DA5402: Machine Learning Operations Laboratory

Assignment 8

Nandhakishore C S (DA24M011)

April 14, 2025

Problem Statement

Consider the following code base to build a handwriting recognition model using the tensorflow/keras combination.

https://keras.io/examples/vision/handwriting$_{r}ecognition$/. With MLFlow do the following

Task 1 [30 points]

Setup the MLflow ecosystem and ensure that you are able to reproduce the examples that were shown in the class for tracking, API packaging and MLprojects [no points for this!]. Now, refactor the handwriting recognition codebase to infuse MLflow tracking to track the metrics, parameters, models, and other relevant artifacts. Add logging and exception handling as necessary. Repea the model build at least 3 times by changing the train-val-test splits. Register the respective model versions. Generate the following graphs:

1. Epochs vs Training & Validation losses
2. Epochs vs Average Edit distance

A custom CNN model with a CNN layer with 2 max pooling and 2 dense layers is created with different splits. The model is logged in ML Flow and the metrics are visualized using the ML Flow UI.

Setup

Create a Python virtual environment and install the depenices in the file requirements.txt (Refer the README file in the github repository for more details.)
The model configuration is as follows:

```
# Build the model (simplified CNN)
def build_model():
    model = keras.Sequential([
        keras.layers.Input(shape=(28, 28, 1)),
        keras.layers.Conv2D(32, (3, 3), activation="relu"),
        keras.layers.MaxPooling2D((2, 2)),
        keras.layers.MaxPooling2D((2, 2)),
        keras.layers.Flatten(),
        keras.layers.Dense(128, activation="relu"),
        keras.layers.Dense(128, activation="relu"),
        keras.layers.Dense(10, activation="softmax")
    ])
    return model
```

Run the model and Start the MLFLow server (in the system, port number 8000 is used). The MLflow UI can be accseed in localhost:8000 The Graphs are visualized as follows:

Task 2 [20 points]

Use the MLflow APIfication to expose the handwritting recognition model (the most performant version) as a REST API. You should demonstrate the handwriting recognition ability by inputting an image of from test dataset and the API end point should return the extracted text from the image. You may use Postman or curl as your API client to send the image to the API service.
The Model with the lowest validation loss is choosen and predictions are done using the *curl* command. The API is exposed at port number 54320 and a dummy image from the internet is fed into the model as a .json file. // The image is processed into a .json file as follows:

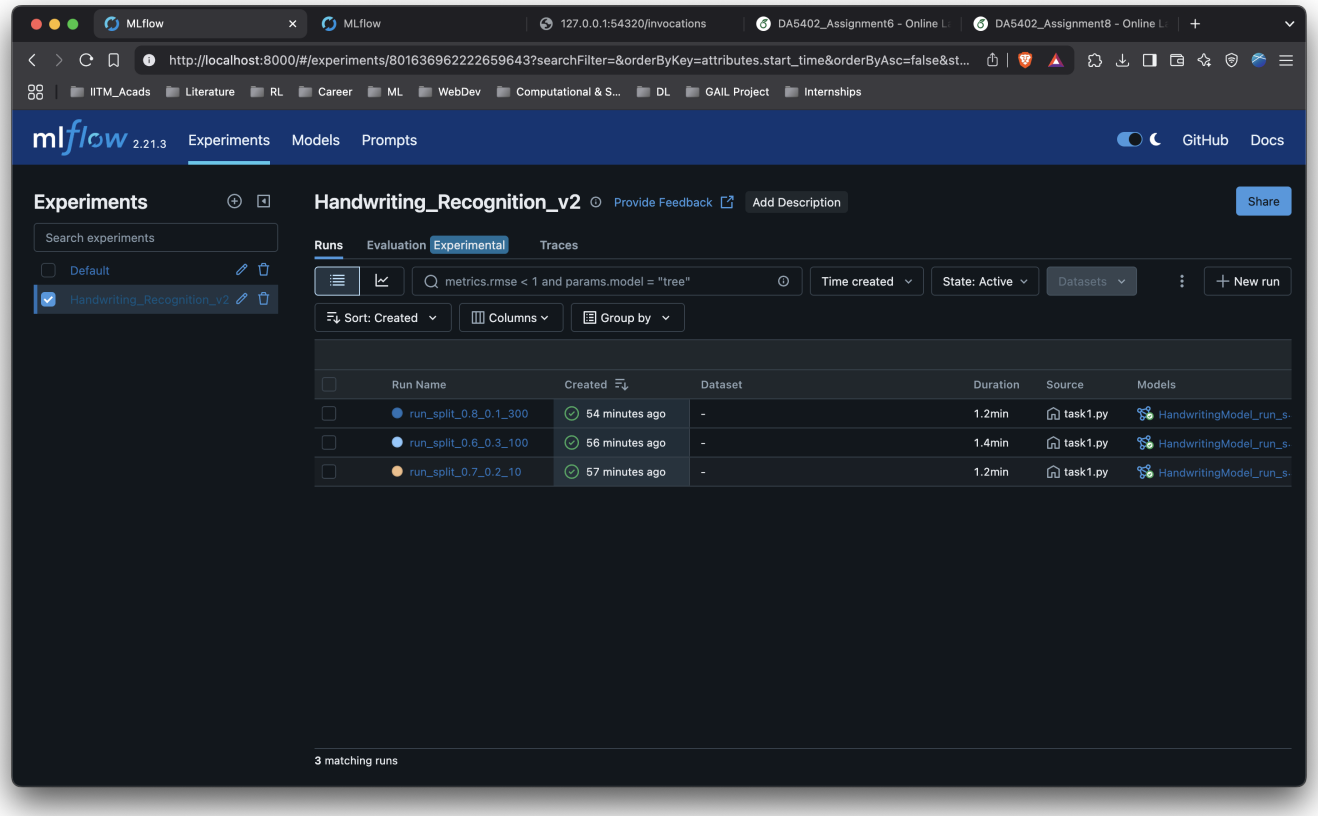

Figure 1: ML FLow UI with the three different model runs

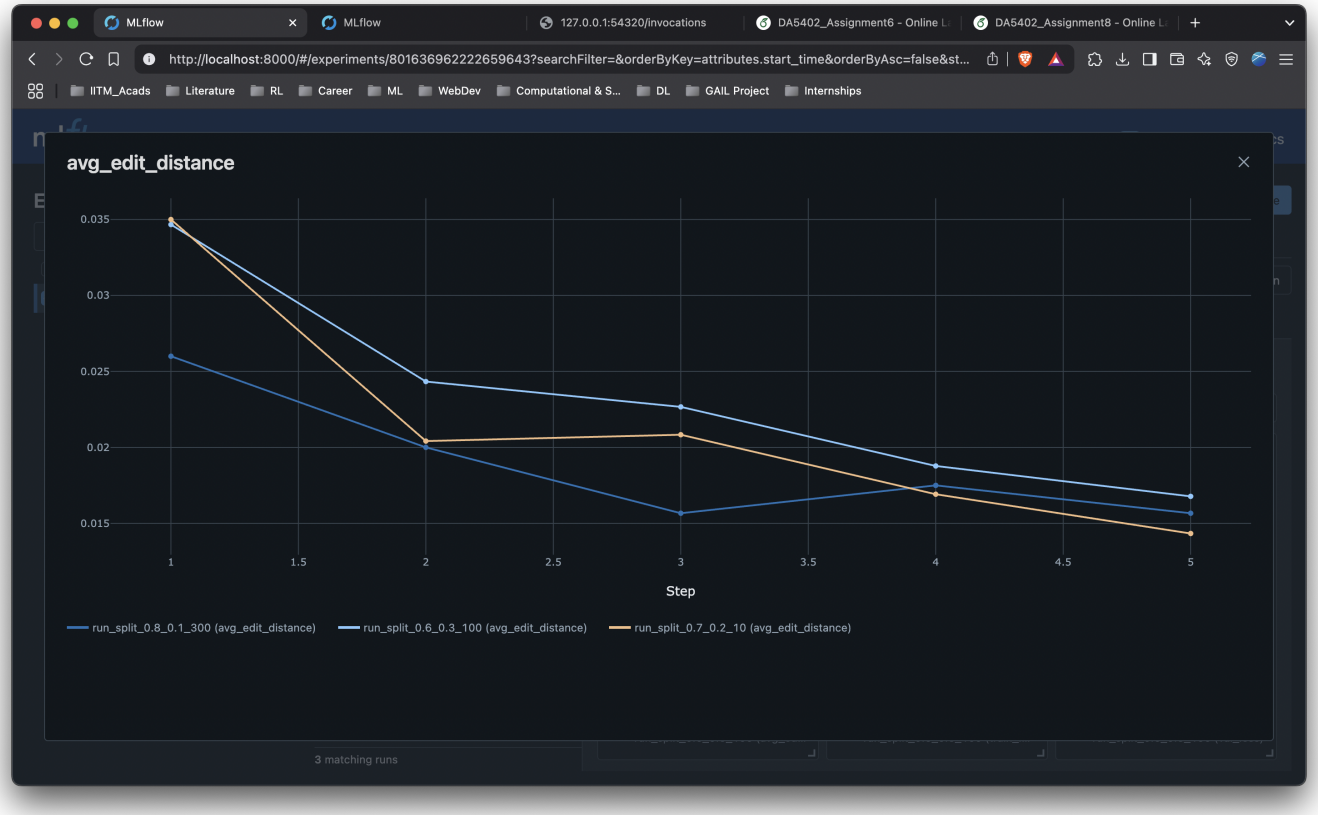

Figure 2: Average Edit distance vs Epochs

```
def preprocess_image(image_path, output_json="image_data.json"):
    try:
```

