuring the Autoencoder Parameters with Early Stopper

Likewise, Figure 1.11 shows the configuration of the denoising autoencoder with an adam optimizer and MSE loss. For the training, an early stopper is used with a baseline of 0.01 for the validation loss and the batch size is 100 and verbosity of 2.

|  |
| --- |
| # Extract the losses during training and validation  losses = hist\_autoencoder\_cnn.history["loss"]  validation\_losses = hist\_autoencoder\_cnn.history["val\_loss"]  epochs = range(1, len(losses) + 1)  # Plot the history of training and validation losses  plt.figure(figsize=(10, 5), constrained\_layout=True)  plt.subplot(1, 2, 1)  ### --YOUR CODE HERE-- ###  plt.plot(epochs, losses)  plt.xlabel("Epochs")  plt.ylabel("Training Loss")  plt.title("Training Loss per Epoch")  plt.subplot(1, 2, 2)  plt.plot(epochs, validation\_losses)  plt.xlabel("Epochs")  plt.ylabel("Validation Loss")  plt.title("Validation Loss per Epoch")  plt.show() |

Figure 1.12 Plotting the Training and Validation Loss of Autoencoder

A graph of a loss

AI-generated content may be incorrect.

Figure 1.12.1 Training and Validation Loss of Autoencoder

For the plotting of the training and validation loss of the denoising autoencoder, the code is shown on Figure 1.12 and the plot curve is shown on Figure 1.12.1 where the validation loss is not able to reach below 0.01. This might be due to using adam optimizer for training the model.

|  |
| --- |
| # Obtain model output for ten (10) input noisy images  # Display both the input noisy images and the output denoised image in two rows  # The first row contains the input noisy images  # The second row contains the output denoised images  ### --YOUR CODE HERE-- ###  denoised\_images = autoencoder\_cnn.predict(x\_test\_noisy[:10])  fig, axes = plt.subplots(2, 10, figsize=(15, 4))  for i in range(10):    axes[0,i].imshow((x\_test\_noisy[i]).reshape(28,28), cmap='gray')    axes[0,i].set\_title("Noisy Img")    axes[1,i].imshow((denoised\_images[i]).reshape(28,28), cmap='gray')    axes[1,i].set\_title("Denoised Img")  plt.suptitle("Image Denoising")  plt.show() |

Figure 1.13. Displaying the Noisy and Denoised Images

A collage of images of a person's body

AI-generated content may be incorrect.

Figure 1.13.1. Image Denoising Dataset

Figure 1.13 shows the code for displaying the denoise and noisy images in the dataset. Furthermore, the plot with the images can be seen on Figure 1.13.1 where the denoising is able to remove higher noticeable noise while keeping the clothes distinguishable. However, the denoised image still has some noticeable noise at the background.

|  |
| --- |
| ### --YOUR CODE HERE-- ###  from sklearn.metrics import mean\_absolute\_error  decoded\_images = autoencoder\_cnn.predict(x\_test\_noisy)  errors = []  for i in range(len(x\_test)):      mae = mean\_absolute\_error(          x\_test[i].reshape(-1),          decoded\_images[i].reshape(-1)      )      errors.append(mae)  errors = np.array(errors)  avg\_error = np.mean(errors)  print("Average MAE:", avg\_error)  plt.hist(errors, bins=50)  plt.title("Distribution of MAE")  plt.xlabel("MAE")  plt.ylabel("Number of Images")  plt.show() |

Figure 1.14. Displaying the Average MAE and its Image Distribution

A graph of a distribution of a number

AI-generated content may be incorrect.

Figure 1.14.1 MAE vs Number of Images Distribution

The code for doing the MAE distribution and the plot of the images to MAE distribution is shown at Figure 1.14 and Figure 1.14.1. Likewise, the average MAE is shown to be 0.09466809390932321.

