LAB05 TensorFlow exercise

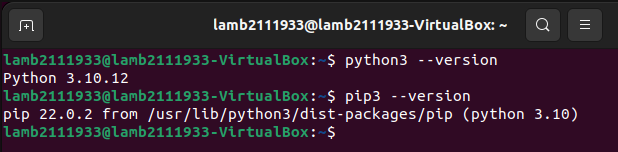
| Class | CT201H [M01-M04] |
| --- | --- |
| Student ID | B2111933 |
| Name | Truong Dang Truc Lam |
| Email address | lamb2111933@student.ctu.edu.vn |
| Class | CT201H [M01-M04] |
| Browser | Chrome |

Choose one topic, exercise as far as you do (write reference site)

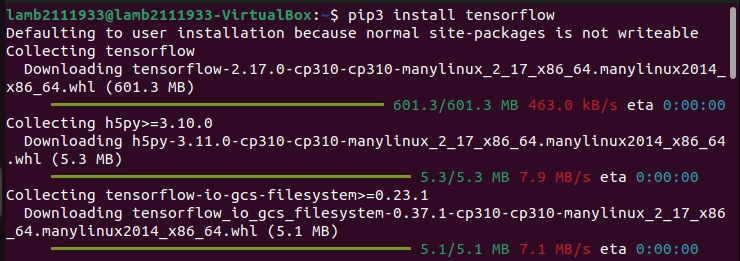
Reference site : Gemini, Bing, chat GPT, WRTN, git-HUB

I choose Gemini

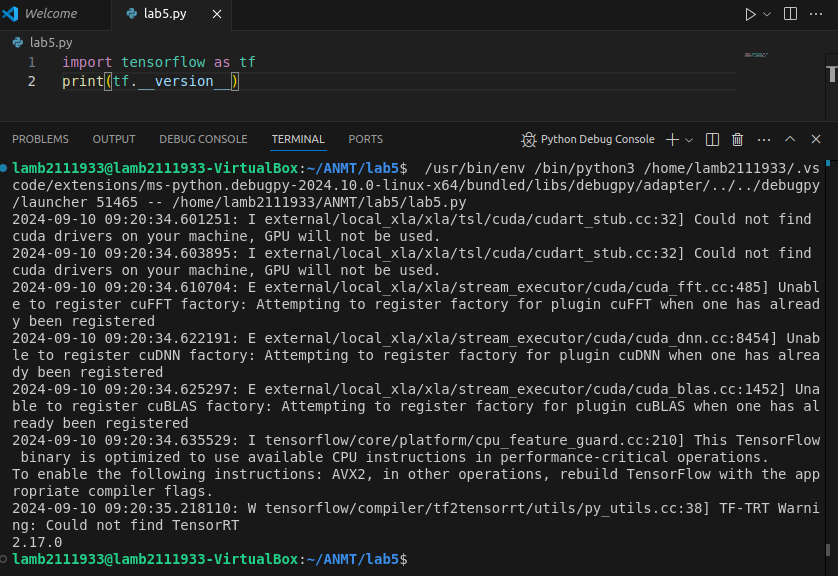
Install Python Libraries



Install python and pip



Install tensorflow



Because I’m doing this project on a VM, I will not use tensorflow with gpu

[Model1]

1. Survey the basic steps of coding using TensorFlow from reference site

According to [Gemini](https://gemini.google.com/):

Basic Steps of Coding with TensorFlow

TensorFlow is a popular open-source machine learning framework used for building deep learning models. Here are the basic steps involved in coding with TensorFlow:

1. Import Necessary Libraries:

* Import the TensorFlow library using import tensorflow as tf.
* Import other necessary libraries like NumPy (import numpy as np) for numerical operations, Matplotlib (import matplotlib.pyplot as plt) for visualization, and more.

2. Prepare Your Data:

* Load Data: Load your dataset into a suitable format (e.g., NumPy arrays, Pandas DataFrames).
* Preprocess Data: Normalize or standardize your data if necessary. Handle missing values or outliers.
* Split Data: Divide your data into training, validation, and testing sets to evaluate model performance.

3. Define Model Architecture:

* Choose Model Type: Select a suitable model architecture based on your problem (e.g., neural network, convolutional neural network, recurrent neural network).
* Create Layers: Define the layers of your model using TensorFlow's layer API (e.g., tf.keras.layers.Dense, tf.keras.layers.Conv2D, tf.keras.layers.LSTM).
* Compile Model: Specify the loss function, optimizer, and metrics to evaluate the model's performance.

