19CSE456 Neural Network and Deep Learning Laboratory

List of Experiments

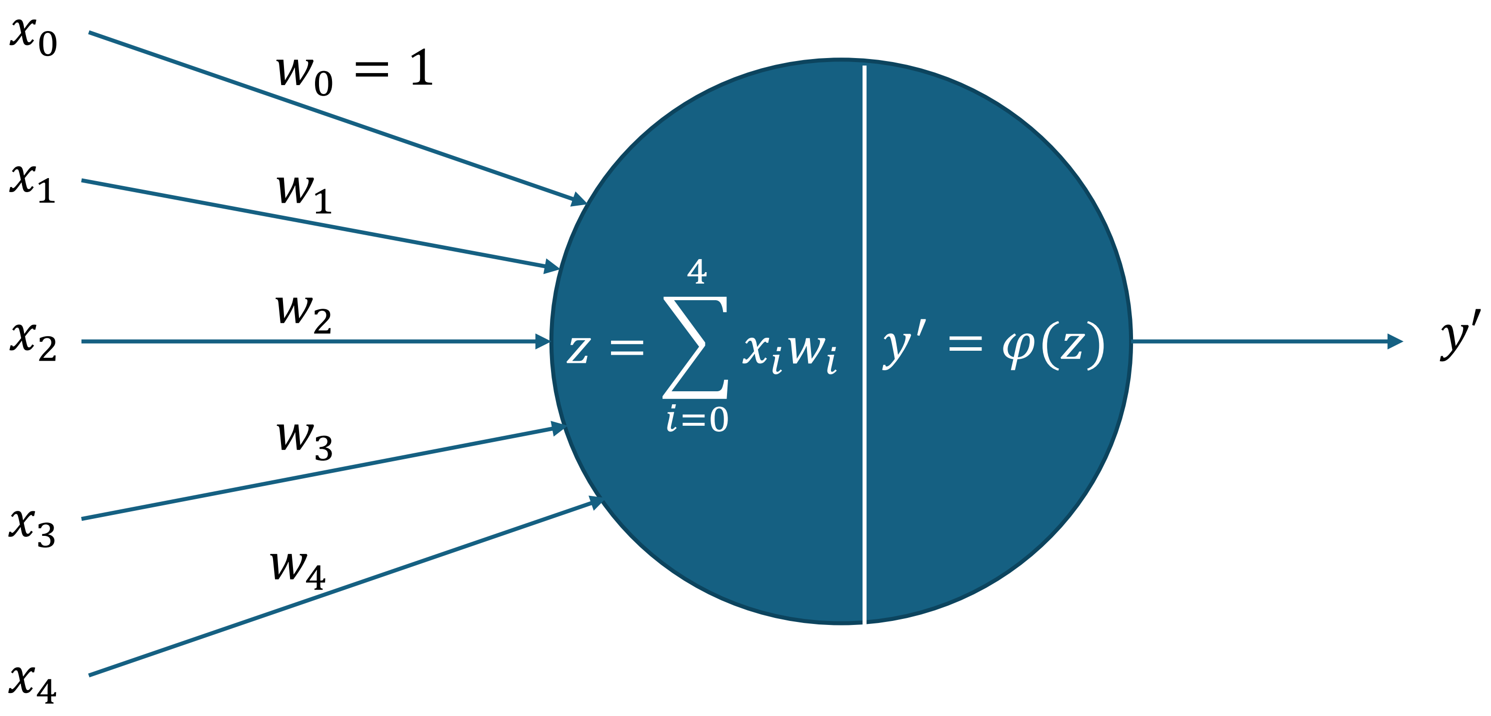
Week #	Experiment Title
1	Introduction to the lab and Implementation of a simple Perceptron (Hardcoding)
2	Implementation of Perceptron for Logic Gates (Hardcoding, Sklearn, TF)
3	Implementation of Multilayer Perceptron for XOR Gate and other classification problems with ML toy datasets (Hardcoding & TF)
4	Implementation of MLP for Image Classification with MNIST dataset (Hardcoding & TF)
5	Activation Functions, Loss Functions, Optimizers (Hardcoding & TF)
6	Lab Evaluation 1 (based on topics covered from w1 to w5)
7	Convolution Neural Networks for Toy Datasets (MNIST & CIFAR)
8	Convolution Neural Networks for Image Classification (Oxford Pets, Tiny ImageNet, etc.)
9	Recurrent Neural Networks for Sentiment Analysis with IMDB Movie Reviews
10	Long Short Term Memory for Stock Prices (Yahoo Finance API)

List of Experiments contd.