|  |
| --- |
| ### --YOUR CODE HERE-- ###  import requests  from PIL import Image  from io import BytesIO  clothing\_urls = [    "https://storage.googleapis.com/kagglesdsdata/datasets/929774/1572891/images\_original/00805d0e-7fe5-4251-b577-86065e4f6587.jpg?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=databundle-worker-v2%40kaggle-161607.iam.gserviceaccount.com%2F20250914%2Fauto%2Fstorage%2Fgoog4\_request&X-Goog-Date=20250914T111709Z&X-Goog-Expires=345600&X-Goog-SignedHeaders=host&X-Goog-Signature=",    "https://storage.googleapis.com/kagglesdsdata/datasets/139630/329006/fashion-dataset/images/10020.jpg?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=databundle-worker-v2%40kaggle-161607.iam.gserviceaccount.com%2F20250914%2Fauto%2Fstorage%2Fgoog4\_request&X-Goog-Date=20250914T102554Z&X-Goog-Expires=345600&X-Goog-SignedHeaders=host&X-Goog-Signature=",    "https://storage.googleapis.com/kagglesdsdata/datasets/139630/329006/fashion-dataset/images/10029.jpg?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=databundle-worker-v2%40kaggle-161607.iam.gserviceaccount.com%2F20250914%2Fauto%2Fstorage%2Fgoog4\_request&X-Goog-Date=20250914T102554Z&X-Goog-Expires=345600&X-Goog-SignedHeaders=host&X-Goog-Signature=",    "https://storage.googleapis.com/kagglesdsdata/datasets/139630/329006/fashion-dataset/images/10013.jpg?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=databundle-worker-v2%40kaggle-161607.iam.gserviceaccount.com%2F20250914%2Fauto%2Fstorage%2Fgoog4\_request&X-Goog-Date=20250914T102554Z&X-Goog-Expires=345600&X-Goog-SignedHeaders=host&X-Goog-Signature=",    "https://storage.googleapis.com/kagglesdsdata/datasets/929774/1572891/images\_original/002eb5b8-6541-42a3-9596-0d94f7b866ae.jpg?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=databundle-worker-v2%40kaggle-161607.iam.gserviceaccount.com%2F20250914%2Fauto%2Fstorage%2Fgoog4\_request&X-Goog-Date=20250914T111709Z&X-Goog-Expires=345600&X-Goog-SignedHeaders=host&X-Goog-Signature=",  ]  x\_external = []  for url in clothing\_urls:    response = requests.get(url, timeout=10)    img = Image.open(BytesIO(response.content))    img = img.convert("L")    img = img.resize((28, 28))    img\_array = np.array(img) / 255.0    x\_external.append(img\_array.reshape(28, 28))  x\_external = np.array(x\_external)  x\_external\_noisy = x\_external + 0.2 \* np.random.normal(      loc=0, scale=1, size=x\_external.shape  )  x\_external\_noisy = np.clip(x\_external\_noisy, 0, 1)  decoded\_external = autoencoder\_cnn.predict(x\_external\_noisy)  fig, axes = plt.subplots(2, 5, figsize=(12, 5))  for i in range(5):      axes[0, i].imshow(x\_external\_noisy[i].reshape(28, 28), cmap="binary")      axes[0, i].set\_title("Noisy")      axes[1, i].imshow(decoded\_external[i].reshape(28, 28), cmap="binary")      axes[1, i].set\_title("Denoised")  plt.suptitle("Noisy and Denoised Images")  plt.show() |

Figure 1.15. Displaying the 5 Tested Noisy to Denoised Images

A group of images of a person

AI-generated content may be incorrect.

Figure 1.15.1 Five Noisy to Denoised Images

For the last activity, five noisy to denoised images were tested where the images are taken from the internet and not from the same dataset. The code of this can be seen on Figure 1.15 and the plot with the images can be seen on Figure 1.15.1. For the observations, the first image was not able to denoise because the image has a light-colored background which shows the weakness to the grayscale to the image. Furthermore, the second and third image was successfully able to denoise the image however it became blurry with some noise in the background. Lastly, the fourth and fifth image were not successful in deblurring while making the clothes distinguishable.

### Answer to Guide Questions, Insights, and Reflections

A.1. Is the final validation loss better than the previous model?

The data shown on Figure 1.7.1 shows that the final validation loss is 0.0165 while the other previous model in Figure 1.4 has a final validation loss of 0.0177. This implies that the second model is better than the initially trained autoencoder model.

A.2. Can the model clearly separate the targets into distinguishable clusters?

The data on Figure 1.6.2 and Figure 1.6.1 shows that Figure 1.6.2 has more distinguishable clusters in comparison to Figure 1.6.1.

For my insight and reflection, I learned a lot about autoencoders and the different parts which are the original data, encoder, code, decoder and reconstructed data. This means that autoencoders are becoming less like a black box concept to me because parts such as the hidden layer are being explained how to be fine tuned for the input and output layer.

## Activity 2: Variational Autoencoders and Generative Adversarial Networks

### Objectives

* Understand the working principle and architecture of variational autoencoders and generative adversarial networks
* Build and train a variational autoencoder using deep learning framework
* Build and train a generative adversarial network using deep learning framework
* Visualize the output of the generative networks
* Assess the performance of the implemented generative networks

### Experimental Procedure

<…>

### Results and Analysis

|  |
| --- |
| # Import libraries  from torchvision.datasets import EMNIST  # Get the EMNIST dataset. Use the balanced split  # Normalize the image pixel values to within the range [0, 1]  # Save the training data to variable X\_train as a Numpy array  # Save the testing data to variable X\_test as a Numpy array  # Save the targets to variable y as a Numpy array  # Note that the images provided are inverted horizontally and  # rotated 90 anti-clockwise. It must be oriented properly  ### --YOUR CODE HERE-- ###  from torchvision import transforms  emnist\_train = EMNIST("./data", split="balanced", train=True, download=True,  transform=transforms.ToTensor()  )  emnist\_test = EMNIST("./data", split="balanced", train=False, download=True,  transform=transforms.ToTensor()  )  X\_train = emnist\_train.data.numpy().astype("float32")/255  X\_train = np.rot90(X\_train, k=2)  X\_train = np.flipud(X\_train)  #X\_train = np.fliplr(X\_train)  X\_train = np.expand\_dims(X\_train, axis=-1)  X\_test = emnist\_test.data.numpy().astype("float32")/255  X\_test = np.rot90(X\_test, k=2)  X\_test = np.flipud(X\_test)  #X\_test = np.fliplr(X\_test)  X\_test = np.expand\_dims(X\_test, axis=-1)  y = emnist\_train.targets.numpy()  # Display the first thirty (30) images from the training split  # Place ten (10) images per row  # Each image should have a label underneath  ### --YOUR CODE HERE-- ###  figure, axs = plt.subplots(3, 10, figsize=(20, 6))  print(X\_train.shape)  for i, j in enumerate(axs.flatten()):  j.imshow(X\_train[i])  j.set\_title(y[i])  plt.tight\_layout()  plt.show() |

