Conquering Fashion MNIST With CNNs Using Computer Vision

TEAM RIYA- SAINTGITS COLLEGE OF ENGINEERING

June 2023

1 Model

Link to the model

2 Tasks Performed

2.1 Importing libraries

We import Tensorflow library as it provides a wide range of operations and functions for building and training machine learning models.

We also import the Fashion MNIST dataset

2.2 Loading data

We load the Fashion MNIST dataset. The Fashion MNIST dataset is a collection of grayscale images that represent different clothing item categories. It serves as a popular benchmark dataset for image classification tasks

We load it using 'fashion_mnist.load_data()'.

2.3 Preprocessing data

The dataset consists of gray-scale images with a size of 28x28 pixels. However, our deep learning model expects the input to have a specific shape, including a channel dimension. Therefore, we reshape the training and test data to (number_of_samples, 28, 28, 1). Additionally, we normalize the pixel values by dividing them by 255.0, which scales them to the range of 0 to 1.

2.4 Applying data augmentation

An ImageDataGenerator object is created to apply data augmentation techniques to the training dataset. Data augmentation helps increase the diversity and size of the training data by applying random transformations such as rotations, shifts, and flips.

The datagen.fit(x_train) call prepares the generator with the desired augmentation settings.

2.5 Building the model

A sequential model is created using Sequential(), which allows us to stack layers in a linear fashion. Convolutional layers, pooling layers, flatten layer, and fully connected layers are added to the model using the add() method.

The chosen activation function for the convolutional and fully connected layers is ReLU (Rectified Linear Unit). The final layer uses the softmax activation function, which provides probabilities for each class.

We build our deep learning model using the Sequential API provided by TensorFlow. This API allows us to define a linear stack of layers for our model.

We start by adding a convolutional layer (Conv2D) with 32 filters, each of size 3x3, and apply the ReLU activation function. This layer is responsible for extracting features from the input images.

We then add a max-pooling layer (MaxPooling2D) with a pool size of 2x2. This layer reduces the spatial dimensions of the input, focusing on the most important features.

We repeat this pattern two more times, adding another pair of convolutional and max-pooling layers. This helps the model learn more complex patterns.

We also use batch normalization after every layer. Batch normalization can help stabilize the training process and improve generalization.

After the convolutional layers, we add a Flatten layer to convert the 2D output from the previous layer into a 1D vector. This prepares the data for the following fully connected layers.

We add two fully connected layers (Dense) with 64 units each. The ReLU activation function is applied to introduce non-linearity.

We add dropout regularization as it can help reduce overfitting by randomly dropping out a fraction of the neurons during training.

Finally, we add a dense layer with 10 units, corresponding to the number of output classes in Fashion MNIST. The softmax activation function is used to produce probabilities for each class.

2.6 Compiling the model

The model is compiled using the compile() method. The optimizer Adam is chosen, which is an efficient optimization algorithm. The loss function is set to 'sparse_categorical_crossentropy' since we have integer-encoded labels. The metric to track during training and evaluation is set to 'accuracy'.

2.7 Defining and creating a learning rate scheduler callback

We use a learning rate scheduler to dynamically adjust the learning rate during training. This can help improve convergence and accuracy.

2.8 Training the model[Test 1]

The model is trained using the fit() method. The training data, augmented using the datagen.flow() method, is provided along with the number of epochs and batch size. The validation data (x_test, y_test) is used to monitor the model's performance during training.The training process iterates over the dataset for a specified number of epochs, updating the model's weights to minimize the loss and improve accuracy. This test verifies whether the model can learn and improve its predictions over multiple epochs of training.

2.9 Evaluating the model[Test 2]

After training, the model's performance is evaluated on the test data using the evaluate() method. The test accuracy and test loss are printed to assess how well the model performs on unseen data.

The test loss represents how well the model's predictions match the true labels, while the test accuracy indicates the percentage of correct predictions. This test provides an assessment of how well the model generalizes to unseen data and helps estimate its real-world performance.

3 Final accuracy

The final accuracy ended up being 0.910099983215332.

4 References

Geeksforgeek
Github
Machinelearningmaster
Tutorialspoint