PROJECT 01: Building your first AI
Building a neural network to recognize handwritten Digits

INTRODUCTION

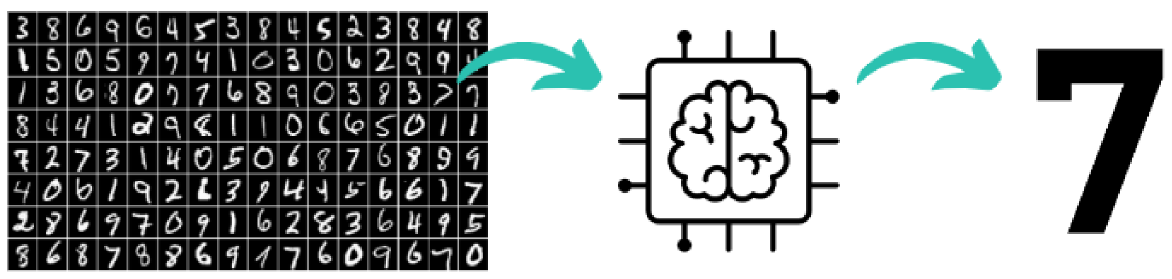
This tutorial is designed to introduce you to the basics of neural networks through a hands-on approach. We'll be using the MNIST dataset, one of the most famous datasets in the machine learning community, to build and train a simple neural network.

Whether you're a student, a hobbyist, or just someone interested in machine learning, this guide is designed to be accessible and engaging. So let's get started on this exciting journey into the realm of neural networks!

What are Neural Networks?
Neural networks are a type of machine learning model inspired by the way human brains work. They consist of layers of interconnected nodes (or "neurons") that process data and learn to perform specific tasks. Neural networks are particularly good at recognizing patterns in images, sounds, texts, and other complex datasets.

The MNIST Dataset
The MNIST dataset is a collection of 70,000 handwritten digits ranging from 0 to 9. It's widely used as a benchmark in the machine learning field for training and testing image processing systems.

The dataset is divided into two parts: 60,000 images for training and 10,000 for testing. Each image is a 28x28 pixel grayscale representation of a digit.

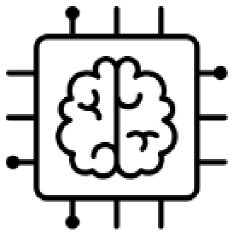


Purpose of This Tutorial
This tutorial aims to provide you with a clear and concise introduction to the world of neural networks. We'll start by setting up our programming environment, then dive into loading and preprocessing the MNIST dataset. Following that, we'll discuss how to build a neural network model, train it with our dataset, and evaluate its performance. By the end of this tutorial, you'll have a foundational understanding of neural networks and the skills to build one that can recognize handwritten digits.

RESOURCES

The complete code with explanations are available at:
https://github.com/czplus/AI-101

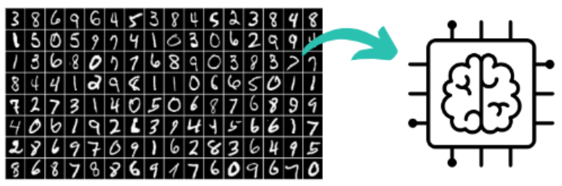
THEORETICAL OVERVIEW



Neural Network:
A neural network is a computational model inspired by the way biological neural networks in the human brain process information.

A neural network learns from the provided examples through a set of metrics provided. The neural network recognizes patterns and learns to predict.

The neural network **(also called as AI Model)** is a set of layers (mathematical operations) with several numerical parameters. In this tutorial, we will define a neural network architecture.

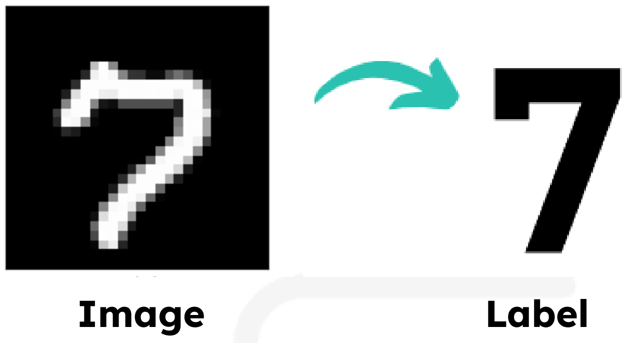


Dataset:
A dataset is a collection of data organized for training a neural network.

In this case, the data contains an image of a Handwritten digit and a corresponding label. The label indicates the value of the written digit and helps the neural network to learn.