Figure 2.1. Code for Loading the EMNIST Dataset and Plotting it

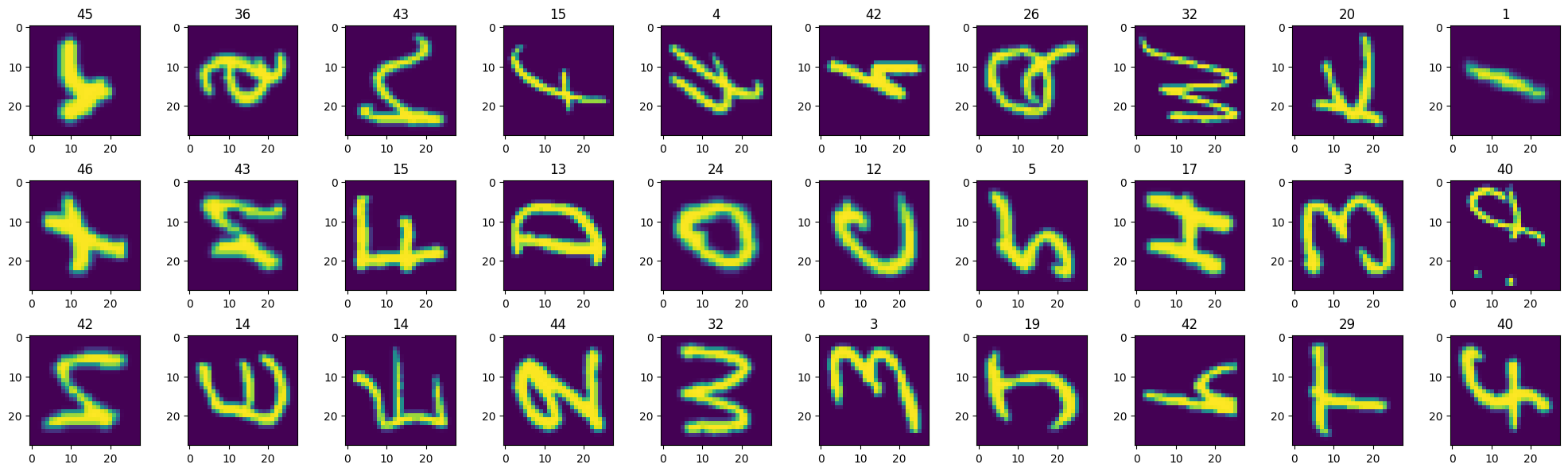


Figure 2.1.1 EMNIST Dataset Plot

Figure 2.1. and Figure 2.1.1 show

|  |
| --- |
| # Create the encoder network layers from the illustrated model plot  # Use the latent\_dim variable to define the probability distribution parameters  # Save the input layer to input\_img variable  # Save the mean layer output to z\_mean variable  # Save the logarithm of the variance layer output to z\_log\_var variable  # Save the sampling layer output to z\_sample variable  ### --YOUR CODE HERE-- ###  input\_img = Input(shape=(28,28,1))  x = Conv2D(32, 1, activation="relu")(input\_img)  print(x.shape)  m = MaxPooling2D(2)(x)  print(m.shape)  x1 = Conv2D(64, 1, activation="relu")(m)  print(x1.shape)  m1 = MaxPooling2D(2, padding="same")(x1)  print(m1.shape)  xm = Flatten()(m1)  print(xm.shape)  d = Dense(64, activation="relu")(xm)  print(d.shape)  z\_mean = Dense(latent\_dim, activation="linear")(d)  print(z\_mean.shape)  z\_log\_var = Dense(latent\_dim, activation="linear")(d)  print(z\_log\_var.shape)  z\_sample = Sampling()([z\_mean, z\_log\_var])  print(z\_sample.shape)  # Create the encoder network using the layers defined above  vae\_encoder = Model(input\_img, [z\_mean, z\_log\_var, z\_sample], name="encoder") |

Figure 2.2. Code for Building Encoder Network Layers

A black and white diagram

AI-generated content may be incorrect.