4. Train the Model:

* Fit Model: Train the model on the training data using the fit method.
* Adjust Hyperparameters: Experiment with different hyperparameters (e.g., learning rate, batch size, number of epochs) to improve model performance.
* Monitor Training: Track the model's progress using metrics like loss and accuracy.

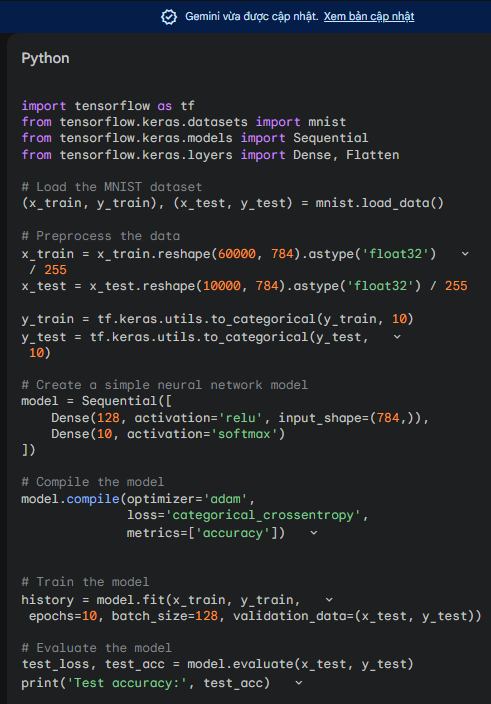
5. Evaluate Model Performance:

* Evaluate on Validation Set: Assess the model's performance on the validation set to avoid overfitting.
* Make Predictions: Use the trained model to make predictions on new data.
* Evaluate Metrics: Calculate relevant metrics (e.g., accuracy, precision, recall, F1-score) to evaluate the model's effectiveness.

6. Save and Load Model:

* Save Model: Save the trained model for future use using tf.keras.models.save.
* Load Model: Load a saved model using tf.keras.models.load\_model.

1. Search the example code of basic steps



Example of basic steps with TensorFlow on the MNIST dataset -Gemini-

1. Explain the process and code

1. Import necessary libraries:

* tensorflow: The core library for machine learning and deep learning tasks.
* mnist: A dataset of handwritten digits used for training and testing machine learning models.
* Sequential: A class for creating a linear stack of layers.
* Dense: A fully connected layer.
* Flatten: A layer that flattens the input data to a 1D vector.

2. Load the MNIST dataset:

Loads the MNIST dataset into two tuples:

* (x\_train, y\_train): Training data with images and corresponding labels.
* (x\_test, y\_test): Testing data with images and corresponding labels.

3. Preprocess the data:

* Reshapes the image data from 28x28 pixel matrices to 1D vectors of 784 elements.
* Converts the data type to float32 for numerical calculations.
* Normalizes the pixel values to the range [0, 1] for better training.
* Converts the labels to one-hot encoded format, where each label is represented as a 10-dimensional vector with a 1 at the corresponding class index and 0s elsewhere.

4. Create a neural network model:

Creates a Sequential model with two layers:

* Input layer: A Dense layer with 128 neurons and ReLU activation. The input\_shape argument specifies the shape of the input data.
* Output layer: A Dense layer with 10 neurons (corresponding to the number of classes) and softmax activation. Softmax ensures that the output values sum to 1, representing probabilities for each class.

5. Compile the model:

Configures the model for training:

* Optimizer: Adam is a popular optimization algorithm that efficiently updates the model's weights during training.
* Loss function: Categorical crossentropy is used to measure the model's performance, especially for multi-class classification tasks.
* Metrics: Accuracy is used to evaluate the model's performance on the test set.

6. Train the model:

Trains the model on the training data:

* x\_train and y\_train: Training data.
* epochs: Number of times the entire training set is passed through the model.
* batch\_size: Number of samples processed at once during training.
* validation\_data: Testing data used to evaluate the model's performance during training.

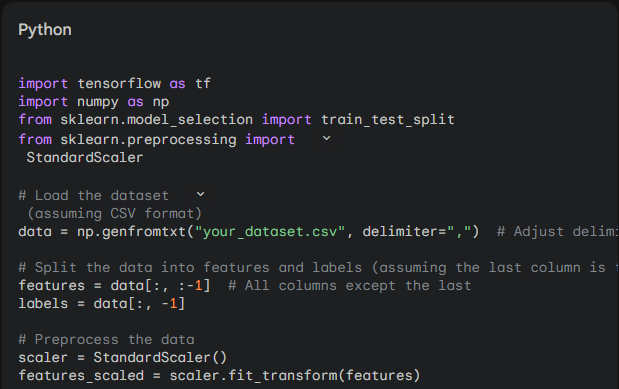
7. Evaluate the model:

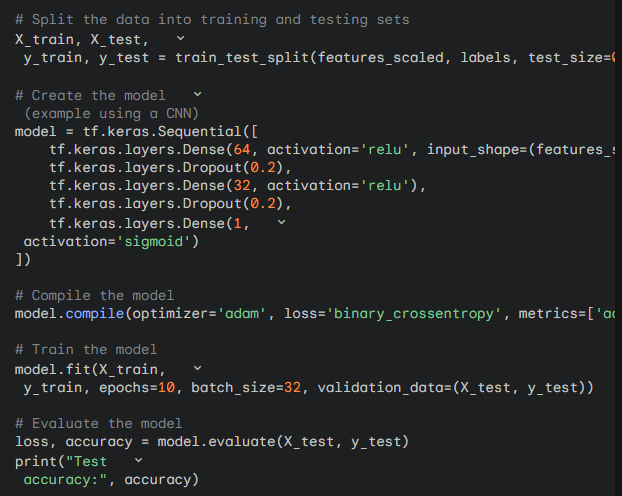
Evaluates the model's performance on the test data:

* x\_test and y\_test: Testing data.
* Returns the test loss and test accuracy.
* Prints the test accuracy.

[Model2]

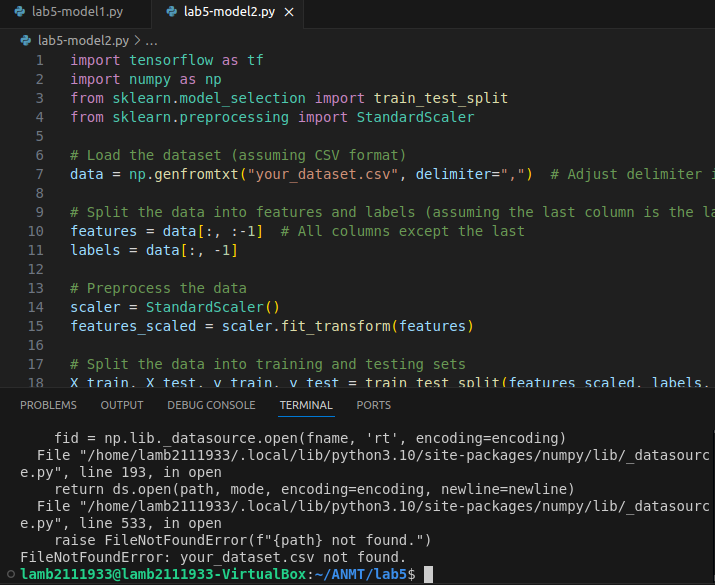
1. Code example of detecting SYN Flooding in TensorFlow reference site





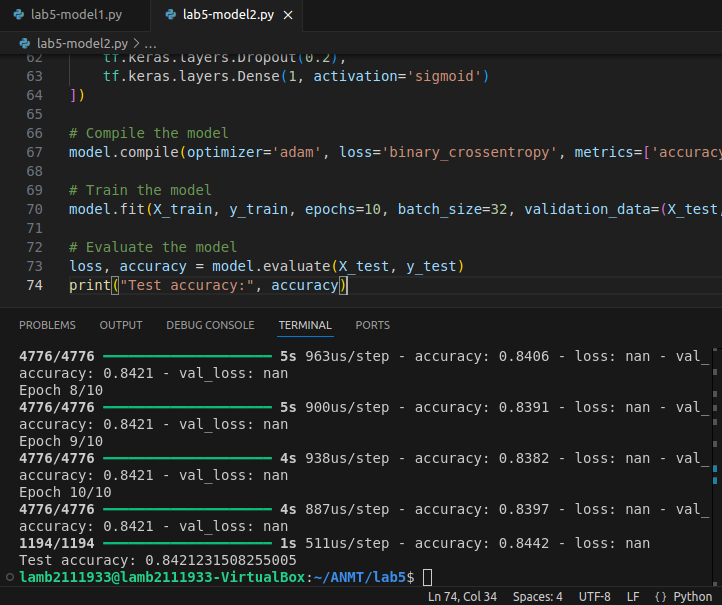
Code from Gemini

1. Execute the example code



It got some errors so I will fix it

1. Check the error msg. fix and modify the code



Code modified

1. Explain your code

import tensorflow as tf

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, RobustScaler # RobustScaler for potential outliers

from sklearn.impute import SimpleImputer # for missing value imputation

# Load the dataset

data = np.genfromtxt("/home/lamb2111933/Downloads/MachineLearningCVE/Friday-WorkingHours-Morning.pcap\_ISCX.csv", delimiter=",")

