Jacob & Laksh

ISAT B409

12/8

Final Project Paper - Number Recognition Game

1. Introduction

The goal of this project is to develop an interactive game app where a player competes against the computer in a two-stage game, focusing on number recognition and machine learning. The app uses neural networks to make it interesting, showing how AI can predict and work with real data. The game is structured to test both the player’s ability to recognize numbers and the computer’s capability to predict them accurately.

The game consists of two stages, each with 10 turns. In Stage 1, the user is prompted to enter 10 numbers by drawing them on the screen. The app uses the FingerPaint library to capture these drawings as images, which are processed by converting them into pixel data for the neural network. The computer then attempts to predict the drawn numbers based on its prior training, comparing the predictions with the user’s input. In Stage 2, the app randomly selects 10 numbers from the load\_digits dataset from sklearn, displaying them to the user. The user must then guess these digits, earning one point for each correct guess.

At the end of both stages, the results are compared in determining who performed better, the computer or the user. This comparison serves as the final game outcome and demonstrates the efficiency of the neural network model in real-time number prediction and recognition. Future enhancements could involve improving the training of the model to make the computer's number predictions more efficient in the first stage of the game. This can be achieved by refining the dataset and mechanics based on user feedback. The app currently uses machine learning, specifically a neural network trained on the MNIST dataset, to accurately recognize and predict digits.

1. Training of Model

The neural network for the project was trained on the MNIST dataset, which is a large database of handwritten digits commonly used for training various image processing systems. This dataset contains 60,000 training images and 10,000 testing images, each normalized to fit a 28x28 pixel grid. The MNIST dataset is a standard benchmark in machine learning and computer vision, widely used for demonstrating and testing algorithms.

To prepare the dataset, the images were preprocessed by scaling pixel values to a range of 0 to 1 for improved optimization. The neural network consists of an input layer that processes the 64-pixel grayscale data, hidden dense layers with ReLU activation for identifying patterns, and a softmax output layer to predict the digits. To enhance generalization and prevent overfitting, a dropout layer was incorporated during training. The model was trained with categorical cross entropy loss, then converted to TensorFlow Lite for efficient deployment on mobile devices. This helps make sure there is accurate digit recognition and smooth user experience in the app's second stage.

We directly took the MNIST.ipynb file given to us on Project directory on blackboard for course ISAT B409.

1. Stage 1 – Computer Prediction Stage

In Stage 1, the user inputs handwritten digits using the FingerPaintView component. This input is captured and converted into a 28x28 pixel bitmap, which is then processed into a ByteBuffer using the convertBitmapToByteBuffer method. This buffer is fed into the runInference method, where the trained TensorFlow Lite model, loaded from the digit.tflite file, performs inference to predict the digit. Specifically, the model evaluates the input image, producing an array of probabilities for each digit, 0-9. The getPredictedDigit function then selects the digit corresponding to the highest probability, which is compared against the user’s input. The app then compares the predicted digit with the digit the user drew (using the input in textbox). If they match, the computer’s correct guesses count is incremented. The updateGameMetrics function tracks the game progress, ensuring the round and metrics are updated accordingly. The game proceeds to Stage 2 once all 10 rounds are completed.

Some important functions to explain -

1. *private ByteBuffer convertBitmapToByteBuffer(Bitmap bitmap)*

This code converts a Bitmap image into a ByteBuffer formatted for neural network input:

* Allocate Buffer: Allocates a direct byte buffer of size 4 \* 28 \* 28.
* Set Byte Order: Sets the byte order to the native order of the hardware.
* Get Pixel Data: Extracts pixel values from the bitmap into an integer array.
* Convert to Grayscale: Converts RGB values to grayscale.
* Normalize Pixels: Normalizes the grayscale values to a range of 0 to 1 and puts them into the byte buffer.

The result is a normalized ByteBuffer ready for neural network processing.

1. *private float[] runInference(ByteBuffer inputBuffer)*

* Initialize Output: Creates a 2D float array output with dimensions [1][10] to store the inference results.
* Run Inference: Uses the TensorFlow Lite model (tflite) to run inference on the inputBuffer, storing the results in the output array.
* Return Results: Returns the first element of the output array, which contains the inference results.

This function takes a ByteBuffer as input, runs the TensorFlow Lite model on it, and returns the results as a float array having the probabilities for all the digits.

1. *private MappedByteBuffer loadModelFile() throws IOException*

* Open Asset File: Opens the asset file descriptor for the "digit.tflite" model file using getAssets().openFd("digit.tflite").
* Create InputStream: Creates a FileInputStream from the file descriptor obtained.
* Get FileChannel: Retrieves the FileChannel from the input stream.
* Get Start Offset: Gets the start offset of the file from the file descriptor.
* Get Declared Length: Gets the declared length of the file from the file descriptor.
* Map File to Memory: Maps the file into memory as a read-only MappedByteBuffer using the file channel, start offset, and declared length.
* Return Buffer: Returns the mapped byte buffer containing the model file.

This function loads the TensorFlow Lite model file "digit.tflite" into a MappedByteBuffer for use in inference.

1. Stage 2 – User Prediction Stage

In Stage 2, the app presents 10 random digits from the load\_digits dataset. These digits are displayed one at a time, and the user is asked to guess each one. The digits are preprocessed and rendered as 8x8 grayscale images to maintain a consistent format. The user enters their guess into an EditText field, and upon submission, the app uses the checkUserGuess function to compare the user’s input with the correct digit. The value of correct digit is stored in target array present in assets directory. If the guess is correct, the user's score is incremented using the updateScore function, and feedback is provided for each guess. The checkUserGuess function performs a simple comparison between the entered guess and the actual digit, returning a boolean result. If correct, the score is updated by 1 point using the updateScore function.

After the user completes the rounds, the app compares the user’s score to the computer’s performance from Stage 1, where the computer predicted numbers using its neural network. The results are shown, determining the winner and demonstrating the accuracy of the neural network in real-time number recognition.

Some important functions to explain -

1. *private void showNextDigit()*

* Generate Random Index: Generates a random index between 0 and 1796.
* Log Index: Logs the current digit index for debugging purposes.
* Create Image Name: Constructs the image name using the format "digit\_<index>".
* Get Resource ID: Retrieves the resource ID of the drawable image corresponding to the generated index.
* Set Image: Sets the digit image view to display the image with the retrieved resource ID.
* Clear Guess Field: Clears the text in the user guess input field.
* Clear Result Field: Clears the text in the guess result view.

This function updates the displayed digit image with a randomly selected digit, clears the user input and result display fields, and logs the current digit index.

1. *private int[] loadTargetArray(InputStream inputStream) throws IOException*

* Create BufferedReader: Initializes a BufferedReader to read from the input stream.
* Initialize Array: Creates an integer array of size 1797 to store the target values.
* Initialize Index: Sets the starting index to 0.
* Read Lines: Reads lines from the input stream in a loop.
* Parse Integer: Converts each line to an integer and stores it in the array at the current index.
* Increment Index: Increments the index after storing each integer.
* Close Reader: Closes the BufferedReader.
* Return Array: Returns the filled integer array.

This function reads integer values from an input stream, stores them in an array, and returns the array. This is basically used to read the target.txt file that stores all the labels for the images displayed in the 2nd stage of the game.

1. Conclusion

This project has been an excellent opportunity to develop both technical and teamwork skills while creating an interactive game app. One of the primary goals we achieved was learning to manage our project efficiently, ensuring smooth collaboration and progress. Additionally, having a foundation and game plan set early on who will do what, went a long way in managing our time and efficiency. We are satisfied with our work, but we could have gone about it slightly differently. For example, we could have used GitHub to improve collaboration on the coding. We struggled to find a time or place to meet, so we relied on a few phone calls to discuss our progress. In hindsight, we should have planned better by setting up dedicated meeting times earlier to streamline communication and ensure smoother collaboration.

In terms of responsibilities, Jacob was responsible for writing the project paper, creating the presentation materials, and providing feedback on the coding. Meanwhile, Laksh focused on implementing the game project in IntelliJ IDEA and offered feedback on the paper. Laksh also contributed the abstract for the project and originated the idea for the game. This clear division of responsibilities ensured that both of us could focus on our strengths while effectively supporting each other.

This has been a valuable learning experience, particularly in understanding how neural networks can be used effectively in our number recognition game. The process of combining technical skills with creative thinking has been both challenging and rewarding, showing the potential of teamwork and the versatility of machine learning in software development.

References

*API documentation*. TensorFlow. (n.d.). https://www.tensorflow.org/api\_docs

Florian. (2019, September 29). *Handwritten digits recognition with tensorflow*. NeuralNine. https://www.neuralnine.com/handwritten-digits-recognition-with-tensorflow/

Ibm. (n.d.). *What is machine learning?*. IBM. https://www.ibm.com/topics/machine-learning

Page, J. (2020, January 17). *Handritten Digit Recognition Using Tensorflow Lite on Android*. GitHub. https://github.com/mrinalTheCoder/HandwrittenDigitsRecognition

*Sklearn.datasets.load\_digits*. scikit. (n.d.). https://scikit-learn.org/1.0/modules/generated/sklearn.datasets.load\_digits.html

*What is MNIST dataset?*. Engati. (n.d.). https://www.engati.com/glossary/mnist-dataset