Figure 2.2.1 Decoder Subnetwork Architecture

Figure 2.2. and Figure 2.2.1 show

|  |
| --- |
| # Import functions and classes from Keras library  from keras.layers import Reshape, UpSampling2D  # Create the decoder network layers from the illustrated model plot  # Use the latent\_dim variable to define the probability distribution parameters  # Save the input layer to input\_latent variable  # Save the last convolutional layer output to decoded variable  ### --YOUR CODE HERE-- ###  input\_latent = Input(shape=(8,))  print(input\_latent.shape)  d = Dense(3136, activation="relu")(input\_latent)  print(d.shape)  r = Reshape((7,7,64))(d)  print(r.shape)  x = Conv2D(64, 1, activation="relu")(r)  print(x.shape)  u = UpSampling2D(2)(x)  print(u.shape)  x1 = Conv2D(32, 1, activation="relu")(u)  print(x1.shape)  u1 = UpSampling2D(2)(x1)  print(u1.shape)  decoded = Conv2D(1, 1, activation="sigmoid")(u1)  print(decoded.shape)  # Create the decoder network using the layers defined above  vae\_decoder = Model(input\_latent, decoded, name="decoder") |

Figure 2.3. Code for Building the Decoder Subnetwork

A black and white diagram

AI-generated content may be incorrect.

Figure 2.3.1 Decoder Subnetwork Architecture

As seen on Figure 2.3 and Figure 2.3.1

|  |
| --- |
| # Configure the network for training using the compile method  # Set the optimizer to your choice and determine the appropriate loss function  ### --YOUR CODE HERE-- ###  from keras.optimizers import Adam  from keras.callbacks import EarlyStopping  vae\_cnn.compile(optimizer="adam", loss='mse')  # Train the model. Set the batch size to 100  # Use early stopping to determine the appropriate training epochs  # Perform model hyperparameter tuning as needed  # Assign the output to hist\_vae\_cnn variable  ### --YOUR CODE HERE-- ###  callback = EarlyStopping(patience=5, monitor="total\_loss", mode='min')  hist\_vae\_cnn = vae\_cnn.fit(X\_train, batch\_size=100, epochs=100, callbacks=[callback], validation\_split=0.2) |

Figure 2.4. Code for Configuring the training VAE with Early Stopping

As seen on Figure 2.4

|  |
| --- |
| # Plot the history of training and validation losses  # Put the training losses subplots in the first row  # Put the validation losses subplots in the second row  ### --YOUR CODE HERE-- ###  figure, axs = plt.subplots(2, 3, figsize=(15,6))  train\_losses = [reconstruction\_losses, kl\_losses, losses]  validation\_losses = [validation\_reconstruction\_losses, validation\_kl\_losses, validation\_losses]  titles = ["Reconstruction Loss", "KL Divergence Loss", "Total Loss"]  colors = ["red", "blue", "green"]  for i in range(3):    axs[0, i].plot(epochs, train\_losses[i], label="Train", color=colors[i])    axs[0, i].set\_title(f"Training {titles[i]}")    axs[0, i].set\_xlabel("Epoch")    axs[0, i].set\_ylabel("Loss")    axs[0, i].legend()  for i in range(3):    axs[1, i].plot(epochs, train\_losses[i], label="Validation", color=colors[i])    axs[1, i].set\_title(f"Validation {titles[i]}")    axs[1, i].set\_xlabel("Epoch")    axs[1, i].set\_ylabel("Loss")    axs[1, i].legend()  plt.tight\_layout()  plt.show() |