# Check for infinite and missing values

inf\_rows = np.isinf(data).any(axis=1) # Find rows with any infinity

missing\_rows = np.isnan(data).any(axis=1) # Find rows with any missing values

# Handle infinite values

if inf\_rows.any():

# Check for all rows with infinity

if inf\_rows.all():

print("Error: All data points contain infinite values. Data cleaning is required.")

exit() # Exit if all data is unusable

else:

# Clip remaining infinite values (adjust clip values as needed)

data = np.clip(data, a\_min=-1e10, a\_max=1e10)

# Handle missing values (consider alternative strategies if needed)

if missing\_rows.any():

# Impute missing values (adjust strategy if necessary)

imputer = SimpleImputer(strategy='mean') # Adjust strategy if necessary

data = imputer.fit\_transform(data)

else:

print("No missing values found in the data.")

# Check for empty data

if data.shape[0] == 0:

print("Error: No data remaining after handling missing values and infinite values. Please check your data source.")

exit() # Exit the program if no data is available

# Split features and labels

features = data[:, :-1]

labels = data[:, -1]

# Preprocess the data

# Use RobustScaler for potential outliers

scaler = RobustScaler()

# Check if data has any rows before scaling (to avoid the error)

if data.shape[0] > 0:

features\_scaled = scaler.fit\_transform(features)

else:

print("Error: No data to scale. Skipping model training.")

exit() # Exit the program if no data is available

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features\_scaled, labels, test\_size=0.2, random\_state=42)

# ... rest of your model building and evaluation code (same as before)

# Create the model (example using a CNN)

model = tf.keras.Sequential([

tf.keras.layers.Dense(64, activation='relu', input\_shape=(features\_scaled.shape[1],)),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(32, activation='relu'),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(1, activation='sigmoid')

])

# Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(X\_test, y\_test))

# Evaluate the model

loss, accuracy = model.evaluate(X\_test, y\_test)

print("Test accuracy:", accuracy)

This is my modified version

Explain:

The provided code performs several key steps in preparing your data for training a machine learning model using TensorFlow:

**1. Loading and Checking Data:**

* **np.genfromtxt:** This function loads the data from the CSV file located at the specified path.
* **inf\_rows and missing\_rows:** These lines check for infinite and missing values in the data, respectively. They use np.isinf and np.isnan to identify rows with these issues.

**2. Handling Infinite Values:**

* The code checks if any rows contain infinite values using inf\_rows.any().
* If all rows have infinite values, an error message is printed, and the program exits.
* Otherwise, the remaining infinite values are "clipped" to a specific range using np.clip. You can adjust the a\_min and a\_max values based on your data.

**3. Handling Missing Values (if any):**

* The code checks if any rows have missing values using missing\_rows.any().
* If missing values are found, a SimpleImputer with the default strategy of using the mean is applied to fill them. This can be adjusted to other strategies like 'median' for skewed data.
* If no missing values are found, an informative message is printed.

**4. Checking for Empty Data:**

* The code checks if any data points remain after handling infinite and missing values using data.shape[0].
* If no data is left, an error message is printed, and the program exits.

**5. Splitting Features and Labels:**

* The code separates the features (data points) from the labels (target variable) using slicing.
* features contains all columns except the last one, which is assumed to be the label.
* labels contains only the last column, representing the target variable.

**6. Preprocessing Data (Scaling):**

* A RobustScaler is used to normalize the features for better model performance. This is particularly useful for dealing with outliers.
* The code checks if there's any data before scaling to avoid errors.
* If data exists, features\_scaled holds the scaled features.

**7. Splitting Data into Training and Testing Sets:**

* train\_test\_split splits the features and labels into training and testing sets. The training set is used to train the model, and the testing set evaluates its performance.
* X\_train, X\_test, y\_train, and y\_test hold the respective training and testing features and labels.

**8. Model Building and Evaluation (Placeholder):**

* The provided code doesn't include the specific model building and evaluation steps.
* Here's a placeholder comment (... rest of your model building and evaluation code (same as before)) suggesting you replace it with your model definition, compilation, training, and evaluation code using TensorFlow.

**9. Creating and Training a Sample Model:**

* The code shows an example of creating a simple neural network model using tf.keras.Sequential. This is just a demonstration, and you'll likely use a different architecture based on your problem.
* The model trains for 10 epochs with a batch size of 32 and uses validation data during training.

**10. Evaluating the Model:**

* Finally, the model's performance is evaluated on the testing set using model.evaluate. The test accuracy is then printed.

**Overall, this code snippet provides a solid foundation for preparing your data and building a machine learning model with TensorFlow. Remember to adjust the hyperparameters and model architecture to suit your specific dataset and problem.**