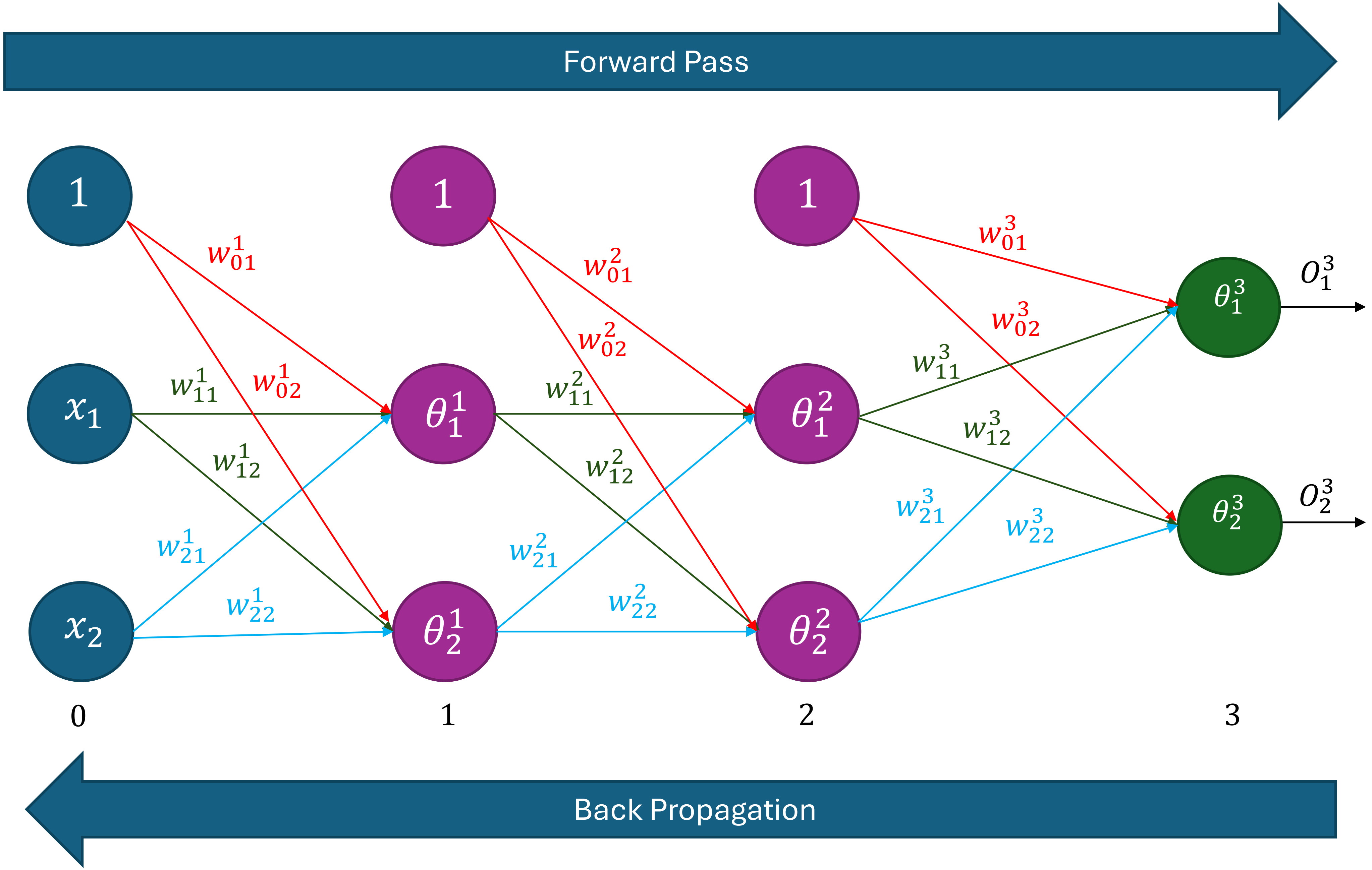
Week #	Experiment Title
11	Implementation of Autoencoders and Denoising Autoencoders (MNIST/CIFAR)
12	Boltzmann Machines (MNIST/CIFAR)
13	Restricted Boltzmann Machines (MNIST/CIFAR)
14	Hopfield Neural Networks (MNIST/CIFAR)
15	Lab Evaluation 2 (based on CNN, RNN, LSTM, and AEs)
16	Case Study Review (Phase 1)
17	Case Study Review (Phase 1)