Figure 2.5. Code for Plotting the Six Plots

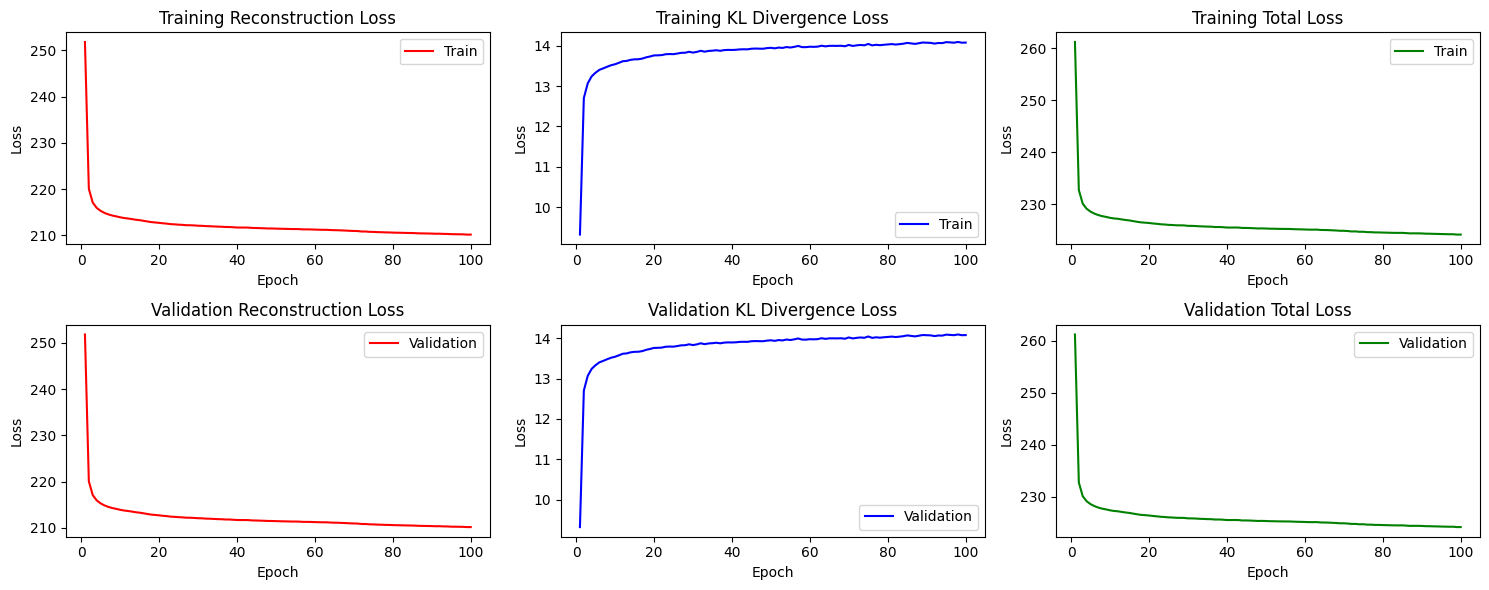


Figure 2.5.1 Training and Validation Reconstruction, Divergence, and Total Loss

As seen on Figure 2.5 and Figure 2.5.1

|  |
| --- |
| # Select at least 15 test images  # Feed the test images to the encoder network. Save it to z\_img variable  # Feed the encoder outputs to the decode network  ### --YOUR CODE HERE-- ###  num\_images = 15  test\_images = X\_test[:num\_images]  z\_mean, z\_log\_var, z\_img = vae\_cnn.encoder.predict(test\_images)  decoded\_images = vae\_cnn.decoder.predict(z\_img)  # Display the selected test images in the first row  # Display the resulting images in the second row  ### --YOUR CODE HERE-- ###  figure, axs = plt.subplots(2, num\_images, figsize=(20, 6))  for i in range(num\_images):    axs[0, i].imshow(test\_images[i])    axs[1, i].imshow(decoded\_images[i])  plt.tight\_layout()  plt.show() |

Figure 2.6 Code for Plotting the VAE Tested Images

A group of symbols in squares

AI-generated content may be incorrect.

Figure 2.6.1 Plot of the 15 VAE Tested Images

As seen on Figure 2.6 and Figure 2.6.1

|  |
| --- |
| # Select at least 1000 test images  # Feed the test images to the encoder network. Save it to z\_lat variable  ### --YOUR CODE HERE-- ###  num\_images = 1000  test\_images = X\_test[:num\_images]  z\_mean, z\_log\_var, z\_lat = vae\_cnn.encoder.predict(test\_images)  # Create a histogram for each latent space variable into subplots  # The subplots should be arrange into a grid with 2 rows and 4 columns  ### --YOUR CODE HERE-- ###  figure, axs = plt.subplots(2, 4, figsize=(15,6))  for i in range(8):    axs[i//4, i%4].hist(z\_lat[:i])    axs[i//4, i%4].set\_title(f"Latent Variable {i+1}")    axs[i//4, i%4].set\_xlabel("Value")    axs[i//4, i%4].set\_ylabel("Frequency")  plt.tight\_layout()  plt.show() |

Figure 2.7. Code for Histogram of all Eight Latent Variables

A group of graphs with different colored lines

AI-generated content may be incorrect.

Figure 2.7.1 Histogram Plot for the Eight Latent Variables

As seen on Figure 2.7 and Figure 2.7.1

|  |
| --- |
| # Select the test image and display them in a single row  ### --YOUR CODE HERE-- ###  lower1 = X\_test[0]  upper1 = X\_test[13]  upper2 = X\_test[1]  figure, axs = plt.subplots(1, 3, figsize=(4,4))  axs[0].imshow(lower1)  axs[1].imshow(upper1)  axs[2].imshow(upper2)  plt.tight\_layout()  plt.show()  # Obtain the output of the encoder for each test image  # Save the encoder output for the first image to z1 variable  # Save the encoder output for the second image to z2 variable  # Save the encoder output for the third image to z3 variable  # Perform the difference z1 - z2 and store to zd variable  # Perform the sum z3 + zd and store to z3new variable  ### --YOUR CODE HERE-- ###  z1, z\_log\_var, z\_img1 = vae\_cnn.encoder.predict(np.expand\_dims(lower1, axis=0))  z2, z\_log\_var, z\_img2 = vae\_cnn.encoder.predict(np.expand\_dims(upper1, axis=0))  z3, z\_log\_var, z\_img3 = vae\_cnn.encoder.predict(np.expand\_dims(upper2, axis=0))  zd = z1 - z2  z3new = z3 + zd  # Feed the new latent vector z3new to the decoder network  # Display the resulting image  ### --YOUR CODE HERE-- ###  decoded\_image = vae\_cnn.decoder.predict(z3new)  plt.figure(figsize=(4,4))  plt.imshow(decoded\_image.squeeze())  plt.tight\_layout()  plt.show() |

Figure 3.8. Instructions to get Difference Between Lower to Uppercase Letter with Plot

A yellow and green logo

AI-generated content may be incorrect.

Figure 3.8.1 Plot for Z1, Z2, and Z3

A colorful squares with numbers

AI-generated content may be incorrect.

