Developing a Convolutional Neural Network for Handwritten Digit Classification

Introduction:

Handwritten digit classification is a fundamental problem in the field of computer vision and machine learning. It involves recognizing and categorizing handwritten digits, which can have variations in length, width, orientation, and position. This task finds applications in various areas such as recognizing handwriting on tablets, entering handwritten numeric values into computerized systems, and processing bank checks. In this write-up, we will discuss the development of a Convolutional Neural Network (CNN) for handwritten digit classification.

Neural Network Used (CNN):

A Convolutional Neural Network is a type of deep learning model that is particularly effective in image recognition tasks. It excels at hierarchical feature extraction, differentiating between low-level features (e.g., edges, corners) and high-level features (e.g., shape of the digit).

Handling Spatial Invariance:

One of the key advantages of CNNs is their ability to handle spatial invariance. This means that regardless of the location of the digit within the picture, the CNN is able to identify the digit. This is achieved through the use of convolutional layers, which apply filters across the image, capturing patterns and features at different locations.

Pooling Layers:

Another important component of CNNs is pooling layers. These layers help reduce the computational cost by decreasing the number of parameters passed onto the next layer. Pooling focuses on essential features, such as the shape of the digit or any consistent patterns within the same digit. By down sampling the input, pooling layers retain the most important information while reducing the spatial dimensions.

Dataset and Preprocessing Details:

For this task, the MNIST dataset is used. The dataset consists of grayscale images containing handwritten samples of digits from 0 to 9, with each image having a size of 28 pixels by 28 pixels. The dataset is divided into a training set and a testing set.

Preprocessing steps are applied to prepare the data for training the model. The dataset is loaded from the Keras library. Each image is reshaped into a 4D tensor to indicate the size of the image, color channel, and original shape. The pixel values of the images are normalized between 0 and 1 by dividing them by 255 since the images are grayscale and the pixel values range from 0 to 255. The target values, representing the digits, are one-hot encoded to convert them into a categorical format with 10 classes (0-9).

Testing and Training:

The following steps are performed for testing and training the CNN using the MNIST dataset:

* Creating a Sequential model: A Sequential model is created, allowing layers to be added sequentially.
* Adding a Convolutional Layer: A convolutional layer is added to the model with 32 filters and a Rectified Linear Unit (ReLU) activation function. This layer extracts features from the input images.
* Adding a Max-Pooling Layer: A max-pooling layer is added to reduce the spatial dimensions of the features captured by the convolutional layer.
* Repeating Steps 2 and 3: The above two steps are repeated to further extract and down sample the features.
* Adding a Flattening Layer: A flattening layer is added to reshape the output from the previous layers into a 1-dimensional vector, preparing it for the fully connected layers.
* Adding Dense Layers: Two dense layers are added to the model. The first dense layer has 128 neurons and a ReLU activation function, which introduces non-linearity and learns complex patterns from the extracted features. The second dense layer has 10 neurons, representing each of the 10 classes (0-9), and uses a Softmax activation function for the output layer to obtain probability distributions over the 10 classes.

Training the Model:

The training data, represented by X\_train and y\_train, is used to train the model. X\_train contains the images, and y\_train contains the corresponding labels. The testing data, represented by X\_test and y\_test, is used for evaluating the model's performance.

During training, the model is trained for 10 epochs, meaning it goes through the entire training dataset 10 times. The batch size is set to 32, which indicates that per gradient update, 32 samples will be processed. This allows for efficient training by updating the model's parameters based on mini-batches of data.

To prevent the loss of progress in case of program crashes or interruptions, a callback function is defined. This function saves the model at each epoch and returns to the last saved model, providing the best accuracy.

After training, the model is saved in an .h5 file, which can be used in a Flask application or any other deployment scenario.

Conclusion:

In this write-up, we discussed the development of a Convolutional Neural Network (CNN) for handwritten digit classification. We explored the key components of CNNs, such as hierarchical feature extraction, handling spatial invariance, and the use of pooling layers. We also described the MNIST dataset used for training and testing the model, along with the preprocessing steps involved.

By following the steps outlined, including the creation of a Sequential model, adding convolutional and pooling layers, and utilizing dense layers for classification, we can develop an effective CNN for handwritten digit classification. The trained model can then be saved and deployed in applications that require recognizing and categorizing handwritten digits.

Reference Link:

<https://www.researchgate.net/publication/361728427_Study_and_Develop_a_Convolutional_Neural_Network_for_MNIST_Handwritten_Digit_Classification>