**University of Central Missouri**

**Department of Computer Science & Cybersecurity**

**CS5720 Neural network and Deep learning**

**Spring 2025**

**Home Assignment 3. (Cover Ch 7, 8)**

**Student name: Sushma Veluru**

**Submission Requirements:**

* Total Points: 100
* Once finished your assignment push your source code to your repo (GitHub) and explain the work through the ReadMe file properly. Make sure you add your student info in the ReadMe file.
* Submit your GitHub link and video on the BB.
* Comment your code appropriately ***IMPORTANT.***
* Make a simple video about 2 to 3 minutes which includes demonstration of your home assignment and explanation of code snippets.
* Any submission after provided deadline is considered as a late submission.

**Q1: Implementing a Basic Autoencoder**

**Task:** Autoencoders learn to reconstruct input data by encoding it into a lower-dimensional space. You will build a **fully connected autoencoder** and evaluate its performance on image reconstruction.

1. Load the **MNIST dataset** using tensorflow.keras.datasets.
2. Define a **fully connected (Dense) autoencoder**:
   * Encoder: Input layer (784), hidden layer (32).
   * Decoder: Hidden layer (32), output layer (784).
3. Compile and train the autoencoder with **binary cross-entropy loss**.
4. Plot **original vs. reconstructed images** after training.
5. Modify the latent dimension size (e.g., 16, 64) and analyze how it affects the quality of reconstruction.

***Hint:*** *Use Model() from tensorflow.keras.models and Dense() layers.*

**Q2: Implementing a Denoising Autoencoder**

**Task:** Denoising autoencoders can reconstruct clean data from noisy inputs. You will train a model to remove noise from images.

1. Modify the **basic autoencoder** from Q2 to a **denoising autoencoder** by adding **Gaussian noise** (mean=0, std=0.5) to input images.
2. Ensure that the **output remains the clean image** while training.
3. Train the model and visualize **noisy vs. reconstructed images**.
4. Compare the **performance of a basic vs. denoising autoencoder** in reconstructing images.
5. Explain one real-world scenario where denoising autoencoders can be useful (e.g., medical imaging, security).

***Hint:*** *Use np.random.normal() to add noise to images before training.*

**Q3: Implementing an RNN for Text Generation**

**Task:** Recurrent Neural Networks (RNNs) can generate sequences of text. You will train an **LSTM-based RNN** to predict the next character in a given text dataset.

1. Load a **text dataset** (e.g., "Shakespeare Sonnets", "The Little Prince").
2. Convert text into a **sequence of characters** (one-hot encoding or embeddings).
3. Define an **RNN model** using LSTM layers to predict the next character.
4. Train the model and generate new text by **sampling characters** one at a time.
5. Explain the role of **temperature scaling** in text generation and its effect on randomness.

***Hint:*** *Use tensorflow.keras.layers.LSTM() for sequence modeling.*

**Q4: Sentiment Classification Using RNN**

**Task:** Sentiment analysis determines if a given text expresses a positive or negative emotion. You will train an **LSTM-based sentiment classifier** using the IMDB dataset.

1. Load the **IMDB sentiment dataset** (tensorflow.keras.datasets.imdb).
2. Preprocess the text data by **tokenization** and **padding** sequences.
3. Train an **LSTM-based model** to classify reviews as **positive or negative**.
4. Generate a **confusion matrix** and classification report (accuracy, precision, recall, F1-score).
5. Interpret why **precision-recall tradeoff** is important in sentiment classification.

***Hint:*** *Use confusion\_matrix and classification\_report from sklearn.metrics.*