Figure 3.8.2 Plot for Z3new

As seen on Figure 3.8, Figure 3.8.1, and Figure 3.8.2,

|  |
| --- |
| # Import libraries  from torchvision import transforms  from torchvision.datasets import EMNIST  # added these 3 imports  import tensorflow as tf  import numpy as np  import matplotlib.pyplot as plt  # Load the EMNIST dataset  emnist\_train = EMNIST("./data", split="balanced", train=True, download=True,                        transform=transforms.ToTensor()                        )  emnist\_test = EMNIST("./data", split="balanced", train=False, download=True,                       transform=transforms.ToTensor()                       )  # Define the generator function for Tensorflow dataset  def emnist\_ds\_generator(dataset):    for image, \_ in dataset:      yield np.transpose(image.numpy(), axes=(0, 2, 1)).reshape(28, 28, 1)  # Define the Tensorflow dataset output signature  emnist\_ds\_osig = tf.TensorSpec(shape=(28, 28, 1), dtype=tf.float32)  # Create the Tensorflow dataset  emnist\_ds\_train = tf.data.Dataset.from\_generator(      lambda: emnist\_ds\_generator(emnist\_train),      output\_signature=emnist\_ds\_osig  )  emnist\_ds\_test = tf.data.Dataset.from\_generator(      lambda: emnist\_ds\_generator(emnist\_test),      output\_signature=emnist\_ds\_osig  )  emnist\_ds = emnist\_ds\_train.concatenate(emnist\_ds\_test)  emnist\_ds = emnist\_ds\_train.concatenate(emnist\_ds\_test)  emnist\_ds = emnist\_ds.shuffle(buffer\_size=1000)  # Display 30 images from the Tensorflow dataset in three rows (10 images per row)  ### --YOUR CODE HERE-- ###  sample\_images = list(emnist\_ds.take(30))  fig, axes = plt.subplots(3, 10, figsize=(15, 6))  for i, image in enumerate(sample\_images):      row = i // 10  # get which row (0, 1, or 2)      col = i % 10   # get which column (0-9)      img\_array = image.numpy().squeeze()      axes[row, col].imshow(img\_array, cmap='gray')      axes[row, col].axis('off')  plt.tight\_layout()  plt.suptitle('30 Random Images from EMNIST Dataset')  plt.show() |

Figure 3.9. Code for Loading EMNIST Image Dataset

A black and white photo of letters

AI-generated content may be incorrect.

Figure 3.9.1 Plot for Displaying 30 EMNIST Images

As seen in Figure 3.9 and Figure 3.9.1

|  |
| --- |
| # Import functions and classes from Keras library  from keras import Input, Sequential  from keras.layers import LeakyReLU, Dense, Dropout  from keras.layers import Conv2D, Flatten  # Create the discriminator. Save it to gan\_discriminator variable  # Convolutions should have a filter size of 5 and a stride of 2  # Dropout rate is set to 0.3  ### --YOUR CODE HERE-- ###  gan\_discriminator = Sequential([      Input(shape=(28, 28, 1)),      Conv2D(64, kernel\_size=5, strides=2, padding='same'),      LeakyReLU(alpha=0.2),      Dropout(0.3),      Conv2D(128, kernel\_size=5, strides=2, padding='same'),      LeakyReLU(alpha=0.2),      Dropout(0.3),      Flatten(),      Dense(1)  ])  gan\_discriminator.summary() |

Figure 3.10 Code for Making the Discriminator

A black and white diagram

AI-generated content may be incorrect.

Figure 3.10.1 Discriminator Subnetwork Architecture

As seen on Figure 3.10 and Figure 3.10.1,

|  |
| --- |
| # Import functions and classes from Keras library  from keras import Input, Sequential  from keras.layers import LeakyReLU, Dense  from keras.layers import Reshape, Conv2DTranspose, BatchNormalization  # Specify the latent space dimension  latent\_dim = 128  # Create the generator. Save it to gan\_generator variable  # Transpose convolutions should have a filter size of 5  # The first transpose convolution has stride of 1. Others have stride of 2  ### --YOUR CODE HERE-- ###  gan\_generator = Sequential([      Input(shape=(latent\_dim,)),      # Dense layer: 128 -> 12544 (7\*7\*256 = 12544)      Dense(12544),      BatchNormalization(),      LeakyReLU(alpha=0.2),      Reshape((7, 7, 256)),      Conv2DTranspose(128, kernel\_size=5, strides=1, padding='same'),      BatchNormalization(),      LeakyReLU(alpha=0.2),      Conv2DTranspose(64, kernel\_size=5, strides=2, padding='same'),      BatchNormalization(),      LeakyReLU(alpha=0.2),      Conv2DTranspose(1, kernel\_size=5, strides=2, padding='same', activation='relu')  ]) |

Figure 3.11. Code making the Generator

A black and white diagram

AI-generated content may be incorrect.