Perceptron

- A single-layer perceptron is the basic unit of a neural network
- A perceptron consists of input values, weights and a bias, a weighted sum and activation function



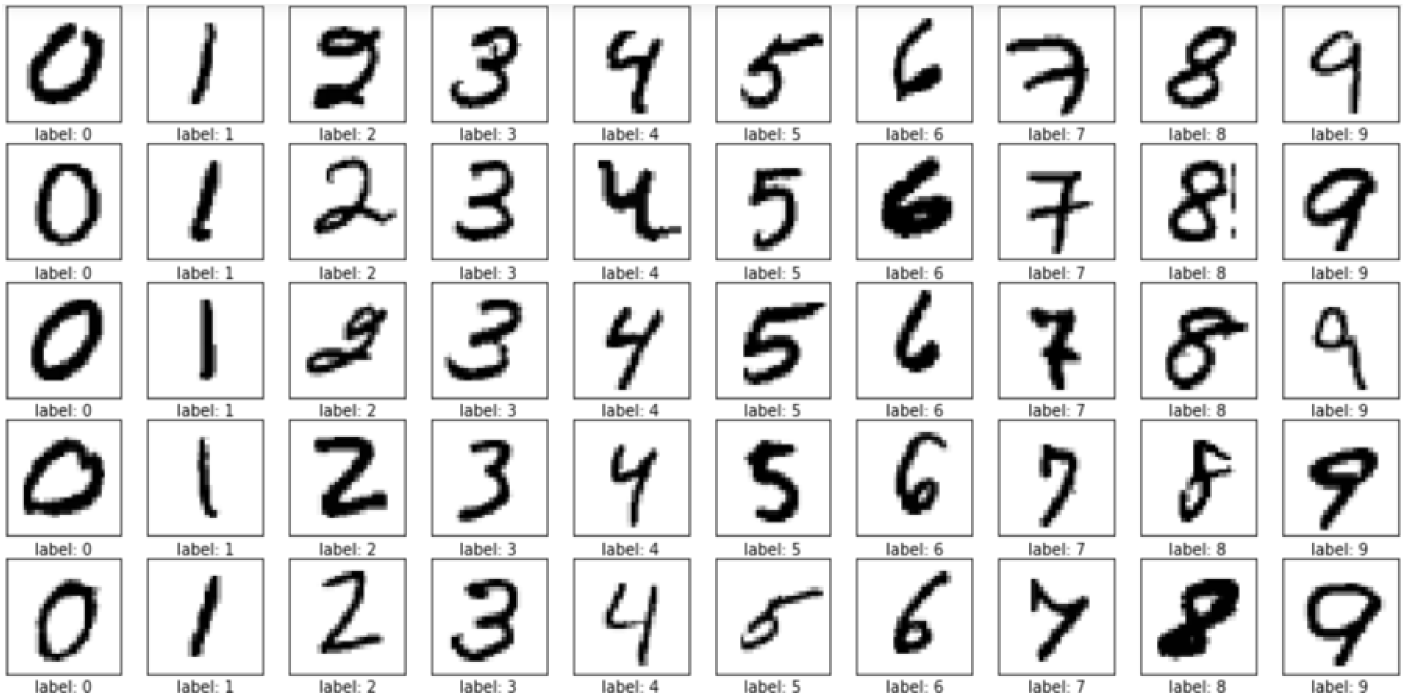
MLP



Forward Pass
1
x_1
x_2
1
θ_1^1
θ_2^1
1
θ_1^2
θ_2^2
θ_1^3
θ_2^3
O_1^3
O_2^3
w_{01}^1
w_{02}^1
w_{11}^1
w_{12}^1
w_{21}^1
w_{22}^1
w_{01}^2
w_{02}^2
w_{11}^2
w_{12}^2
w_{21}^2
w_{22}^2
w_{01}^3
w_{02}^3
w_{11}^3
w_{12}^3
w_{21}^3
w_{22}^3
0
1
2
3
Back Propagation

MNIST Dataset

The MNIST dataset (Modified National Institute of Standards and Technology) is one of the most well-known datasets in the field of machine learning and computer vision



- The dataset consists of 70,000 grayscale images of handwritten digits from 0 to 9
- Each image is 28x28 pixels, providing a total of 784 features per image

MLP for Image Classification

Load and Preprocess the Data → Model Creation → Model Training → Model Evaluation

```
def load_and_preprocess_data():
    # Load MNIST dataset
    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

    # Normalize pixel values
    x_train = x_train.astype('float32') / 255.0
    x_test = x_test.astype('float32') / 255.0

