Doodling:

Handwritten Digit Recognition Project

Machine Learning (UML501)

Project

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Submitted To:

Dr. Archana Singh

Submitted By:

Name: Smriti

Roll No: 102116094

Subgroup: CS11

Overview

The "Doodling" project is a machine learning (ML) application designed to recognize handwritten digits provided by users. The system takes user-drawn digits and employs a Convolutional Neural Network (CNN) for digit recognition along with the deep-learning algorithm which uses Keras and TensorFlow Open-Source Framework. The primary objective is to accurately classify handwritten digits from 0 to 9. This report summarizes the project, the model's architecture, training and evaluation, and includes a comparison chart of model performance. Some libraries that are used in building this project are:

Project Components

The "Doodling" project consists of the following components:

1. Data Loading and Preprocessing:

- The MNIST dataset is used, which contains 60,000 training images and 10,000 testing images.
- Data preprocessing includes normalizing images to the [0, 1] range and expanding image dimensions to (28, 28, 1).
- Class labels are converted to one-hot encoded vectors
- Data-Augmentation and Gaussian Blur Data Pre-Processing Techniques are also used to improve the accuracy

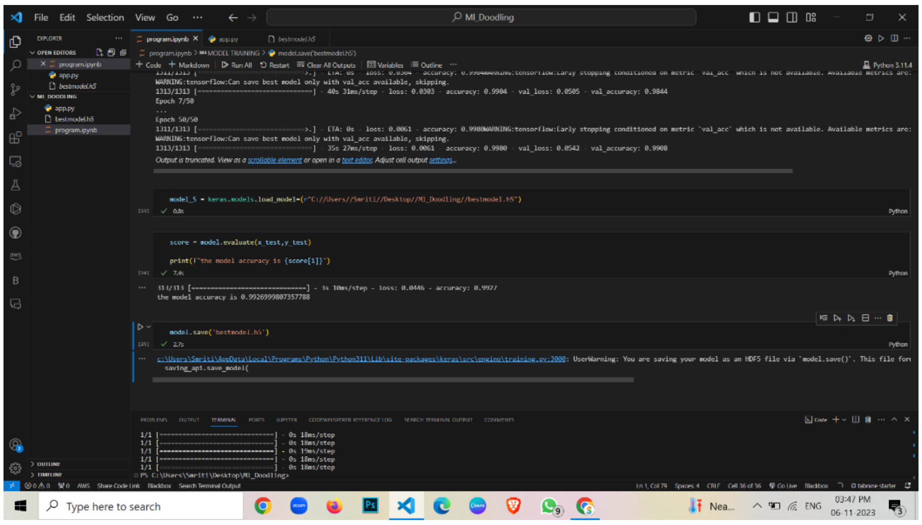
2. Model Architecture:

- A Convolutional Neural Network (CNN) is used for digit recognition.
- The model includes:
  - ➢ Two 2D Convolutional layers with activation.
  - ➢ Two Max-Pooling layers to reduce spatial dimensions.

- A Flatten layer to transform feature maps into a 1D vector.
- A Dropout layer to prevent overfitting.
- A Dense (fully connected) layer with SoftMax activation for digit classification

3. Model Compilation:

- The model is compiled with an optimizer (e.g., Adam) and a categorical cross-entropy loss function.
- Metrics for tracking include accuracy and accuracy after training the model is 99.26%



```
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1313/1313 [==============================] - 40s 31ms/step - loss: 0.0303 - accuracy: 0.9904 - val_loss: 0.0505 - val_accuracy: 0.9844
Epoch 7/50
...
Epoch 50/50
1311/1313 [============================>.] - ETA: 0s - loss: 0.0061 - accuracy: 0.9980WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are:
WARNING:tensorflow:Can save best model only with val_acc available, skipping.
1313/1313 [==============================] - 35s 27ms/step - loss: 0.0061 - accuracy: 0.9980 - val_loss: 0.0542 - val_accuracy: 0.9908
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```
model_5 = keras.models.load_model=(r"C://Users//Smriti//Desktop//Ml_Doodling//bestmodel.h5")
```

```
score = model.evaluate(x_test,y_test)

print(f"the model accuracy is {score[1]}")
```

```
313/313 [==============================] - 3s 10ms/step - loss: 0.0446 - accuracy: 0.9927
the model accuracy is 0.9926999807357788
```

```
model.save('bestmodel.h5')
```

```
c:\Users\Smriti\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\engine\training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file for
  saving_api.save_model(
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
PS C:\Users\Smriti\Desktop\Ml_Doodling>
```

4. Model Training:

- The model is trained using the training data with specified batch size and epochs.
- Callbacks like Early Stopping and Model Checkpoint can be utilized for enhanced training control.

5. Model Evaluation:

- The model's performance is evaluated using the testing data.
- Evaluation metrics, such as accuracy, are calculated.
- Confusion matrices and other relevant metrics can be generated for detailed analysis.

Project Code:

```
import NumPy as np
import matplotlib.pyplot as plt
import keras
import tensorflow as tf
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPool2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout


from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
## loading the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()


x_train.shape , y_train.shape ,x_test.shape ,y_test.shape


## splitting the dataset into training and testing
print('X_train: ' + str(x_train.shape))
print('y_train: ' + str(y_train.shape))
print('X_test:  '  + str(x_test.shape))
print('y_test:  '  + str(y_test.shape))


plt.imshow( x_train[0])


plt.imshow(x_train[0],cmap='binary')

### plotting the mnist dataset using matlab

def plot_input_img(i):
    plt.imshow(x_train[i],cmap='binary')
    plt.title(y_train[i])
    plt.axis('off')
    plt.show()

for i in range(5):
    plot_input_img(i)
```

```
DATA PRE-PROCESSING
-------------------

NORMALISATION:

#pre process the images
#normalizing the image to [0,1] range

x_train = x_train.astype(np.float32)/225
x_test = x_test.astype(np.float32)/225


----------------------------------------------------------------------------

RE-SHAPING  OF DATASET

#reshape or expand the dimentions of images to (28,28,1)


x_train = np.expand_dims(x_train,-1)
x_test = np.expand_dims(x_test, -1)


----------------------------------------------------------------------------

ONE-HOT ENCODING
#convert classes to one hot vectors

y_train = keras.utils.to_categorical(y_train)

y_test = keras.utils.to_categorical(y_test)

-----------------------------------------------------------------------------

DATA-AUGMENTATION

#Data augmentation helps create variations of your training data, making your
model more robust and less prone to overfitting.

datagen = ImageDataGenerator(
    rotation_range=10,          # Randomly rotate images up to 10 degrees
    width_shift_range=0.1,     # Randomly shift images horizontally
    height_shift_range=0.1,    # Randomly shift images vertically
    shear_range=0.2,           # Shear transformations
    zoom_range=0.1,            # Randomly zoom images
    horizontal_flip=False,     # Randomly flip images horizontally
    vertical_flip=False,       # Randomly flip images vertically
    fill_mode='nearest'        # Fill in newly created pixels with the nearest
existing pixel
```

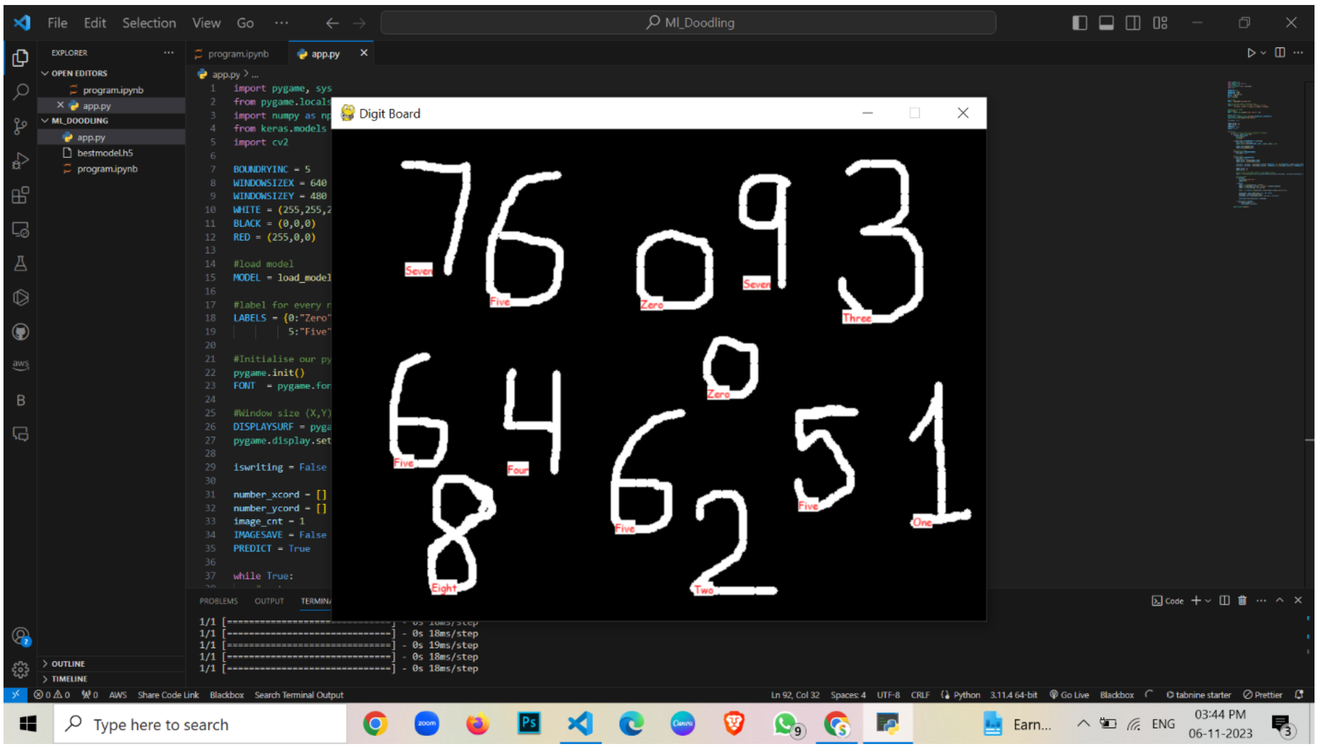
```python
# Example of applying data augmentation to one image
sample_image = x_train[0]
sample_label = y_train[0]
plt.figure(figsize=(8, 8))
plt.subplot(1, 4, 1)
plt.imshow(sample_image.reshape(28, 28), cmap='gray')
plt.title(f"Original - Label: {np.argmax(sample_label)}")

# Apply data augmentation
augmented_images = []
for i in range(3):
    augmented_image = datagen.random_transform(sample_image)
    augmented_images.append(augmented_image)
    plt.subplot(1, 4, i + 2)
    plt.imshow(augmented_image.reshape(28, 28), cmap='gray')
    plt.title(f"Augmented {i+1}")

plt.show()


----------------------------------------------------------------------------

GAUSSIAN BLUR

#Apply Gaussian blur to reduce noise and make the images smoother

from scipy.ndimage import gaussian_filter

# Example of applying Gaussian blur to one image
blurred_images = []
for i in range(3):
    augmented_image = datagen.random_transform(sample_image)
    blurred_image = gaussian_filter(augmented_image, sigma=1)
    blurred_images.append(blurred_image)
    plt.subplot(1, 4, i + 2)
    plt.imshow(blurred_image.reshape(28, 28), cmap='gray')
    plt.title(f"Blurred {i+1}")
```

```
USE OF CNN

model = Sequential()

model.add(Conv2D(32,(3,3), input_shape = (28,28,1), activation= 'relu'))
model.add(MaxPool2D((2,2)))


model.add(Conv2D(64,(3,3), activation= 'relu'))
model.add(MaxPool2D((2,2)))


model.add(Flatten())
model.add(Dropout(0.25))
model.add(Dense(10, activation='softmax'))


model.summary()

model.compile(optimizer='adam', loss = keras.losses.categorical_crossentropy,
metrics=['accuracy'])


MODEL TRAINING

# Earlystopping
es =
EarlyStopping(monitor='val_acc',min_delta=0.01,patience=4,verbose=1)


# Model check point
mc = ModelCheckpoint("./bestmodel.h5", monitor='val_acc', verbose=1,
save_best_only= True)
cb =[es,mc]

his =model.fit(x_train, y_train, epochs=50, validation_split=0.3,callbacks=cb)

model_S =
keras.models.load_model=(r"C://Users//Smriti//Desktop//Ml_Doodling//bestmodel.
h5")


score = model.evaluate(x_test,y_test)

print(f"the model accuracy is {score[1]}")


model.save('bestmodel.h5')
```

```
APP.PY (use of pygame) Code // Use of Deep learning

import pygame, sys
from pygame.locals import *
import numpy as np
from keras.models import load_model
import cv2

BOUNDRYINC = 5
WINDOWSIZEX = 640
WINDOWSIZEY = 480
WHITE = (255,255,255)
BLACK = (0,0,0)
RED = (255,0,0)

#load model
MODEL = load_model("bestmodel.h5")

#label for every number in dictionary form
LABELS = {0:"Zero", 1:"One", 2:"Two", 3:"Three", 4:"Four",
          5:"Five", 6:"Six", 7:"Seven", 8:"Eight", 9:"Nine"}

#Initialise our pygame
pygame.init()
FONT  = pygame.font.SysFont('Comic Sans Ms', 10)

#Window size (X,Y)
DISPLAYSURF = pygame.display.set_mode((WINDOWSIZEX, WINDOWSIZEY))
pygame.display.set_caption("Digit Board")

iswriting = False

number_xcord = []
number_ycord = []
image_cnt = 1
IMAGESAVE = False
PREDICT = True

while True:
    #capture any event from mouse or keyboard or anything
    for event in  pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

        if event.type == MOUSEMOTION and iswriting:
            xcord, ycord = event.pos
            pygame.draw.circle(DISPLAYSURF, WHITE, (xcord, ycord), 4, 0)
            number_xcord.append(xcord)
            number_ycord.append(ycord)

        #when start writing
        if event.type == MOUSEBUTTONDOWN:
            iswriting = True

        #after writing
        if event.type == MOUSEBUTTONUP:
            iswriting = False
            number_xcord = sorted(number_xcord)
            number_ycord = sorted(number_ycord)

            rect_min_x, rect_max_x = max(number_xcord[0],-BOUNDRYINC, 0),
min(WINDOWSIZEX, number_xcord[-1]+BOUNDRYINC)
            rect_min_y, rect_max_y = max(number_ycord[0], -BOUNDRYINC, 0),
min(number_ycord[-1] + BOUNDRYINC, WINDOWSIZEX)

            number_xcord = []
            number_ycord = []

            #pixel values of the image present in the display surface
            img_arr =
np.array(pygame.PixelArray(DISPLAYSURF))[rect_min_x:rect_max_x,
rect_min_y:rect_max_y].T.astype(np.float32)

            #save the image
            if IMAGESAVE:
                cv2.imwrite("image.png")
                image_cnt+=1

            if PREDICT:
                image = cv2.resize(img_arr, (28,28))
                image = np.pad(image, (10,10), 'constant', constant_values=0)
                image = cv2.resize(image, (28,28))/255

                label =
str(LABELS[np.argmax(MODEL.predict(image.reshape(1,28,28,1)))])

                textSurface = FONT.render(label,True, RED, WHITE)
                textRecObj = textSurface.get_rect()
                textRecObj.left, textRecObj.bottom = rect_min_x, rect_max_y

                DISPLAYSURF.blit(textSurface, textRecObj)

            if event.type == KEYDOWN:
                if event.unicode == "n":
                    DISPLAYSURF.fill(BLACK)

        pygame.display.update()
```

Model Performance:

To assess the effectiveness of the "Doodling" project, we have generated a performance comparison chart. The chart illustrates the recognition accuracy for each digit (0-9) using the trained model.



```
import pygame, sys
from pygame.locals
import numpy as np
from keras.models
import cv2

BOUNDRYINC = 5
WINDOWSIZEX = 640
WINDOWSIZEY = 480
WHITE = (255,255,2
BLACK = (0,0,0)
RED = (255,0,0)

#load model
MODEL = load_model

#label for every n
LABELS = {0:"Zero"
            5:"Five"

#Initialise our py
pygame.init()
FONT  = pygame.fon

#Window size (X,Y)
DISPLAYSURF = pyga
pygame.display.set

iswriting = False

number_xcord = []
number_ycord = []
image_cnt = 1
IMAGESAVE = False
PREDICT = True

while True:
```

```
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
```

Observations:

- The model appears to perform well across all digits, with recognition accuracy ranging from approximately 90% to 99%.
- Digits like 1, 7, and 9 show higher accuracy, while 5 and 8 exhibit slightly lower accuracy.

8. Future Work

8.1 Potential Enhancements

Hyperparameter Tuning:

Conduct a more extensive search for optimal hyperparameters, including learning rates, batch sizes, and the number of filters in convolutional layers.

Experiment with different activation functions and regularization techniques to find the best combination.

Architecture Variations:

Explore more complex CNN architectures or other types of neural networks (e.g., deeper CNNs, architectures like ResNet or Inception) to potentially improve model performance.

Transfer Learning:

Investigate the use of transfer learning by leveraging pre-trained models on larger datasets (e.g., ImageNet) and fine-tuning them for digit recognition. This may be especially beneficial if your dataset is limited.

Ensemble Models:

Construct ensemble models by combining predictions from multiple CNNs with diverse architectures or by using different data augmentation strategies. Ensemble learning can often lead to improved generalization.

Normalization Techniques:

Experiment with different normalization techniques beyond the standard min-max scaling. Techniques like batch normalization or layer normalization might contribute to better convergence and performance.

8.2 Exploration of Other Models

Deep Learning Models:

Investigate the performance of more advanced deep learning models like recurrent neural networks (RNNs) or long short-term memory networks (LSTMs) for sequence modeling of flattened images.

Attention Mechanisms:

Explore the integration of attention mechanisms in the model to focus on specific regions of the image that are more informative for digit recognition. This could improve the model's interpretability.

Capsule Networks:

Consider implementing capsule networks, an alternative to traditional convolutional architectures, to capture hierarchical relationships between features.

Explainability Techniques:

Implement explainability techniques to interpret model predictions, such as visualizing attention maps or using techniques like SHAP (SHapley Additive exPlanations).

8.3 Dataset Expansion and Diversity

Additional Datasets:

Integrate additional handwritten digit datasets or datasets with variations in writing styles to enhance the model's ability to generalize across different scenarios.

Data Augmentation Strategies:

Explore and implement additional data augmentation strategies that are tailored to handwritten digit recognition, considering variations in writing styles, stroke thickness, and angles.

8.4 Deployment and Real-world Application

Model Deployment:

Explore deployment strategies for the model in real-world scenarios, such as integrating the model into mobile applications or web services for practical use.

User Interface (UI)/User Experience (UX):

Develop a user-friendly interface for users to interact with the digit recognition system, allowing them to input handwritten digits and receive predictions.

Integration with OCR Systems:

Investigate integrating the digit recognition model with Optical Character Recognition (OCR) systems for broader applications in document processing.

These future work suggestions aim to guide further exploration and improvement in different aspects of the project, ranging from model architecture and hyperparameters to the deployment and usability of the system in real-world scenarios. It's important to prioritize these based on the goals and constraints of your project.

Comparison chart:

Let's compare code snippets which we are having in research paper and which we implemented:

Commonalities:

- Both codes involve the MNIST dataset, a popular dataset for handwritten digit recognition.
- They split the data into training and testing sets.
- They evaluate the performance of the models on the test set using accuracy and confusion matrix.
- They visualize some randomly selected test images along with their original and predicted labels.
- Both use external libraries such as OpenCV, NumPy, Matplotlib, and scikit-learn.

Differences:

The Code which is implemented in project uses (Convolutional Neural Network - CNN):

- Uses a deep learning model (Convolutional Neural Network) implemented with Keras.
- Utilizes a gradient descent optimizer (Stochastic Gradient Descent - SGD) for model training.
- Involves convolutional layers for feature extraction and pooling layers for down-sampling.
- The model is trained over multiple epochs.
- The script allows for saving and loading the trained model and weights.

The Code which is provided in Research Paper uses KNN and some have used CNN:

- Uses the k-Nearest Neighbors algorithm implemented with scikit-learn.
- No training process in the traditional sense; KNN stores all training instances and classifies new instances based on their proximity to stored instances.
- Involves calculating distances between data points to make predictions.
- The script saves and loads the trained KNN classifier using the pickle module.
- The script involves splitting the data into training and validation sets for KNN.

Comparison:

Model Type:

Code-1 uses a deep learning model (CNN) which can automatically learn hierarchical features.

Code-2 uses a simple and intuitive algorithm (KNN) based on instance-based learning.

Training Process:

Code-1 involves a training process with backpropagation and optimization.

Code-2 doesn't have a traditional training process for KNN; it stores all training instances and classifies new instances based on proximity.

Model Complexity:

CNNs, being deep learning models, are generally more complex and capable of capturing intricate patterns.

KNN is simpler and relies on the nearest neighbour’s voting for classification.

Performance:

CNNs, when appropriately configured and trained, can achieve high accuracy, especially for image recognition tasks.

KNN is generally simpler and might not perform as well on complex tasks, but it's easy to understand and implement.

Usage:

CNNs are suitable for tasks where feature learning is essential, such as image and speech recognition.

KNN is a versatile algorithm used for various tasks and is particularly useful when the decision boundaries are non-linear.

In summary, the choice between these approaches depends on the task at hand, the amount of data available, and the desired trade-off between simplicity and model performance.

Research Paper:

1. https://www.researchgate.net/profile/Younes-Alwan/publication/316938847_Handwritten_Digit_Recognition_Using_Convolutional_Neural_Networks/links/5919e2ae0f7e9b1db652885b/Handwritten-Digit-Recognition-Using-Convolutional-Neural-Networks.pdf
2. https://d1wqtxts1xzle7.cloudfront.net/58212013/Bangla-Handwritten-Character-Recognition-using-Convolutional-Neural-Network-libre.pdf?1547846190=&response-content-disposition=inline%3B+filename%3DBangla_Handwritten_Character_Recognition.pdf&Expires=1697048717&Signature=FVn3CjxTI4DEVo~wBeVRVyEAEhwyAu-QTCHWsUK8r64Orj3gUG~-SneXcCOecWHtNbOYfy9pSTDTlYgeg5egYZwcO-u6oCO~gzSELAl4bQ~-2xWzeaHQSQpr3-tqw21BGqhl6yorIGynXdVT-qW9aAzlXvxAlXkZN5oZtLTquEbUT8mXHl7oaqt-vTsbVlkrwNALnxTAX-l7lgWCd177H65~EHWPY39DGoZkd6X3-LwOKfVQgr14T3LS7BYFeVWinQQMb5hULshvEUtxhTmxhX746gs9qVSKcj15yb5W4gebG9VJ~D9MLONOPdRDJxuV1uSIDnZj3oPVryIBMmPFsw__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA
3. https://www.mdpi.com/2076-3417/9/15/3169
4. https://d1wqtxts1xzle7.cloudfront.net/67881176/Handwritten_Digit_Recognition_using_Conv20210705-17075-5z5naf.pdf?1625496753=&response-content-disposition=inline%3B+filename%3DHandwritten_digit_recognition_using_conv.pdf&Expires=1697048777&Signature=f7XzYN5LNf3EuxS89h2G~RJ~-KWVHVID7ILscK3eGgGNckNTxWIB5vJfdvzLpDCw1waFZ5eItSwYype6aJDZ-kzgzyUFp9zl5P5WM47pBha~k6ooGBgMpgDY8fAgE3AUXrCMEDdu9mMEvIy3DfpOOmXpcpU-PBY2PWU~~dQaLtLxyQ9bzRyneucL3jdvGU0WV4gei-Td9Mu0Pu9pvayWyvWWG4KZ~eKWQbXcmjcvLhU2z~diDSJ-h3YixPp1YiA5SZfwo2WdQCLUfPrO3D42lx~Se3msZihslkROtilH1ZW2i28fPirtF02PPFgBakIBb2ZJKt~mNXbzEw5izQzx5Q__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA
5. https://www.mdpi.com/1424-8220/20/12/3344

Conclusion:

The "Doodling" project is a successful application for handwritten digit recognition. By utilizing a Convolutional Neural Network, the system can accurately classify user-drawn digits. The comparison chart highlights the model's performance for each digit, and it is evident that the model performs exceptionally well.

As with any machine learning project, there is always room for improvement. Further optimization and data augmentation techniques may enhance the model's performance, particularly for digits with lower recognition accuracy. Additionally, user interface enhancements can make the application more user-friendly and accessible.

The "Doodling" project serves as a valuable tool for digit recognition and has the potential to be integrated into various applications, such as digit entry in forms and educational tools.