

# Data Intake Report

Name: <Digit Recognition>

Report date: <>

Internship Batch:<LISUM19>

Version:<1.0>

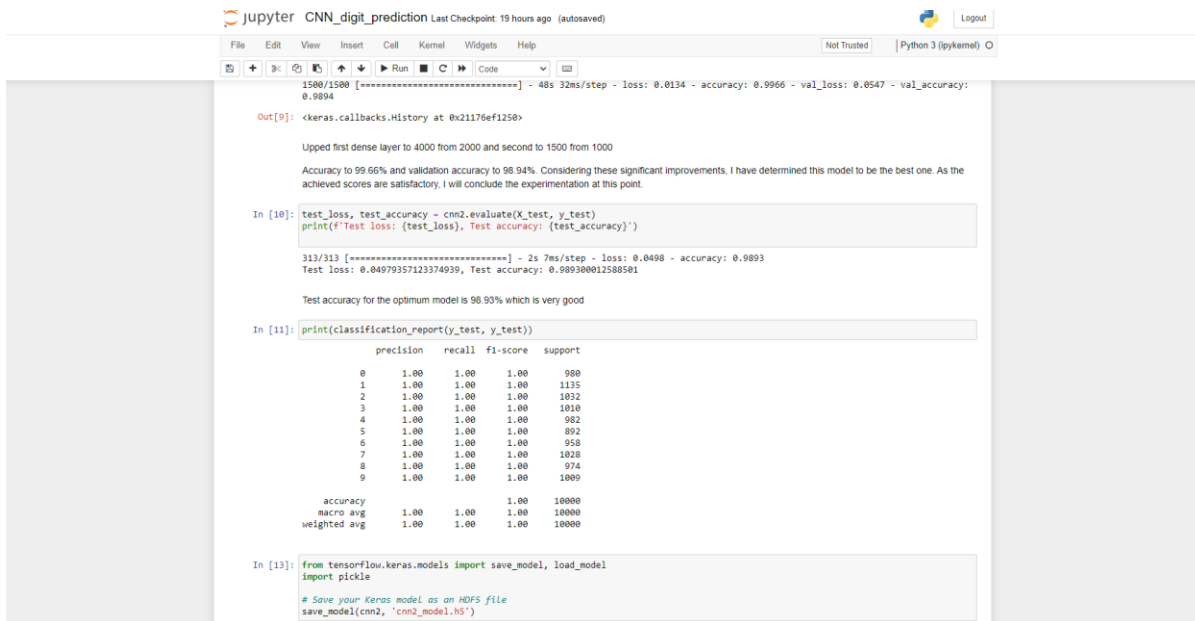
Data intake by:<Connor Walker>

Data intake reviewer:<intern who reviewed the report>

Data storage location: <github>

[Github link](#)  
[Website](#)

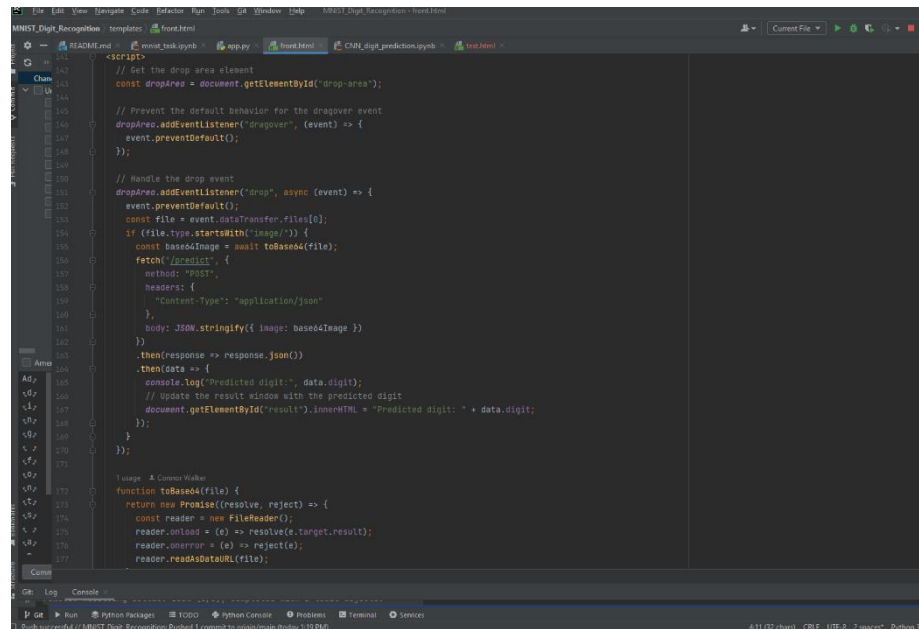
## Model deployment



The screenshot shows a Jupyter Notebook titled 'CNN\_digit\_prediction' with the following content:

```
jupyter CNN_digit_prediction Last Checkpoint: 19 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)
1500/1500 [=====] - 48s 32ms/step - loss: 0.0134 - accuracy: 0.9966 - val_loss: 0.0547 - val_accuracy: 0.9894
Out[9]: <keras.callbacks.History at 0x21176ef1250>
Upsped first dense layer to 4000 from 2000 and second to 1500 from 1000
Accuracy to 99.66% and validation accuracy to 98.94%. Considering these significant improvements, I have determined this model to be the best one. As the achieved scores are satisfactory, I will conclude the experimentation at this point.
In [10]: test_loss, test_accuracy = cm2.evaluate(X_test, y_test)
         print(f'Test loss: {test_loss}, Test accuracy: {test_accuracy}')
313/313 [=====] - 2s 7ms/step - loss: 0.0498 - accuracy: 0.9893
Test loss: 0.04979357123374939, Test accuracy: 0.989300012588581
Test accuracy for the optimum model is 98.93% which is very good
In [11]: print(classification_report(y_test, y_test))
          precision    recall  f1-score   support
0         1.00         1.00         1.00         980
1         1.00         1.00         1.00        1135
2         1.00         1.00         1.00        1032
3         1.00         1.00         1.00        1010
4         1.00         1.00         1.00         902
5         1.00         1.00         1.00         892
6         1.00         1.00         1.00         950
7         1.00         1.00         1.00        1028
8         1.00         1.00         1.00         974
9         1.00         1.00         1.00        1009
 accuracy          1.00         1.00        10000
macro avg          1.00         1.00         1.00        10000
weighted avg        1.00         1.00         1.00        10000
In [13]: from tensorflow.keras.models import save_model, load_model
         import pickle
         # Save your Keras model as an HDF5 file
         save_model(cm2, 'cm2_model.h5')
```

## Front End



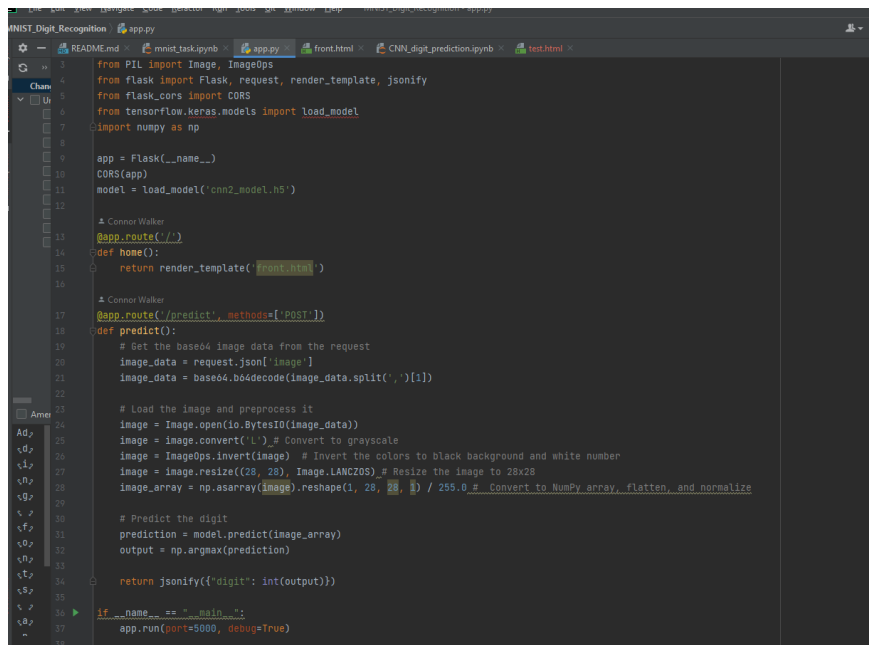
```
<script>
// Set the drop area element
const dropArea = document.getElementById("drop-area");

// Prevent the default behavior for the dragover event
dropArea.addEventListener("dragover", (event) => {
  event.preventDefault();
});

// Handle the drop event
dropArea.addEventListener("drop", async (event) => {
  event.preventDefault();
  const file = event.dataTransfer.files[0];
  if (file.type.startsWith("image/")) {
    const base64Image = await toBase64(file);
    fetch("/predict", {
      method: "POST",
      headers: {
        "Content-Type": "application/json"
      },
      body: JSON.stringify({ image: base64Image })
    })
      .then(response => response.json())
      .then(data => {
        console.log("Predicted digit:", data.digit);
        // Update the result window with the predicted digit
        document.getElementById("result").innerHTML = "Predicted digit: " + data.digit;
      });
  }
});

// Usage: A Connor Walker
function toBase64(file) {
  return new Promise((resolve, reject) => {
    const reader = new FileReader();
    reader.onload = (e) => resolve(target.result);
    reader.onerror = (e) => reject(e);
    reader.readAsDataURL(file);
  });
}
```

## Flask Web app



```
from PIL import Image, ImageOps
from flask import Flask, request, render_template, jsonify
from flask_cors import CORS
from tensorflow.keras.models import load_model
import numpy as np

app = Flask(__name__)
CORS(app)
model = load_model('cnn2_model.h5')

# Connor Walker
@app.route("/")
def home():
    return render_template("front.html")

# Connor Walker
@app.route("/predict", methods=['POST'])
def predict():
    # Get the base64 image data from the request
    image_data = request.json['image']
    image_data = base64.b64decode(image_data.split(',')[1])

    # Load the image and preprocess it
    image = Image.open(io.BytesIO(image_data))
    image = image.convert('L') # Convert to grayscale
    image = ImageOps.invert(image) # Invert the colors to black background and white number
    image = image.resize((28, 28), Image.LANCZOS) # Resize the image to 28x28
    image_array = np.asarray(image).reshape(1, 28, 28, 3) / 255.0 # Convert to NumPy array, flatten, and normalize

    # Predict the digit
    prediction = model.predict(image_array)
    output = np.argmax(prediction)

    return jsonify({"digit": int(output)})

if __name__ == "__main__":
    app.run(port=5000, debug=True)
```

## Website

