A Beginner's Guide to Deep Learning with MNIST Dataset

 Azim Khan Follow 6 min read · Apr 16, 2024

INTRODUCTION :

Welcome to the world of deep learning! In this blog, we'll embark on an exciting journey to understand the fundamentals of deep learning using the MNIST dataset. MNIST is a classic dataset widely used in the field of machine learning and deep learning for digit recognition tasks. By the end of this blog, you'll have a solid understanding of how deep learning works and how to build a simple neural network model using TensorFlow and Keras.

Understanding the MNIST Dataset:

The MNIST dataset consists of a vast collection of handwritten digits from 0 to 9. It's like a giant library filled with pictures, where each picture is a grayscale image measuring 28x28 pixels. Our mission is to teach a deep learning model to look at these images and tell us which digit they represent. With a whopping 60,000 images for training and another 10,000 for testing, we have plenty of examples to work with. So, let's roll up our sleeves and get ready to train our model to become a digit recognition expert!

Step 1 :

importing some important libraries


```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,Flatten
```

Loading and Exploring the Data:

We start by loading the MNIST dataset using TensorFlow's Keras API. This dataset is divided into training and testing sets. We visualize a sample image

from the dataset to get a sense of the data we're working with.

```
(X_train,y_train),(X_test,y_test) = keras.datasets.mnist.load_data()
```

In this line of code, we're using TensorFlow's Keras API to load the MNIST dataset. This dataset is a collection of handwritten digits, widely used for training and testing deep learning models. The `load_data()` function splits the dataset into two parts: `X_train` and `y_train` contain images of handwritten digits and their corresponding labels, respectively, for training the model. Similarly, `X_test` and `y_test` hold images and labels for testing the model's performance. Essentially, these lines set up our dataset, providing the input images and their corresponding correct answers for our deep learning model to learn from and evaluate against.

#Checking the image

Let's print y_train so that we will get the output

```
y_train
```

```
y_train
```

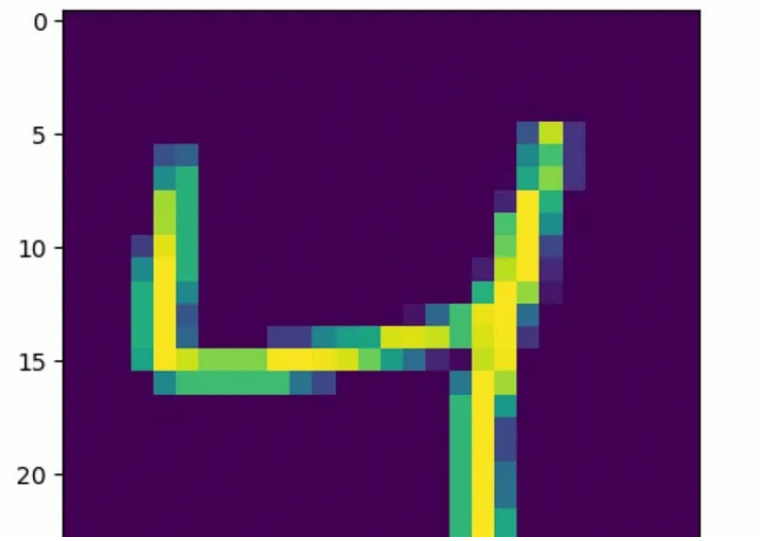
```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

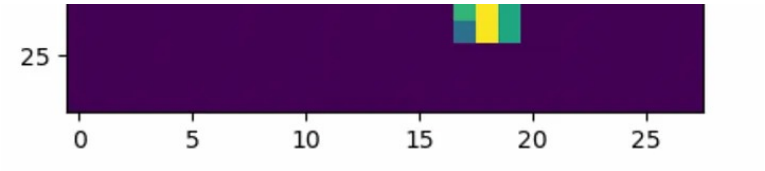
So from the output, we can see the 3rd value is 4 so we plot that x_train[2]

```
import matplotlib.pyplot as plt
plt.imshow(X_train[2])
```

```
import matplotlib.pyplot as plt
plt.imshow(X_train[2])
```

```
<matplotlib.image.AxesImage at 0x31088ce10>
```





So yeah, here we can see we get 4

Preprocessing the Data:

Before we can train our model, it's essential to preprocess the data. For that, we Normalize pixel values to the range [0, 1], because as we see our pixel value ranges between 0 to 255.

```
x_train, x_test = x_train / 255.0, x_test / 255.0
```

In these lines of code, we're preparing our image data for training our model. The variable `x_train` holds the images from our training dataset, while `x_test` contains images from the testing dataset. We divide each pixel value in these images by 255.0. Why? Well, pixel values typically range from 0 to 255, representing different shades of gray. By dividing them all by 255.0, we're essentially scaling these values down to a range between 0 and 1. This normalization step ensures that our neural network can learn effectively without being influenced too much by large pixel values.

Building the Neural Network Model:

Now comes the exciting part — building our neural network model! We construct a simple sequential model using TensorFlow and Keras. The model comprises a flatten layer to convert the 2D image into a 1D array, followed by dense layers with various activation functions. We use the softmax activation function in the final layer for multi-class classification.