    # Reshape images to 1D arrays
    x_train = x_train.reshape(-1, 28*28)
    x_test = x_test.reshape(-1, 28*28)

    # One-hot encode labels
    y_train = tf.keras.utils.to_categorical(y_train, 10)
    y_test = tf.keras.utils.to_categorical(y_test, 10)

    return (x_train, y_train), (x_test, y_test)
```

Normalizes the pixel values of the images to be in the range [0, 1] and convert them to floating-point numbers

Reshapes each 28 × 28 pixel image into a 1D array of length 784 (28*28)

Allows the number of images to be inferred automatically

Converts the integer labels (0-9) into one-hot encoded vectors

MLP for Image Classification

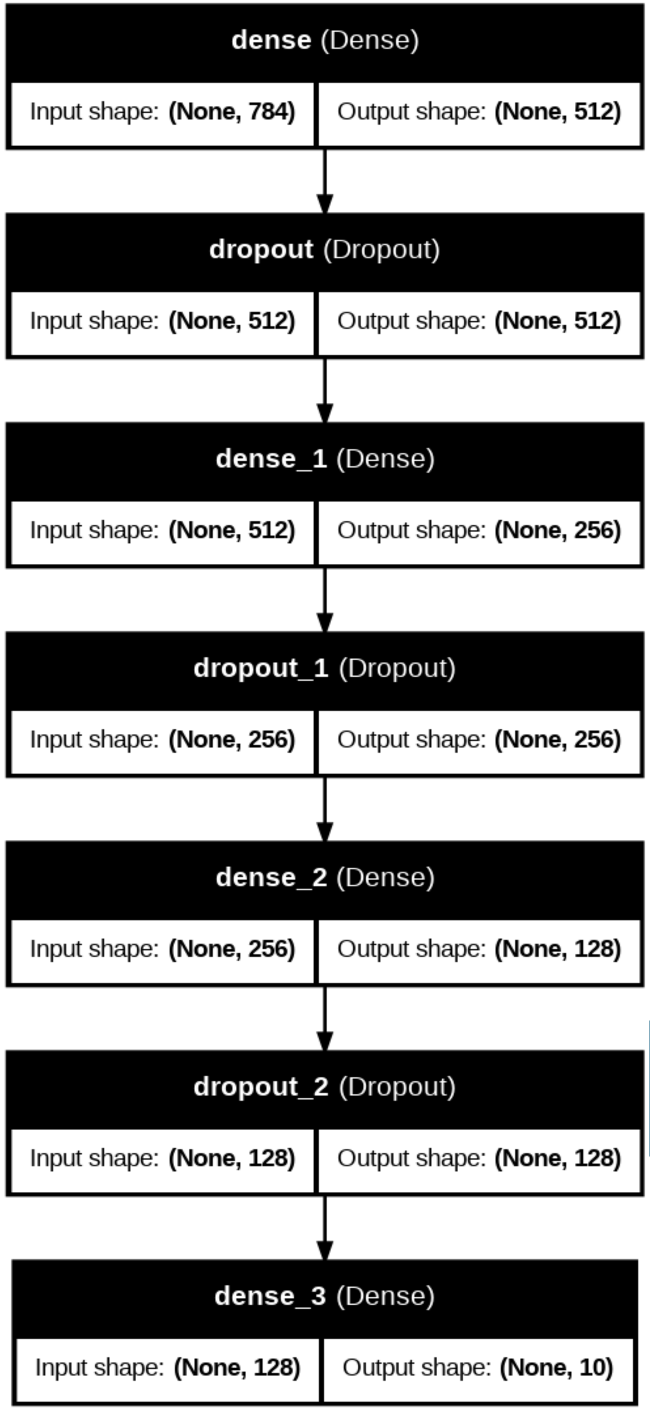
Load and Preprocess the Data → Model Creation → Model Training → Model Evaluation

```
def create_model():
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(512, activation='relu', input_shape=(784,)),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(256, activation='relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	401,920
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32,896
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1,290

MLP Visualization




