TensorFlow MNIST: Read Your Own Handwritten Digit Video Walkthrough Tutorial

In this tutorial, we'll build a TensorFlow model that can read handwritten digits. We'll use the MNIST dataset, which is a large dataset of grayscale images of handwritten digits. We'll use a convolutional neural network (CNN) to recognize the digits in the images.

This tutorial is a video walkthrough, so you can follow along and see exactly how to build the model. We'll cover everything from loading the data to training the model to evaluating its performance.

TensorFlow MNIST I Read your own handwritten digit I Video Walkthrough Tutorial (58+ min.) with Source Code I Cuda Education by Neville Goddard

★★★★★ 5 out of 5

Language	: English
File size	: 801 KB
Text-to-Speech	: Enabled
Screen Reader	: Supported
Enhanced typesetting	: Enabled
Print length	: 17 pages
Lending	: Enabled

Loading the Data

The first step is to load the MNIST dataset. We can do this using the TensorFlow `keras.datasets` module.

python (x_train, y_train),(x_test, y_test) = keras.datasets.mnist.load_data()

The `x_train` and `y_train` variables contain the training data, and the `x_test` and `y_test` variables contain the test data. The `x` variables contain the images of the handwritten digits, and the `y` variables contain the labels for the digits.

Preprocessing the Data

Before we can train the model, we need to preprocess the data. This involves scaling the pixel values to be between 0 and 1, and converting the labels to a one-hot encoding.

python x_train = x_train.astype('float32') / 255.0 x_test = x_test.astype('float32') / 255.0

y_train = keras.utils.to_categorical(y_train, 10) y_test = keras.utils.to_categorical(y_test, 10)

The `astype` function scales the pixel values to be between 0 and 1. The `to_categorical` function converts the labels to a one-hot encoding. This means that each label is represented by a vector of 10 elements, with a 1 in the position of the correct label and 0s in all other positions.

Building the Model

Now that the data is preprocessed, we can build the model. We'll use a CNN with two convolutional layers, followed by a fully connected layer.

python model = keras.models.Sequential([keras.layers.Conv2D(32, (3, 3),activation='relu', input_shape=(28, 28,

1)),keras.layers.MaxPooling2D((2, 2)),keras.layers.Conv2D(64, (3, 3),activation='relu'),keras.layers.MaxPooling2D((2, 2)),keras.layers.Flatten(),keras.layers.Dense(128, activation='relu'),keras.layers.Dense(10, activation='softmax')])

The first two layers in the model are convolutional layers. These layers use a filter to convolve over the input data. The filter is a small matrix of weights, and the convolution operation involves multiplying the filter by the input data and summing the products. The output of the convolution is a feature map, which is a 2D representation of the features that the filter has detected in the input data.

The `MaxPooling2D` layers are used to reduce the dimensionality of the feature maps. These layers take the maximum value from each 2x2 block of the input data. This reduces the size of the feature maps by a factor of 2 in each dimension.

The `Flatten` layer is used to convert the feature maps into a 1D array. This is necessary before we can pass the data to the fully connected layer.

The `Dense` layers are fully connected layers. These layers take a vector of input data and produce a vector of output data. The first fully connected layer has 128 units, and the second fully connected layer has 10 units.

Training the Model

Once the model is built, we can train it on the training data. We'll use the `compile` method to specify the loss function, optimizer, and metrics to track during training.

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

model.fit(x_train, y_train, epochs=10)

The `compile` method takes three arguments:

* `optimizer` : The optimizer to use to update the model's weights during training. We're using the Adam optimizer, which is a popular choice for training deep learning models. * `loss` : The loss function to use to measure the error between the model's output and the true labels. We're using the categorical crossentropy loss, which is a common choice for multi-class classification problems. * `metrics` : A list of metrics to track during training. We're tracking the accuracy, which is the percentage of correct predictions.

The `fit` method takes two arguments:

* `x_train` : The training data. * `y_train` : The labels for the training data. * `epochs` : The number of epochs to train the model for. An epoch is one pass through the entire training dataset.

Evaluating the Model

Once the model is trained, we can evaluate its performance on the test data. We'll use the `evaluate` method to calculate the loss and accuracy on the test data.

python model.evaluate(x_test, y_test)

The `evaluate` method returns a list of two values:

* The loss on the test data. * The accuracy on the test data.

Making Predictions

Once the model is trained, we can use it to make predictions on new data. We can pass a new image to the model, and it will return the predicted label for the image.

python new_image = ...

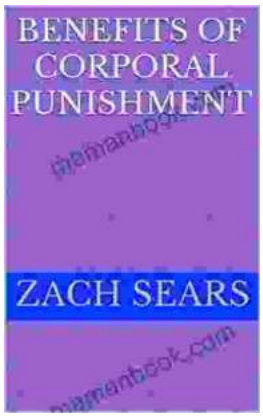
prediction = model.predict(new_image)

The `predict` method takes an image as input and returns the predicted label for the image.

In this tutorial, we built a TensorFlow model that can read handwritten digits. We used a CNN to recognize the digits in the images. We also learned how to preprocess the data, train the model, and evaluate its performance.

This tutorial is a great starting point for learning how to build deep learning models. You can use the skills that you learned in this tutorial to build your own models for other tasks, such as image classification, object detection, and natural language processing.

Screen Reader	: Supported
Enhanced typesetting	: Enabled
Print length	: 17 pages
Lending	: Enabled



Benefits of Corporal Punishment: A Review of the Literature

Corporal punishment is a form of physical discipline that involves the use of force to inflict pain on a child. It is a controversial topic, and there is much debate about its...

The Premier Package: Candace Quickies - A Comprehensive Review of the Ultimate Do-It-Yourself Cleaning Solution

Candace Quickies is a revolutionary do-it-yourself cleaning solution that has taken the home cleaning industry by storm. With a deep...