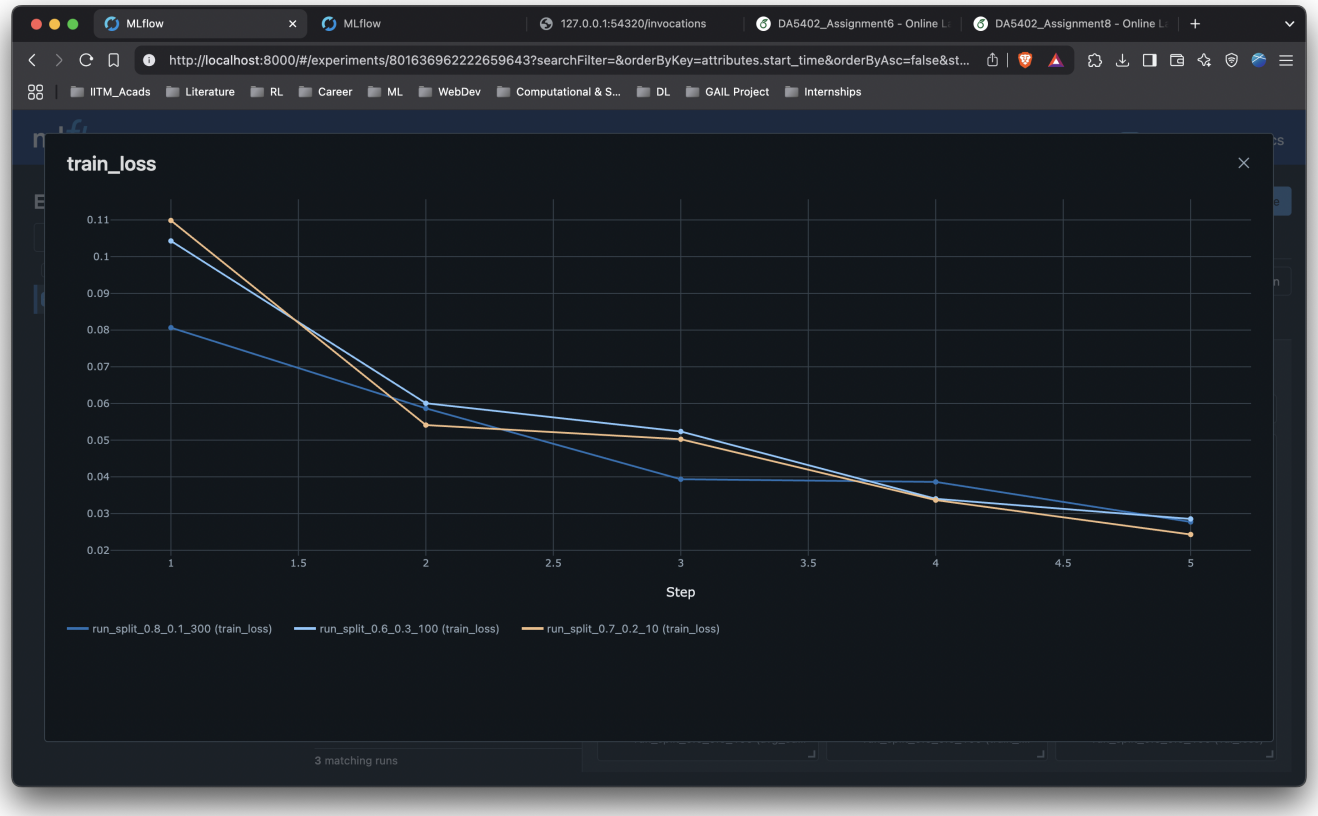

Figure 3: Training Loss vs Epochs

Figure 4: Validation Loss vs Epochs

```
        img = Image.open(image_path).convert("L")  # Grayscale
        img = img.resize((28, 28))  # Match model input shape
```

```
        img_array = np.array(img).astype("float32") / 255.0
        img_array = np.expand_dims(img_array, axis=(0, -1))  # Shape: (1, 28, 28, 1)
        payload = {"inputs": img_array.tolist()}
        with open(output_json, "w") as f:
            json.dump(payload, f)
        logger.info(f"Saved preprocessed image data to {output_json}")
        return payload
    except Exception as e:
        logger.error(f"Error preprocessing image: {str(e)}")
        raise

if __name__ == "__main__":
    image_path = input("Enter the path to the image: ")
    preprocess_image(image_path)
```

The local image used for testing is as follows: The Json dump of the file is as follows:

Figure 5: Dummy Image for testing

```
{"inputs": [[[[1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0],
    [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0],
.
.
.
```

Figure 6: The prediction for the dummy image file with written number can be seen on the CLI

```
[1, 0], [1.0], [1.0]], [[1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [0.8941176533699036],
    [0.5058823823928833], [0.3176470696926117], [0.3019607961177826], [0.3843137323856354],
    [0.5176470875740051], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0]]]]}
```

The prediction via API is done by POST method and the scrrenshot can be found below: