Institute of Technology of Cambodia

Department of AMS

I3-AMS-C

Khmer Handwritten Digits Recognition

Team: 6

	Name	ID
1.	SOEUK Bondol	e20221592
2.	SOU Pichchomrong	e20210874
3.	SEING Ratana	e20221147
4.	SROY Liza	e20220912
5.	TIENG Sopanha	e20220547

Under Supervision of Dr. HAS Sothea

Academic year 2024-2025

Table of Content

ABSTRACT

The concept of handwritten recognition using machine learning is still relatively new, considering the fact that the first paper ever released on handwritten digit recognition emerged in 1994 by Yann LeCun et al. Nowadays, many researchers publish papers related to handwritten digit recognition using various methodologies and mostly perform experiments on the MNIST dataset (Arabic numerals). However, only a few papers have been published on Khmer numeral recognition.

Among the papers related to handwritten digit recognition performed on the MNIST dataset, the CNN (Convolutional Neural Network) methodology has become the state-of-the-art approach for building models that can recognize Arabic numerals from 0 to 9, achieving around 95-99% accuracy.

In this paper, we will use our own Khmer handwritten dataset, consisting of 3,000 images ranging from 0 to 9, and feed it into CNN models that have been built by other researchers to evaluate their performance. We will measure their accuracy and further use the to build a website that allows user interaction, potentially enabling it to perform arithmetic operations based on user drawings.

This paper was created with the purpose of promoting the use of Khmer numerals and connecting modern technology with our cultural traditions. We hope it will inspire others to appreciate Khmer numerals and to explore ways of integrating modern technology into the Khmer language. Such efforts could lead to the development of tools that benefit the Cambodian people, such as a Khmer AI chatbot, and more.

I. Introduction

Humans can easily recognize things because of light entering our eyes, which is then converted into electrical signals that travel to the brain via the optic nerve. The brain interprets these signals, creating the images we see and recognizing them based on prior experience and memory.

However, for a machine to recognize things such as images, we need to convert those images into numbers for example, representing the brightness of each pixel as a value between 0 and 255. These numerical representations are then fed into a Convolutional Neural Network (CNN), which consists of multiple layers responsible for enabling the model to learn patterns present in each image and determine which class they belong to.

In our case, we will first resize our images to 28×28 pixels and convert them to grayscale to reduce computational complexity while still preserving important features for extraction. Next, we will shuffle our dataset and convert each image into a 28×28 array of pixel values before passing them through the CNN.

The output of the CNN will be a probability distribution across each class, where the total combined probability equals 1 due to the use of the Softmax activation function at the output layer. This output layer consists of 10 nodes, each corresponding to a class from 0 to 9. The node with the highest probability will represent the predicted label based on the input features.

II. Literature Review

There are many researchers who have published their work on number recognition using the MNIST dataset, but only a few have focused on Khmer text or Khmer number recognition.

In the paper "*Handwritten Khmer Text Recognition*" published by Bayram Annanurov and Norliza Noor (2016), the authors employ Convolutional Neural Networks (CNNs) to recognize Khmer handwritten symbols. Each CNN is trained to recognize a specific consonant, resulting in 33 networks combined into an assembly. The performance is compared against Artificial Neural Network (ANN)-based classifiers using full feature sets and dimensionality reduction techniques such as two-dimensional Fourier transformation (FT2D) and Gabor filters. Their CNN model achieves an accuracy of 94.85%.

According to another paper published by Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998), titled "*Gradient-Based Learning Applied to Document Recognition*", they proposed and implemented LeNet-5, a Convolutional Neural Network (CNN) for document recognition. Using gradient-based learning (backpropagation) with minimal preprocessing, they achieved an accuracy of 99.2% on the MNIST dataset.

In another research paper published by Mendapara Shubham, Pabani Krish, and Paneliya Yash (2021), titled "*Handwritten Digit Recognition System*", the authors aim to develop a system that accurately recognizes handwritten digits using a CNN trained on the MNIST dataset. The paper details every step of building the model, which achieves an accuracy of 98.85%.

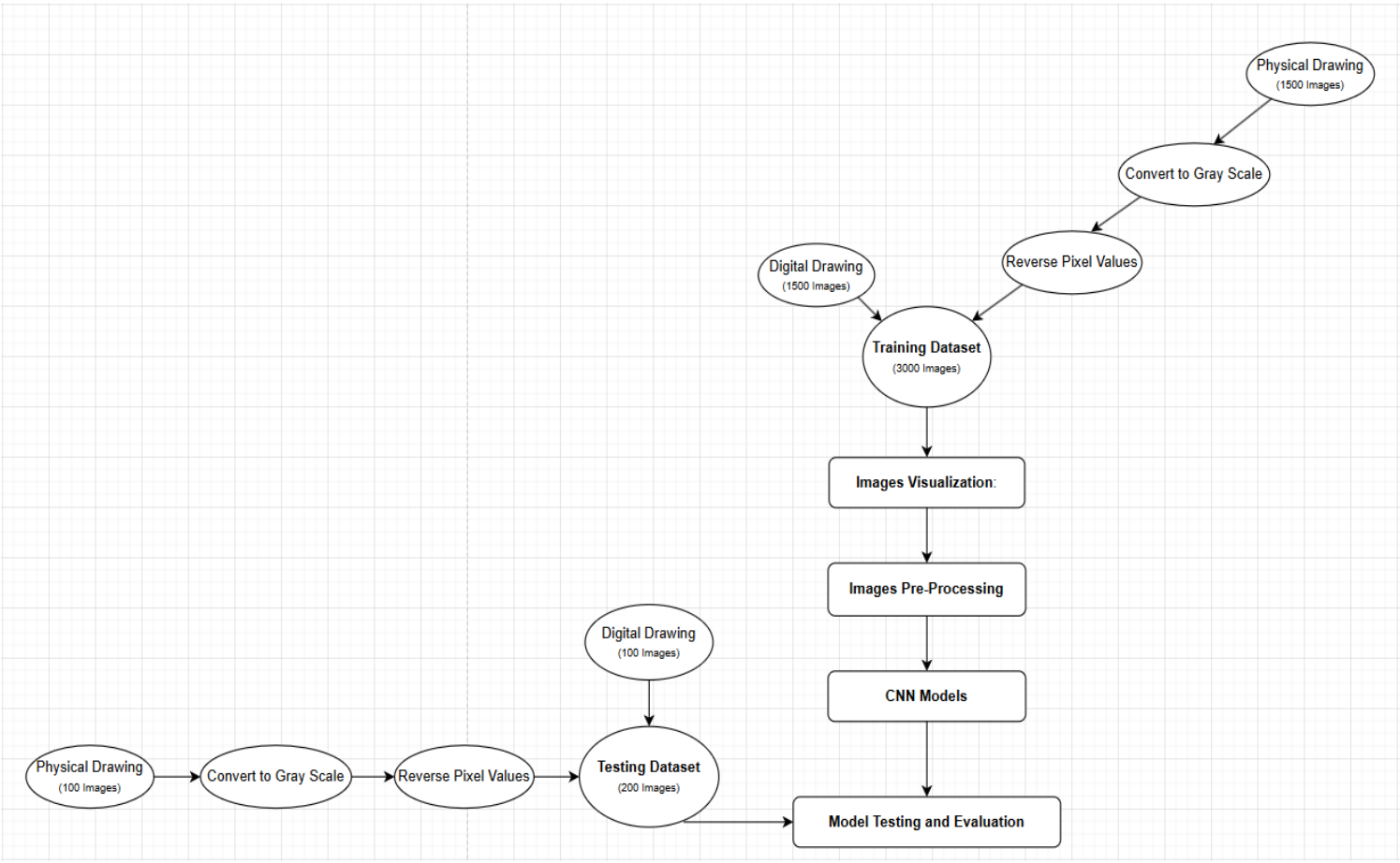
Additionally, a paper published by Norhidayu Abdul Hamid and Nilam Nur Amir Sjarif (2017), titled "*Handwritten Recognition Using SVM, KNN and Neural Network*", evaluates and compares the performance of three classification algorithms—Support Vector Machine (SVM), K-Nearest Neighbour (KNN), and Multi-layer Perceptron (MLP) Neural Network—for handwritten digit recognition using the MNIST dataset. Their model achieves an accuracy of 99.26% using KNN.

Lastly, the paper published by Kottakota Asish, P. Sarath Teja, R. Kishan Chander, and Dr. D. Deva Hema (2023), titled "*NeuroWrite: Predictive Handwritten Digit Classification using Deep Neural Networks*", presents a deep learning model ("NeuroWrite") that accurately classifies handwritten digits using CNNs. The model primarily focuses on the MNIST dataset and is also implemented for real-time prediction using a camera. Their model achieves an accuracy of 99.21% on the MNIST dataset.

III. Methodology

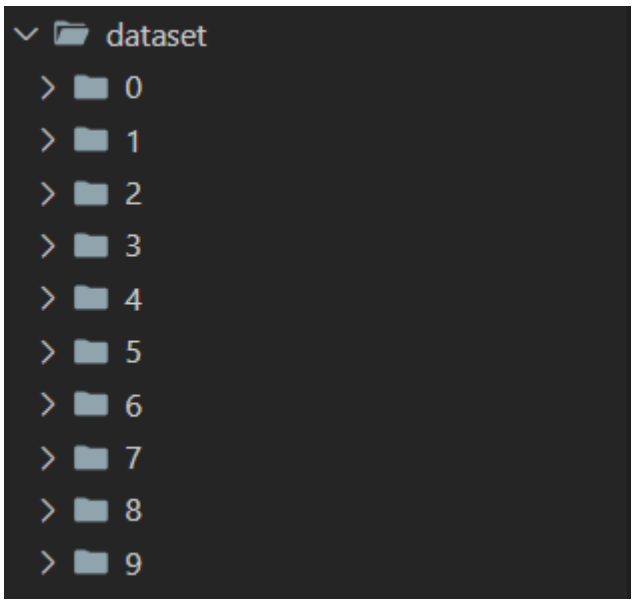
1. Flowchart

The process of building a model that can recognize Khmer numerals can be summarize as follow:



2. Data Loading

First, we created two types of datasets: one by drawing on a digital canvas and the other by drawing on a white blank sheet of paper. Each dataset consists of 1,500 images. All of the images have been organized into their corresponding classes, ranging from 0 to 9.



For the dataset created by drawing on a white sheet of paper, we first convert the images to grayscale and then invert their pixel values before combining the two datasets. This is because, after converting to grayscale, the background becomes white-ish and the digits appear gray-ish. By inverting the pixel values, the background becomes gray-ish and the digits become more white-ish, which makes them visually similar to the digital canvas dataset. We do this to avoid any confusion during model training.

For the dataset created using the digital canvas, we do not make any changes.

After this, we began to access our dataset file and start to explore.

```python
dataset_path = "dataset"
```

3. Data Visualization

In this process, we check the number of classes in our dataset and the number of images in each corresponding class. This is done to ensure there are no mistakes or errors during dataset loading.

```python
classes = sorted(os.listdir(dataset_path))
print(f"Number of classes: {len(classes)}")
print(f"Class names: {classes}")
```

Python

```
Number of classes: 10
Class names: ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

```python
total_images = 0
class_count = []

for cls in classes:
    class_path = os.path.join(dataset_path, cls)
    count = len(os.listdir(class_path))
    class_count.append(count)
    total_images += count
    print(f"Class {cls}: {count} images")

print(f"Total images in dataset: {total_images}")
```

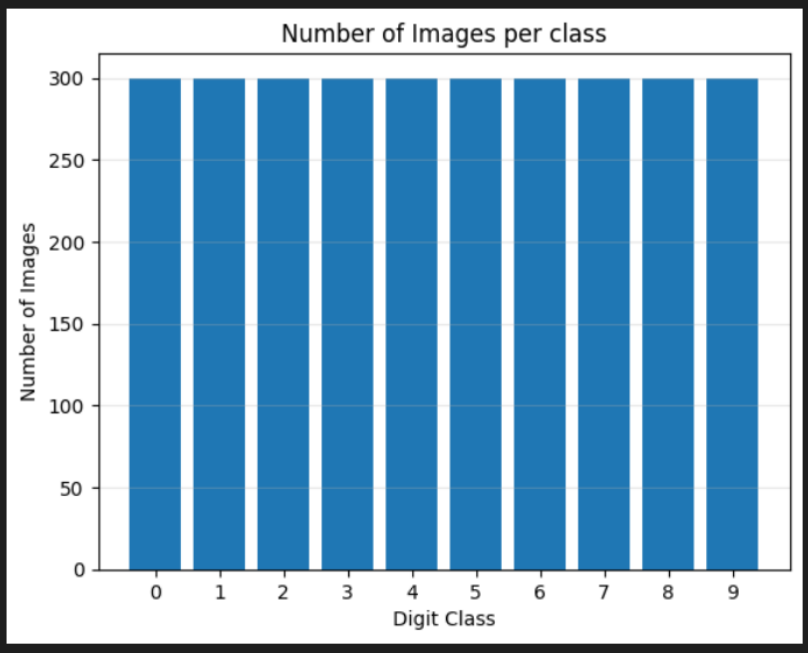
✓ 0.0s Python

```
Class 0: 300 images
Class 1: 300 images
Class 2: 300 images
Class 3: 300 images
Class 4: 300 images
Class 5: 300 images
Class 6: 300 images
Class 7: 300 images
Class 8: 300 images
Class 9: 300 images
Total images in dataset: 3000
```

After this, we plot a bar graph to show the images distribution.

```
plt.Figure(figsize=(10, 5))
plt.bar(classes, class_count)
plt.title("Number of Images per class")
plt.xlabel("Digit Class")
plt.ylabel("Number of Images")
plt.xticks(classes)
plt.grid(axis='y', alpha=0.3)
plt.show()
```

✓ 0.1s Python



After that, we check the size and pixels values of all the images in class 0.

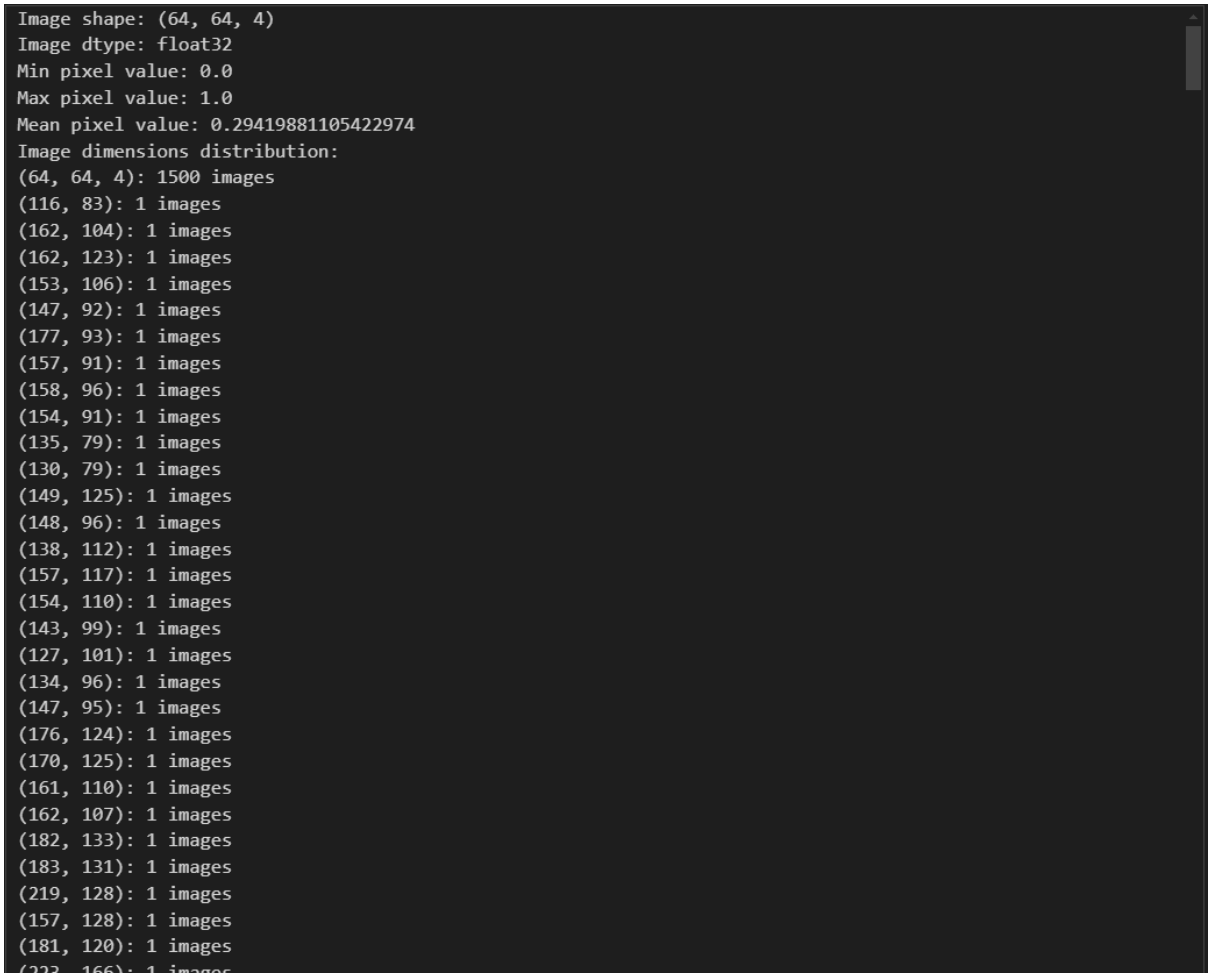
```
sample_class = classes[0]
sample_path = os.path.join(dataset_path, sample_class)
sample_img = os.path.join(sample_path, os.listdir(sample_path)[0])
img = plt.imread(sample_img)

print(f"Image shape: {img.shape}")
print(f"Image dtype: {img.dtype}")
print(f"Min pixel value: {img.min()}")
print(f"Max pixel value: {img.max()}")
print(f"Mean pixel value: {img.mean()}")

# Check if all images have the same dimensions
dimensions = {}
for cls in classes:
    class_path = os.path.join(dataset_path, cls)
    for img_file in os.listdir(class_path):
        img_path = os.path.join(class_path, img_file)
        img = plt.imread(img_path)
        dim = img.shape
        dimensions[dim] = dimensions.get(dim, 0) + 1

print("Image dimensions distribution:")
for dim, count in dimensions.items():
    print(f"{dim}: {count} images")
```

✓ 1.6s Python



We can see all the images size are not the same in class 0. This also implies to other classes too. Those 1500 images that have the shape of (64, 64, 4) are the images that we created using digital canvas. The other irregular shape images are the images that we've created by drawing on the blank sheet of paper.

4. Data Pre-Processing

In this step, we first load the images in their original size and color. Then, we convert them to grayscale. After that, we resize the images to a shape of 28×28 pixels. All of these processed images are then added to our training data along with their corresponding class labels.

```python
from tqdm import tqdm

training_data = []
IMG_SIZE = 28

def create_training_data():
    for category in classes:

        path = os.path.join(dataset_path,category)
        class_num = classes.index(category)

        for img in tqdm(os.listdir(path)):
            try:
                img_path = os.path.join(path, img)
                img_array = cv2.imread(img_path, cv2.IMREAD_COLOR)
                img_array = cv2.cvtColor(img_array, cv2.COLOR_BGR2GRAY)
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                training_data.append([new_array, class_num])
            except Exception as e:
                pass

create_training_data()

print(len(training_data))
```

✓ 1.3s Python

```
100%|██████████| 300/300 [00:00<00:00, 2878.23it/s]
100%|██████████| 300/300 [00:00<00:00, 2999.19it/s]
100%|██████████| 300/300 [00:00<00:00, 2013.54it/s]
100%|██████████| 300/300 [00:00<00:00, 2702.63it/s]
100%|██████████| 300/300 [00:00<00:00, 1898.36it/s]
100%|██████████| 300/300 [00:00<00:00, 2325.66it/s]
100%|██████████| 300/300 [00:00<00:00, 2263.35it/s]
100%|██████████| 300/300 [00:00<00:00, 2307.16it/s]
100%|██████████| 300/300 [00:00<00:00, 2098.13it/s]
100%|██████████| 300/300 [00:00<00:00, 2205.93it/s]
3000
```

Next step is to shuffle the training data to avoid any error during training.

```python
import random
random.shuffle(training_data)
```

Python

Then, we let a variable X = features, variable y = labels. We do this because we want to further normalize our pixels values between 0-1.

```
X = []
y = []

for features,label in training_data:
    X.append(features)
    y.append(label)
```

✓ 0.0s Python

Next step is to convert the features into a numpy array with the shape of (28, 28, 1) then normalize and we also need to one hot encoding our labels.

```
X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
X = X / 255.0
X[100]
```

✓ 0.0s Python

```
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ],
       [0.        ]
```

```
y = to_categorical(y, num_classes=10)
y
```

✓ 0.0s Python

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

5. Model Training

In this step, we feed features and labels into CNN model.

```python
from tensorflow.keras.models import Sequential # type: ignore
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, LeakyReLU # type: ignore

# Define the CNN model
model = Sequential()

# First Convolutional Layer
model.add(Conv2D(filters=8, kernel_size=(5, 5), strides=(1, 1), padding='same', input_shape=(28, 28, 1)))
model.add(LeakyReLU(alpha=0.1))

# First Max Pooling Layer
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Second Convolutional Layer
model.add(Conv2D(filters=16, kernel_size=(5, 5), strides=(1, 1), padding='same'))
model.add(LeakyReLU(alpha=0.1))

# Second Max Pooling Layer
model.add(MaxPooling2D(pool_size=(5, 5), strides=(2, 2)))

# Flattening Layer
model.add(Flatten())

# Fully Connected Output Layer
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X, y, epochs=10, validation_split=0.2)

model.summary()
```

✓ 4.5s Python

```
Epoch 1/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.2190 - loss: 2.2139 - val_accuracy: 0.5550 - val_loss:
1.4656
Epoch 2/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6817 - loss: 1.1485 - val_accuracy: 0.8167 - val_loss:
0.6463
Epoch 3/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8067 - loss: 0.6167 - val_accuracy: 0.8783 - val_loss:
0.4424
Epoch 4/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8794 - loss: 0.4182 - val_accuracy: 0.9133 - val_loss:
0.3154
Epoch 5/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9047 - loss: 0.3233 - val_accuracy: 0.9317 - val_loss:
0.2569
Epoch 6/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9325 - loss: 0.2666 - val_accuracy: 0.9517 - val_loss:
0.1962
Epoch 7/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9578 - loss: 0.1736 - val_accuracy: 0.9300 - val_loss:
0.2283
Epoch 8/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9554 - loss: 0.1658 - val_accuracy: 0.9567 - val_loss:
0.1656
Epoch 9/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9554 - loss: 0.1481 - val_accuracy: 0.9600 - val_loss:
0.1348
Epoch 10/10
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9761 - loss: 0.1015 - val_accuracy: 0.9683 - val_loss:
0.1026
```

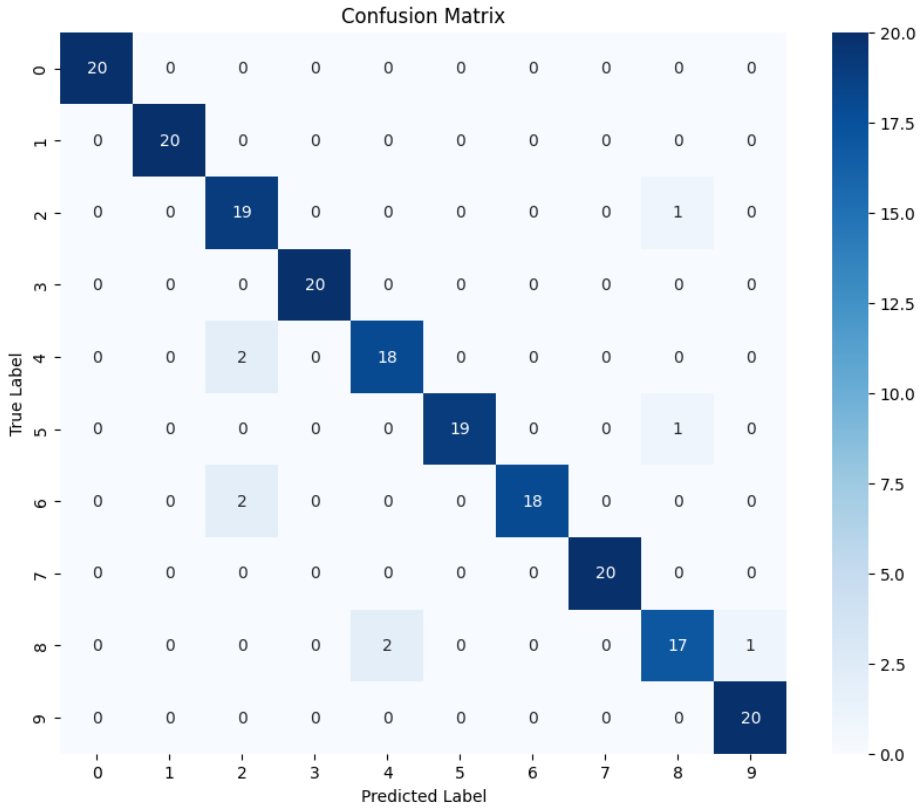
After training for 10 epochs, the accuracy during final training reach 97.61%. After this, we save our trained model as Keras format for future testing and evaluation without the need to train the model again.

```
model.save("digit_model.keras")
```
✓ 0.0s Python

6. Model Testing and Evaluation

In this part we create another testing dataset consist of 200 images (20 images in each class) where 100 of the images are created from drawing on a digital canvas and another 100 images created from drawing on a blank sheet of paper. For the dataset that we've created by drawing on a blank sheet of paper we will need to pre-process it first such as convert to grayscale and invert their pixels values before adding in to the testing dataset. From 200 images in testing dataset, we got the result as follow:

Class	Precision	Recall	F1-Score	Accuracy
0	100%	100%	100%	95%
1	100%	100%	100%	
2	83%	95%	88%	
3	100%	100%	100%	
4	90%	90%	90%	
5	100%	95%	97%	
6	100%	90%	95%	
7	100%	100%	100%	
8	89%	85%	87%	
9	95%	100%	98%	



IV. Conclusion and Future Work

In conclusion we can see the CNN model that we use on our created dataset perform very well and make such good prediction with minimal error. This result suggests with even more datasets; this model can perform even better. There are some room for improvement that we need to consider such as:

- Make the dataset more diverse and bigger
- Hyper parameter tuning

For future works, we plan to develop a website or an app where user can interact with. Furthermore, we want to create a kid-friendly app design specifically for kid to play with and also do some math like addition, subtraction, multiplication, division. We do this because we want to encourage them to learn Khmer numerals and have some basic understanding in math.

Reference:

- Annanurov, B., & Noor, N. M. (2016). Handwritten Khmer text recognition. In 2016 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE) (pp. 176-179). IEEE.

- Annanurov, B., & Noor, N. M. (2021). A compact deep learning model for Khmer handwritten text recognition. IAES International Journal of Artificial Intelligence (IJ-AI), 10(3), 584-591.

- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

- International Journal of Scientific Research in Computer Science, Engineering and Information Technology, Volume 7, Issue 5, September-October 2021, Pages 76-85.

- Norhidayu Abdul Hamid, Nilam Nur Amir Sjarif, "Handwritten Recognition Using SVM, KNN and Neural Network," arXiv preprint arXiv:1702.00723, 2017.

- Kottakota Asish, P. Sarath Teja, R. Kishan Chander, and Dr. D. Deva Hema, "NeuroWrite: Predictive Handwritten Digit Classification using Deep Neural Networks", SRM Institute of Science and Technology, Ramapuram, Chennai, India.

- Cireșan, D., Meier, U., & Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. Computer Vision and Pattern Recognition (CVPR), 3642-3649.

- Dr.Sreedath Panat, 2024 December 27, "*Neural Network From Scratch: No Pytorch & Tensorflow; just pure math | 30 min theory + 30 min coding*" https://youtu.be/A83BbHFoKb8?si=NoaEYn5Nzl9i3j1q

- Brandon Rohrer, 2017 March 2, "*How Deep Neural Networks Work*" https://youtu.be/ILsA4nyG7I0?si=lnNR6E8zw5QGwFtE

- Lix Fridman, 2017 January 26, "*MIT 6.S094: Convolutional Neural Networks for End-to-End Learning of the Driving Task*" https://youtu.be/U1toUkZw6VI?si=1V_Il0_mgV2hzlIn

- Grant Sanderson, 2017 October 5, "*But what is a neural network? | Deep learning chapter 1*" https://youtu.be/aircAruvnKk?si=ZdFoPwu7_O5O1Xdh

- Sentdex, 2018, "*Deep Learning basics with Python, TensorFlow and Keras*" https://youtube.com/playlist?list=PLQVvvaa0QuDfhTox0AjmQ6tvTgMBZBEXN&si=LuNxstXqc--9Jq15

- Mengsay Loem, 2020 June 20, "*Handwritten Khmer Digit Recognition*" https://medium.com/data-science/handwritten-khmer-digit-recognition-860edf06cd57

- Nicholas Renotte, 2022 April 25, "Build a Deep CNN Image Classifier with ANY Images" https://youtu.be/jztwpsIzEGc?si=48s5zBX4Ms5g1yNl