```
model= Sequential()
model.add(Flatten(input_shape=(28,28)))
model.add(Dense(128, activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(11,activation='softmax'))
model.summary()
```

This output represents the architecture of our neural network, called "Sequential." Each layer in the network has different characteristics. The "Flatten" layer converts our 2D image data into a 1D array. Then, we have three "Dense" layers. The number within parentheses denotes the output shape of each layer. The "Param #" column shows the number of parameters (weights and biases) in each layer. These parameters are learned during training to make predictions. The "Total params" section sums up all the parameters in the model, while "Trainable params" refers to those parameters that the model can learn from the data.

Compiling and Training the Model:

we're setting up and training our neural network model. First, we compile the model, specifying the loss function ('sparse_categorical_crossentropy') — a measure of how well our model is performing, the optimizer ('Adam') — a method to adjust the model's parameters to minimize the loss, and the metric we want to track ('accuracy'). Then, we train the model using the training data (X_train and y_train) for 25 epochs (or iterations), validating its performance on a portion of the data (20% in this case) to ensure it's learning effectively without overfitting.

```
model.compile(loss='sparse_categorical_crossentropy',optimizer='Adam',metrics=['
history = model.fit(X_train,y_train,epochs=25,validation_split=0.2)
```

These outputs represent the progress of our model during training. Each "Epoch" signifies one complete pass through the entire training dataset. The "accuracy" and "loss" values show how well the model is performing on both the training and validation sets. We aim for high accuracy and low loss.

Evaluating Model Performance:

Once the model is trained, we evaluate its performance on the testing data. We use accuracy as the metric to measure how well our model predicts the digits. These lines of code are used to evaluate the performance of our trained model. First, we predict the probabilities of each digit using the testing data. Then, we select the digit with the highest probability as our prediction. Finally, we calculate the accuracy of these predictions compared to the actual labels.

```
y_prob = model.predict(X_test)
y_pred = y_prob.argmax(axis=1)
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Plotting :

FOR LOSS

Here we use Matplotlib to plot the training loss (how well the model

performs on the training data) and the validation loss (how well the model generalizes to new data) over each epoch during training. It helps us visualize how our model learns and whether it's overfitting or underfitting.

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```


Here we can see how perfectly my model is trained

FOR ACCURACY

Here visualize the accuracy of our model during training and validation. The `plt.plot()` function creates a line plot using the accuracy values stored in `history.history['accuracy']` and `history.history['val_accuracy']`. This helps us see how well our model performs on both training and validation data across epochs.

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```


And Here we can how my model is accurate

Making Predictions:

Finally, we demonstrate how to use our trained model to make predictions on new unseen data. We provide a sample image from the testing set and predict the digit it represents. Here The first line displays the third image from the testing dataset using matplotlib's imshow function. The second line predicts the digit represented by the image using our trained model. It reshapes the image into a format compatible with the model, then predicts the digit and returns the predicted label.

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

Finally, "array([1])" represents the predicted digit label, which is 1 in this case.

Conclusion:

In conclusion, we've learned the basics of deep learning using the MNIST dataset. We've seen how to preprocess data, build a neural network model, train it, evaluate its performance, and make predictions. Deep learning opens up a world of possibilities for solving complex problems, and MNIST serves as an excellent starting point for beginners to dive into this fascinating field.

DATASET AND SOURCECODE:

MNIST_DatasetANN.ipynb
Edit description
drive.google.com

Thank You

Data Science | Deep Learning | Mnist | TensorFlow | Keras

Written by Azim Khan

32 followers · 5 following

Data Scientist

Follow

No responses yet

Write a response

What are your thoughts?

More from Azim Khan

Azim Khan

Email Spam Detection with Machine Learning: A...

In today's world, email has become a crucial way for people to communicate. But along...

Mar 22, 2024 167

Azim Khan

Understanding Churn Prediction with Artificial Neural Networks...

Introduction:

Apr 9, 2024 7

Azim Khan

Data cleaning for Machine Learning Model

Introduction:

Mar 12, 2024 1 1

Azim Khan

Hotel Rating Classification with Machine Learning

Welcome to our guide on hotel rating classification using machine learning! In this...

Apr 4, 2024 9

See all from Azim Khan

Recommended from Medium

Hemasri

Understanding RNN, LSTM, and GRU

A Deep Dive into Recurrent Neural Networks

Dec 18, 2024 1

In The Deep Hub by Palash Mishra

Getting Started with PyTorch: A Beginner-Friendly Guide

If you've ever wondered how to build and train deep learning models, PyTorch is one of the...

Dec 4, 2024 70

Piyush Kashyap

Transfer Learning in PyTorch: Fine-Tuning Pretrained Models for...

In recent years, deep learning has revolutionized the way we approach comple...

Dec 5, 2024

In Nailing the AI ML Interview by Dr. R. Li

ML Coding Interview: Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent (SGD) is an optimization algorithm used in machine...

Mar 12

Garvit Sapra

Understanding Deep learning Neural Networks: Key Concepts...

Neural Networks: The Building Blocks of Modern AI

Dec 22, 2024 17

Unicorn Day

Building an Image Vector Database with ResNet50 and FAISS 🖼️🔍

In today's world of machine learning, managing large sets of images requires...

Dec 23, 2024 4

See more recommendations