```
from tensorflow.keras.utils import plot_model

tf.keras.utils.plot_model(
    model,
    to_file="model.png",
    show_shapes=True,
    show_layer_names=True,
    rankdir="TB",
    expand_nested=False,
    dpi=96,
)

from IPython.display import Image
Image('model.png')
```

The plot_model function is used to visualize the architecture of a Keras model and save it as an image file

Sets the direction of the graph (“TB” / ”LR”)

Displays the image "model.png" within the IPython environment

MLP for Image Classification

Load and Preprocess the Data → Model Creation → **Model Training** → Model Evaluation

```
def train(model, x_train, y_train, x_test, y_test):
    # Train the model
    history = model.fit(x_train, y_train,
                  batch_size=128,
                  epochs=20,
                  validation_split=0.2,
                  verbose=1)
```

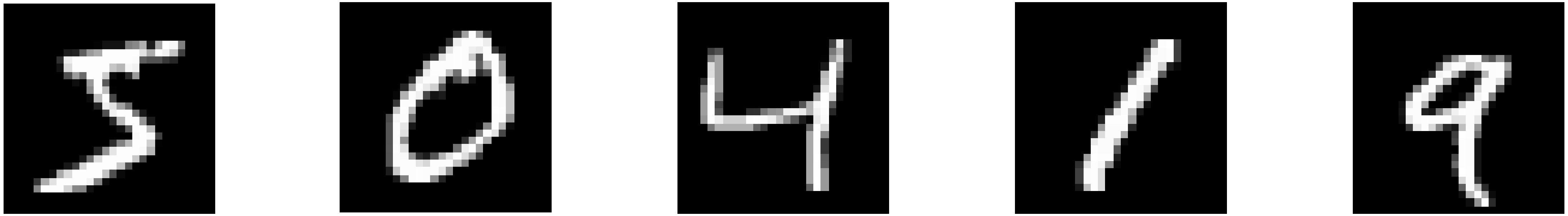
During each epoch, the model will update its weights after processing 128 samples (batch size)

The model will train for 20 epochs, meaning it will process the entire training dataset 20 times

B_1	B_2	B_3	B_4		B_n
1	1	1	1		1
2	2	2	2		2
3	3	3	3	…	3
…	…	…	…		…
128	128	128	128		128

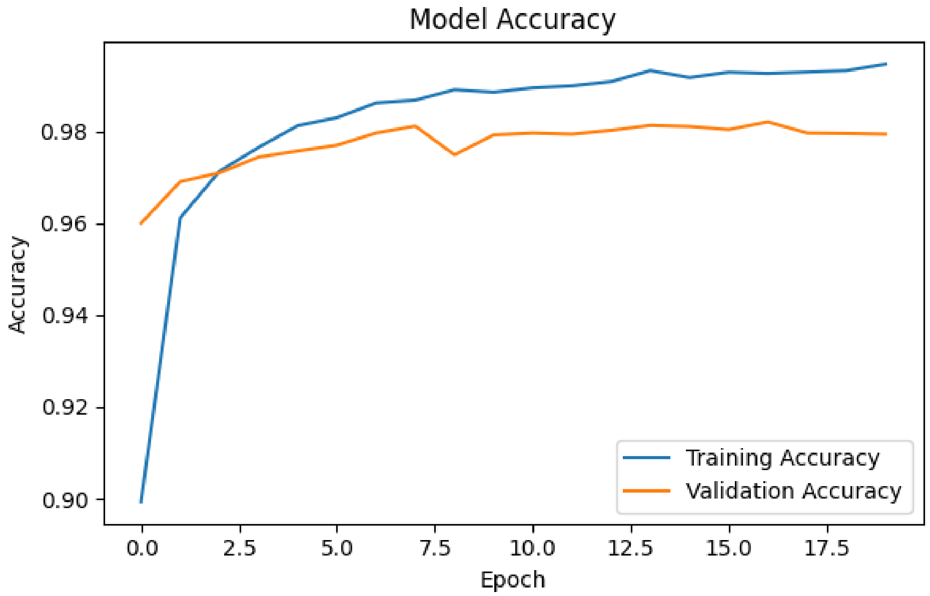
MLP for Image Classification

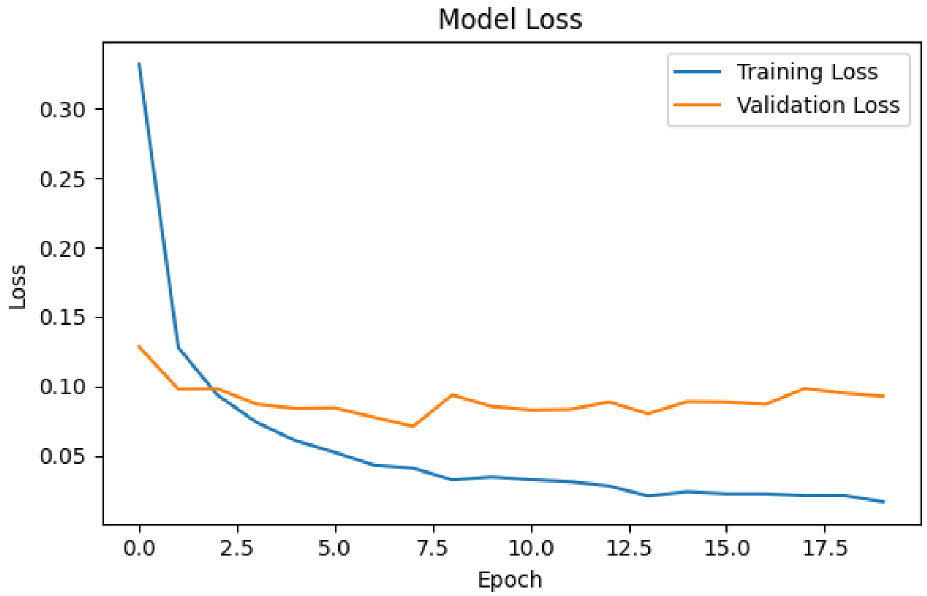
Label: 5 Label: 0 Label: 4 Label: 1 Label: 9

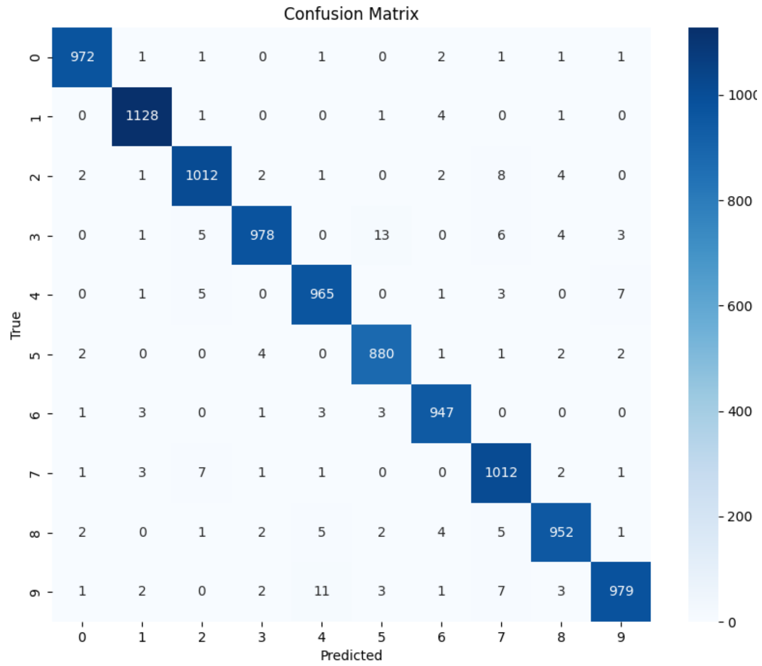