Figure 3.11.1 Generator Subnetwork

As seen on Figure 3.11 and Figure 3.11.1

|  |
| --- |
| # Configure the network for training  from keras.optimizers import Adam  from keras.losses import BinaryCrossentropy  gan.compile(      d\_optimizer=Adam(learning\_rate=0.0002),      g\_optimizer=Adam(learning\_rate=0.0002),      loss\_fn=BinaryCrossentropy(from\_logits=True),  )  # Train the model. Set the dataset batch size to 100  # You may take 100 batches only to speed up the training  # Assign the output to gan\_hist variable  ### --YOUR CODE HERE-- ###  batch\_size = 100  batch\_count = 100  train\_dataset = emnist\_ds.batch(batch\_size).take(batch\_count)  gan\_hist = gan.fit(      train\_dataset,      epochs = 100,      verbose = 1,  )  print(f"Final Generator Loss: {gan\_hist.history['g\_loss'][-1]:.4f}")  print(f"Final Discriminator Loss: {gan\_hist.history['d\_loss'][-1]:.4f}") |

Figure 3.12. Code for Training the EMNIST using the GAN Generator and Discriminator

As seen on Figure 3.12

|  |
| --- |
| # Extract the losses during training  dis\_losses = gan\_hist.history["d\_loss"]  gen\_losses = gan\_hist.history["g\_loss"]  epochs = range(1, len(dis\_losses) + 1)  # Plot the history of training and validation losses  ### --YOUR CODE HERE-- ###  import importlib  import matplotlib.pyplot  importlib.reload(matplotlib.pyplot)  import matplotlib.pyplot as plt  plt.figure(figsize=(10,6))  plt.plot(epochs, gen\_losses, 'b-', label='Generator Loss', linewidth=2, marker='o')  plt.plot(epochs, dis\_losses, 'r-', label='Discriminator Loss', linewidth=2, marker='s')  plt.title('Custom GAN: Generator and Discriminator Losses per Epoch')  plt.xlabel('Epoch')  plt.ylabel('Loss')  plt.legend()  plt.grid(True, alpha=0.3)  plt.tight\_layout()  plt.show() |

Figure 3.13. Code for Plotting the Generator and Discriminator Loss Plot

As seen on Figure 3.13

|  |
| --- |
| # Import functions and classes from Keras library  from keras.random import normal  import numpy as np  import matplotlib.pyplot as plt  importlib.reload(matplotlib.pyplot)  # Generate 30 random 128-dimensional vectors as input  # Feed the random inputs to the generator network  # Display the output images of the generator  ### --YOUR CODE HERE-- ###  num\_samples = 30  random\_inputs = normal(shape=(num\_samples, latent\_dim), seed=42)  generated\_images = gan\_generator.predict(random\_inputs, verbose=0)  plt.figure(figsize=(15, 10))  for i in range(num\_samples):      plt.subplot(5, 6, i+1)      img = generated\_images[i].squeeze()      plt.imshow(img, cmap='gray')      plt.axis('off')      plt.title(f'#{i+1}', fontsize=8)  plt.suptitle('Random Inputs', fontsize=14)  plt.tight\_layout()  plt.show()  # Add a small random offsets to the initial inputs  # Feed the new input to the generator network  # Display the new output images of the generator  ### --YOUR CODE HERE-- ###  offset\_magnitude = 0.1  random\_offsets = normal(shape=(num\_samples, latent\_dim), seed=123) \* offset\_magnitude  new\_inputs = random\_inputs + random\_offsets  new\_generated\_images = gan\_generator.predict(new\_inputs, verbose=0)  plt.figure(figsize=(15, 10))  for i in range(num\_samples):      plt.subplot(5, 6, i+1)      img = new\_generated\_images[i].squeeze()      plt.imshow(img, cmap='gray')      plt.axis('off')      plt.title(f'#{i+1}', fontsize=8)  plt.suptitle('Generated output images', fontsize=14)  plt.tight\_layout()  plt.show() |

Figure 3.14. Code for Feeding Generator with 30 Random Inputs

A collage of images of a person's body

AI-generated content may be incorrect.

Figure 3.14.1 Plot for Displaying the Random Inputs Images

A collage of images of a person's body

AI-generated content may be incorrect.

Figure 3.14.2 Plot for Displaying the Generated Output Images

As seen on Figure 3.14, Figure 3.14.1, and Figure 3.14.2

### Answer to Guide Questions, Insights, and Reflections

<…>

## Activity 3: <Title>

### Objectives

<…>

### Experimental Procedure

<…>

### Results and Analysis

<…>

### Answer to Guide Questions, Insights, and Reflections

<…>

## <Add more activity sections as needed>

|  |
| --- |
| ### --YOUR CODE HERE-- ###  import |

Figure 1.15. Displaying the 5 Tested Noisy to Denoised Images

Figures and Tables

A graph with a line

Description automatically generated

Figure x. Training loss for the autoencoder…

Explain and provide analysis of each plot or model illustration…

Table x. Model accuracy comparison…

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Explain and provide analysis of each table provided…

>

## Activity 4: <Title>

### Objectives

<…>

### Experimental Procedure

<…>

### Results and Analysis

<…>

### Answer to Guide Questions, Insights, and Reflections

<…>

## <Add more activity sections as needed>

|  |
| --- |
| ### --YOUR CODE HERE-- ###  import your mom |

Figure 1.15. Displaying the 5 Tested Noisy to Denoised Images

Figures and Tables

A graph with a line

Description automatically generated

Figure x. Training loss for the autoencoder…

Explain and provide analysis of each plot or model illustration…

Table x. Model accuracy comparison…

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Explain and provide analysis of each table provided…