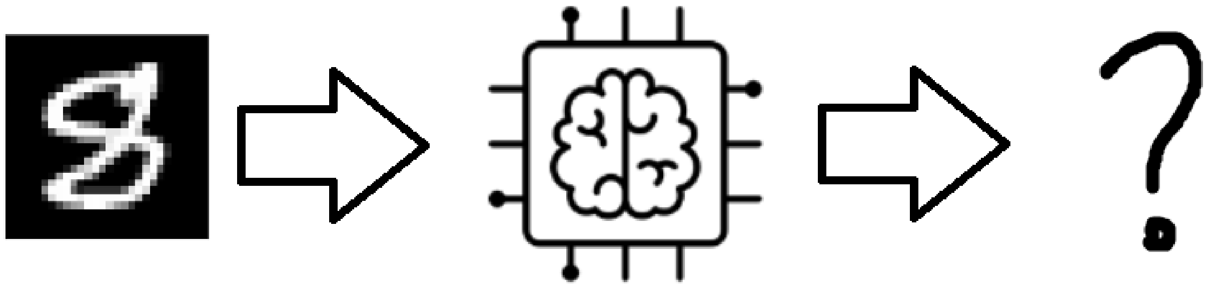
One Image and the corresponding label is passed to the Neural network at once. The neural network learns from the patterns of the image to match the label ‘7’.

This phase is called as - **Training the Neural Network** or **Training the Model.**



After the neural network is trained, the neural network is tested.
Provide an input that the neural network has not seen previous and it should be able to predict the digit in the image.

This phase is called as - **Testing / Evaluating the Model.**



CODE IN DETAIL

1. Imports:

```
# Data related imports
import numpy as np
from sklearn.model_selection import train_test_split

# ML / AI Specific Imports
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets, layers, optimizers , Sequential

# Imports for Visualizations
import matplotlib.pyplot as plt

# Other Util Imports
import random
```

Data related imports

`“import numpy as np”` : This imports the NumPy library, which is fundamental for scientific computing in Python. NumPy is used for working with arrays, performing mathematical operations, and more. The np alias is a standard convention for easier access to NumPy functions.

`“from sklearn.model_selection import train_test_split”` : This imports the train_test_split function from the model_selection module of the Scikit-learn library. This function is used to split a dataset into training and testing sets, which is a crucial step in evaluating the performance of machine learning models.

ML / AI Specific Imports

`“import tensorflow as tf”` : This imports TensorFlow, a popular and powerful library used for both machine learning and deep learning applications. TensorFlow provides tools to build and train models, and the tf alias is commonly used for convenience.

`“from tensorflow import keras”` : Keras is a high-level API for building and training deep learning models and is integrated within TensorFlow. This import provides easy access to Keras functionalities.

`from tensorflow.keras import datasets, layers, optimizers , Sequential`: This line imports specific modules and classes from TensorFlow's Keras API:

- `datasets`: Preloaded datasets, which might include the MNIST dataset used for digit recognition.
- `layers`: Various layers used in constructing neural networks, like Dense, Convolutional, Pooling layers, etc.
- `optimizers`: Different optimization algorithms like Adam, SGD, etc., used to minimize the loss function during training.
- `Sequential`: A class used for building models in a linear stack of layers

<u>Imports for Visualizations</u>

"`import matplotlib.pyplot as plt`" : This imports Matplotlib, a widely-used Python library for data visualization. The plt alias is standard for calling Matplotlib's plotting functions.

<u>Other Util Imports</u>
"`import random`" : This imports Python's built-in random module, which includes functions that generate random numbers and perform random operations. This can be useful for shuffling data or initializing weights in a neural network randomly.

2. Configurations

```python
# channel = 1 for gray scale Images
image_rows,image_colums,channel=28,28,1

# classes: From class-0 to class-9
num_classes=10

# Input shape of the image is of the format: (Length X Width X Channels)
input_shape=(image_rows,image_colums,channel)
```

- *image_rows* and *image_columns* are set to 28 each. This indicates that the images to be processed are 28 pixels in height (image_rows) and 28 pixels in width (image_columns).

- Channel is set to 1, which signifies that the images are grayscale. In image processing, color images are typically represented with 3 channels (Red, Green, and Blue), but grayscale images only require one channel.

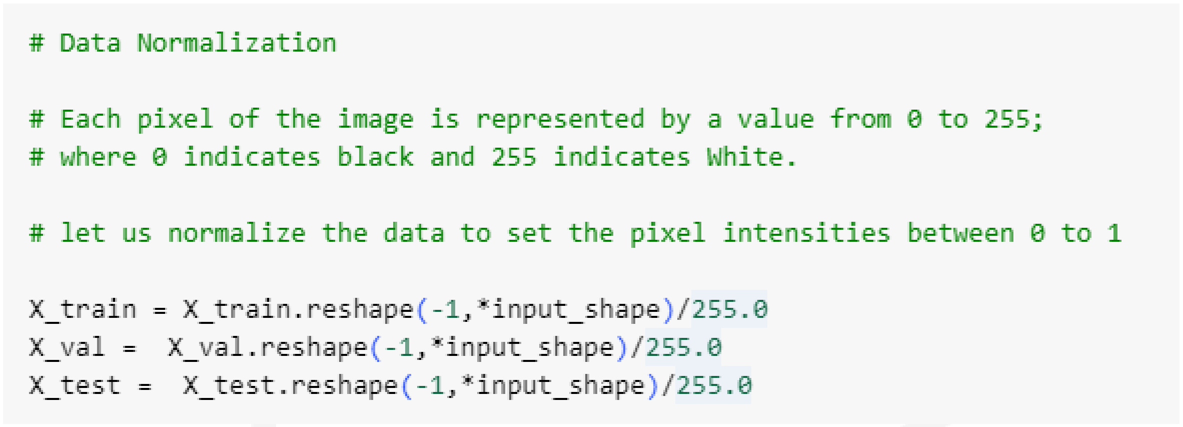
- The last line defines the input_shape of the images that will be fed into the neural network. It uses the dimensions set earlier: 28 pixels in height, 28 pixels in width, and 1 channel (grayscale).

- The *num_classes* is set to 10, implying that there are 10 different categories or classes that the model will classify the images into. In the context of digit recognition, these classes represent the digits from 0 to 9.

- This *input_shape* is a standard way to represent the dimensions of the input data in many machine learning frameworks, including Keras, which is part of TensorFlow. It's particularly important for convolutional neural networks, which require a specific input shape for the first layer.

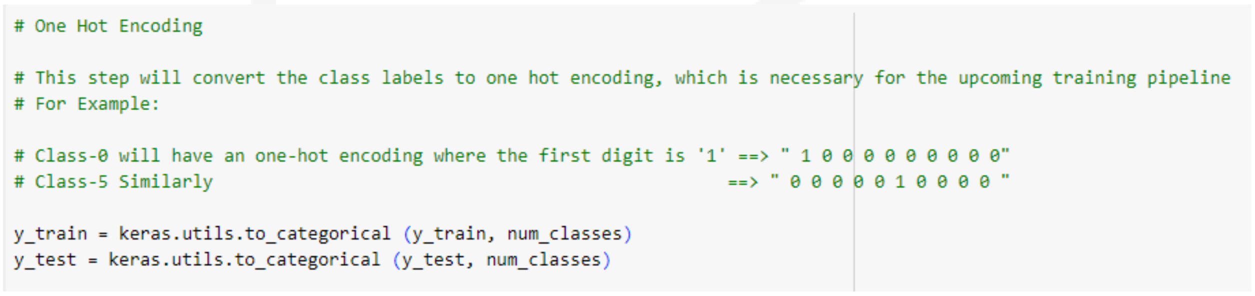
3. Data Normalization

Convert the pixel values from [0, 255] → [0,1]
Normalization helps in improving the calculation speed using Floating point numbers.



```
# Data Normalization

# Each pixel of the image is represented by a value from 0 to 255;
# where 0 indicates black and 255 indicates White.

# let us normalize the data to set the pixel intensities between 0 to 1

X_train = X_train.reshape(-1,*input_shape)/255.0
X_val =  X_val.reshape(-1,*input_shape)/255.0
X_test =  X_test.reshape(-1,*input_shape)/255.0
```

One Hot Encoding

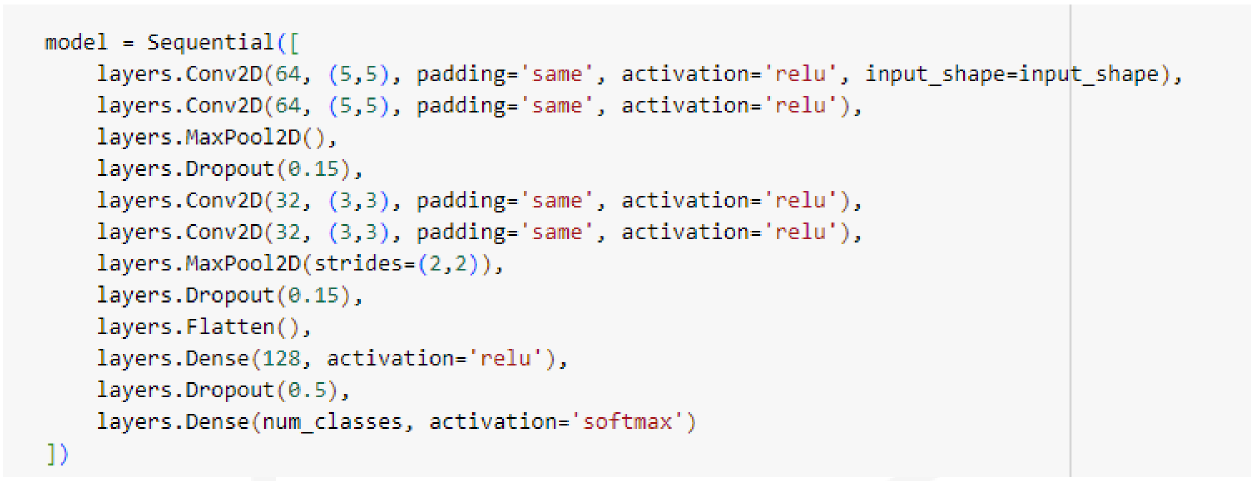


```
# One Hot Encoding

# This step will convert the class labels to one hot encoding, which is necessary for the upcoming training pipeline
# For Example:

# Class-0 will have an one-hot encoding where the first digit is '1' ==> " 1 0 0 0 0 0 0 0 0 0"
# Class-5 Similarly                                                ==> " 0 0 0 0 0 1 0 0 0 0 "

y_train = keras.utils.to_categorical (y_train, num_classes)
y_test = keras.utils.to_categorical (y_test, num_classes)
```

One-Hot Encoding:
One-hot encoding transforms the integer labels into a binary matrix representation. For example, if num_classes is 10, the integer label 2 would be converted to [0, 0, 1, 0, 0, 0, 0, 0, 0, 0] - a 10-element array where all elements are 0, except for the third element, which is 1.

This conversion is important for classification with neural networks because it allows the network to output probabilities for each class through its final layer, typically with a softmax activation function.

4. Model Architecture



```
model = Sequential([
    layers.Conv2D(64, (5,5), padding='same', activation='relu', input_shape=input_shape),
    layers.Conv2D(64, (5,5), padding='same', activation='relu'),
    layers.MaxPool2D(),
    layers.Dropout(0.15),
    layers.Conv2D(32, (3,3), padding='same', activation='relu'),
    layers.Conv2D(32, (3,3), padding='same', activation='relu'),
    layers.MaxPool2D(strides=(2,2)),
    layers.Dropout(0.15),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])
```

The code defines a convolutional neural network (CNN) model using the Sequential API from Keras. The model consists of several layers, each designed for a specific function in image processing and classification:

- Convolutional Layers (Conv2D): The Conv2D layers apply convolution operations to the input. They help the model learn features from the images. This model has four convolutional layers.

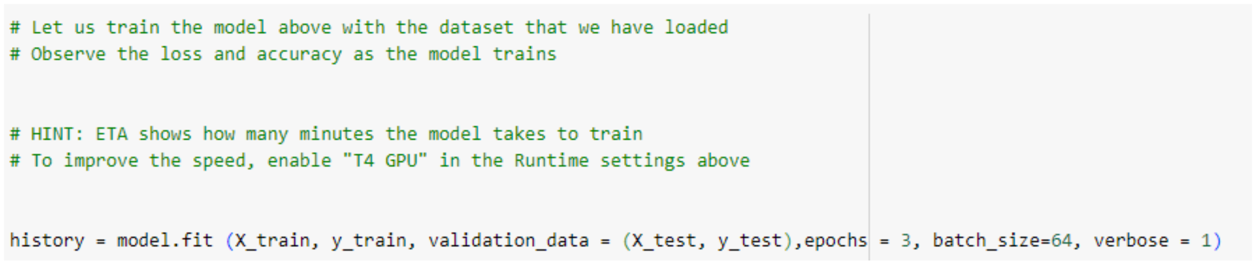
- Max Pooling Layers (MaxPool2D): These layers reduce the spatial dimensions (height and width) of the input volume for the next convolutional layer. They help in reducing the computational load, memory usage, and the number of parameters.

- Dropout Layers (Dropout): These layers randomly set a fraction of input units to 0 at each update during training, which helps in preventing overfitting.

- Flatten Layer (Flatten): This layer flattens the 3D output of the last convolutional layer into a 1D array. It's necessary because the dense layers expect 1D arrays as input.

- Dense Layers (Dense): The flattened output is then passed through two dense (fully connected) layers. The softmax function outputs a probability distribution over the classes.

5. Model Training



```
# Let us train the model above with the dataset that we have loaded
# Observe the loss and accuracy as the model trains


# HINT: ETA shows how many minutes the model takes to train
# To improve the speed, enable "T4 GPU" in the Runtime settings above


history = model.fit (X_train, y_train, validation_data = (X_test, y_test),epochs = 3, batch_size=64, verbose = 1)
```

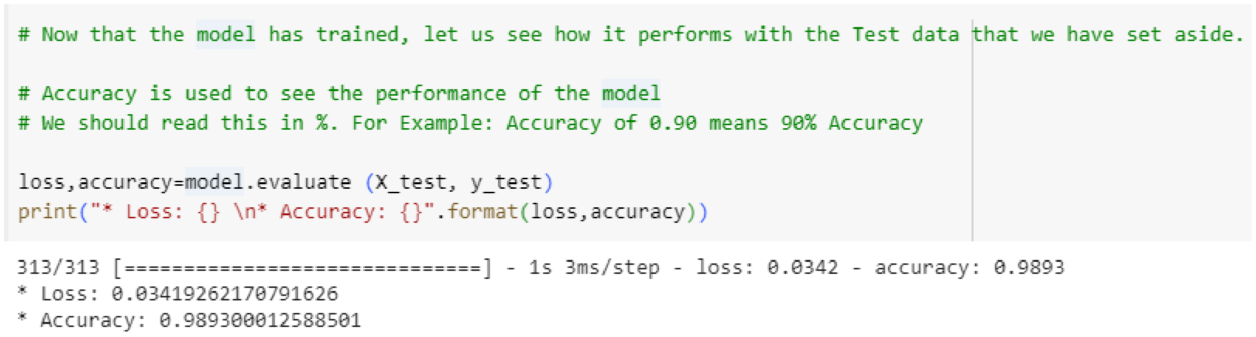
`“model.fit”` : This is a method provided by Keras to train the model on a given dataset. It involves the process of feeding the input data to the neural network, calculating the loss (difference between the predicted output and actual output), and adjusting the weights of the network using backpropagation.

`X_train, y_train` : These are the training data and their corresponding labels. The model will learn from this data.

`epochs = 3` : This specifies the number of times the learning algorithm will work through the entire training dataset. Here, it's set to 3, meaning the entire dataset will be passed forward and backward through the neural network three times.

`batch_size=64` : This defines the number of samples that will be propagated through the network at one time. Here, it's set to 64, meaning 64 images and labels from *X_train* and *y_train* will be loaded into the model at a time during training.

6. Performance Evaluation



```
# Now that the model has trained, let us see how it performs with the Test data that we have set aside.

# Accuracy is used to see the performance of the model
# We should read this in %. For Example: Accuracy of 0.90 means 90% Accuracy

loss,accuracy=model.evaluate (X_test, y_test)
print("* Loss: {} \n* Accuracy: {}".format(loss,accuracy))
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0342 - accuracy: 0.9893
* Loss: 0.03419262170791626
* Accuracy: 0.989300012588501
```

`loss, accuracy = model.evaluate(X_test, y_test)` : This line uses the evaluate method of the Keras model to assess its performance on the test dataset. *X_test* and *y_test* are the inputs and labels of the test dataset, respectively.

Cognitive Zillion AI 101 Series

The **Cognitive Zillion AI 101 Series** is expertly crafted for beginners and enthusiasts eager to embark on their journey in artificial intelligence. This comprehensive series offers an array of resources, including engaging videos, intuitive tutorials, and ready-to-execute code, all designed to simplify the learning process for newcomers.

Delve into a diverse range of hands-on projects that the AI 101 Series showcases. For further insights and to enhance your learning experience, follow us on YouTube, Instagram, and GitHub. Join our community and start transforming your AI aspirations into reality today.

YOUTUBE:
https://www.youtube.com/@CognitiveZillion

INSTAGRAM:
https://www.instagram.com/cognitivezillion/

GITHUB:
https://github.com/czplus

EMAIL:
Cognitivezillion@gmail.com