```
Training model...
Epoch 1/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8096 - loss: 0.6075 - val_accuracy: 0.9600 - val_loss: 0.1284
Epoch 2/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9593 - loss: 0.1305 - val_accuracy: 0.9691 - val_loss: 0.0980
Epoch 3/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9719 - loss: 0.0907 - val_accuracy: 0.9710 - val_loss: 0.0983
Epoch 4/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9755 - loss: 0.0738 - val_accuracy: 0.9744 - val_loss: 0.0871
Epoch 5/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9820 - loss: 0.0572 - val_accuracy: 0.9758 - val_loss: 0.0839
Epoch 6/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9838 - loss: 0.0492 - val_accuracy: 0.9770 - val_loss: 0.0843
Epoch 7/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9880 - loss: 0.0403 - val_accuracy: 0.9797 - val_loss: 0.0776
Epoch 8/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9879 - loss: 0.0355 - val_accuracy: 0.9812 - val_loss: 0.0710
Epoch 9/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9891 - loss: 0.0317 - val_accuracy: 0.9749 - val_loss: 0.0937
Epoch 10/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9888 - loss: 0.0355 - val_accuracy: 0.9793 - val_loss: 0.0854
Epoch 11/20
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9906 - loss: 0.0280 - val_accuracy: 0.9797 - val_loss: 0.0828
Epoch 12/20
```

MLP for Image Classification








```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.98      0.98      1032
           3       0.99      0.97      0.98      1010
           4       0.98      0.98      0.98       982
           5       0.98      0.99      0.98       892
           6       0.98      0.99      0.99       958
           7       0.97      0.98      0.98      1028
           8       0.98      0.98      0.98       974
           9       0.98      0.97      0.98      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

Week 4 Exercises

1. MLP Classifier for MNIST Handwritten Digits

Objective: To build, train, evaluate, and visualize the performance of an MLP image classifier using the MNIST dataset.

2. **MLP Classifier for CIFAR-10 Dataset**

Objective: To build, train, evaluate, and visualize the performance of an MLP image classifier using the CIFAR-10 dataset.

- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class.
- There are 50000 training images and 10000 test images.

https://www.cs.toronto.edu/~kriz/cifar.html

Pushing Your Code GitHub Repository

Clone the Repository

git clone https://github.com/YourUsername/MLP_SKL_TF_W4.git

Navigate to the Repository

cd MLP_SKL_TF_W4

Create a New Branch

git checkout -b <<Your_Roll_No>>

Add Your Code Folder

mkdir <<MyCodeFolder>>
cd <<MyCodeFolder>>

Pushing Your Code GitHub Repository

Add and Commit Changes

git add MyCodeFolder
git commit -m "Add MyCodeFolder"

Push Changes to the Repository

git push origin add-new-code-folder

Create a Pull Request