>

## Activity 5: <Title>

### Objectives

<…>

### Experimental Procedure

<…>

### Results and Analysis

<…>

### Answer to Guide Questions, Insights, and Reflections

<…>

## <Add more activity sections as needed>

|  |
| --- |
| ### --YOUR CODE HERE-- ###  import your mom |

Figure 1.15. Displaying the 5 Tested Noisy to Denoised Images

Figures and Tables

A graph with a line

Description automatically generated

Figure x. Training loss for the autoencoder…

Explain and provide analysis of each plot or model illustration…

Table x. Model accuracy comparison…

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Explain and provide analysis of each table provided…

>

## Activity 6: <Title>

### Objectives

<…>

### Experimental Procedure

<…>

### Results and Analysis

<…>

### Answer to Guide Questions, Insights, and Reflections

<…>

## <Add more activity sections as needed>

|  |
| --- |
| ### --YOUR CODE HERE-- ###  import your mom |

Figure 1.15. Displaying the 5 Tested Noisy to Denoised Images

Figures and Tables

A graph with a line

Description automatically generated

Figure x. Training loss for the autoencoder…

Explain and provide analysis of each plot or model illustration…

Table x. Model accuracy comparison…

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Explain and provide analysis of each table provided…

>

# CONCLUSION AND OVERALL UNDERSTANDING <Replace the guide text with your own conclusions. Approx. 1-2 pages>

* < Summarize your findings on how the fundamental concepts and theory apply to the results of the activity conducted.
* Mention the strengths and weaknesses of applying the concepts and theories in laboratory activities based on your actual model implementation experience.
* Discuss the most significant learning moments and challenges you faced in applying the concepts and how you overcame them.
* Briefly mention how the knowledge gained in these modules will contribute to your understanding of the advanced topics in the future.
* Provide a concluding statement summarizing your overall understanding and your recommendations.>

# REFERENCES <Use APA Citation Style>

* Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

**RUBRIC FOR SUMMATIVE LABORATORY REPORT ASSESSMENT:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CRITERIA** | **EXEMPLARY**  **90-100** | **SATISFACTORY**  **80-89** | **DEVELOPING**  **70-79** | **BEGINNING**  **Below 70** | **WEIGHT** |
| Coverage of Topics | All modules are thoroughly addressed with clear explanations and relevant examples from lab work. | Most modules are addressed adequately with explanations and some examples from lab work. | Some modules are addressed superficially or key concepts are missing. Examples from lab work are limited or unclear. | Many modules are missing or poorly addressed. Understanding of key concepts is not demonstrated. Few or no examples from lab work are provided. | 20% |
| Experiment Procedure ***(SO-PI: B1)*** | Laboratory procedure is logically correct including the method used for evaluating results. | Most of the Laboratory procedure, including the method used for evaluating results, are logically correct. | Only around half of the laboratory procedure, including the method used for evaluating results, is logically correct. | Less than half of the laboratory procedure, including the method used for evaluating results, is logically correct. | 20% |
| Experimental Data  ***(SO-PI: B1)*** | Relevant code snippets from lab work are provided for most key concepts. Data gathered is complete, correct and well documented. Tables and graphs are used as required in the procedure. | Relevant code snippets from lab work are provided for most key concepts. Most of the data expected by the experiment are well documented and correct. Missing one required table or graph. | Code snippets are limited, poorly commented, or not always clearly relevant to the concepts being explained. Around 50% of the data are correct and well documented. Missing two to three required tables or graphs. | Few or no relevant code snippets are provided. The data gathered is severely lacking and many of those gathered are questionable. All the required tables and graphs are missing. | 20% |
| Analysis of Experimental Data  ***(SO-PI: B2)*** | All of the data, including errors were analyzed and interpreted correctly using appropriate theories. Clear and extensive evidence of using appropriate probabilistic, statistical, and data analysis techniques. | Most of the data were analyzed and interpreted correctly using appropriate theories. Clear evidence of using appropriate probabilistic, statistical, and data analysis techniques. | Only around 50% of the data were analyzed and interpreted. Some evidence of using appropriate probabilistic, statistical, and data analysis techniques. | Less than half of the data were analyzed and interpreted. Little or no evidence of appropriate probabilistic, statistical, and data analysis techniques. | 20% |
| Concrete understanding   - writing the conclusion  ***(SO-PI: B2)*** | Summarizes the experiment/activity, cites data or output, cites the causes of error, and suggests a recommendation. Conclusion answers all the objectives. Conclusion made is in agreement with the data gathered. | Demonstrates a good overall understanding of most concepts covered and provides some reflection on the learning experience and challenges faced. Conclusion generally answers most of the objectives. Conclusion is stated on the basis of data gathered. | Conclusion is brief and missing significant pieces of information. Conclusion answers only some of the objectives. Conclusion made has some relation with the data gathered. | Demonstrates a limited understanding of the concepts covered and provides little to no meaningful reflection on the learning experience. There is no connection between the conclusion and the objectives and/or data gathered. | 20% |
|  |  |  |  | TOTAL: | 100% |