Convolutional Neural Network

Constructing and training convolutional neural network for MNIST classification.

- Regularizing the training of the neural network through dropout
- Regularizing the training of neural network through augment selection of 1000 images
- Rotating images for 1-3 degrees clockwise and counter clockwise
- Shifting images for 3 pixels in 8 different directions

Tools & Liblaries: Python 3, Tensorflow, Numpy

Report

Convolutional Neural Network is based on building the invariance properties into the structure of a neural network. The key property of images - nearby pixels are more strongly correlated than more distant pixels, ConvNet use this property and thus is widely used for image classification. There are also can be multiple channels, depending on the image characteristics, 3 channels for colorful and 1 channel for gray-scale images. MNIST dataset has only 1 channel, which decrease the computational time.

There are a number of parameters and settings that can be tuned to improve the accuracy rate and the execution time for convolutional neural network:

Convolution filters

Convolution filter is a sliding window that convolve the input. By default for this task the filter of 5x5 dimensions was used.

We can also update the number of filters used. That describes the depth of the output volume. For our case there were used 16 filters for the first convolutional layer and 32 for the second.

Stride

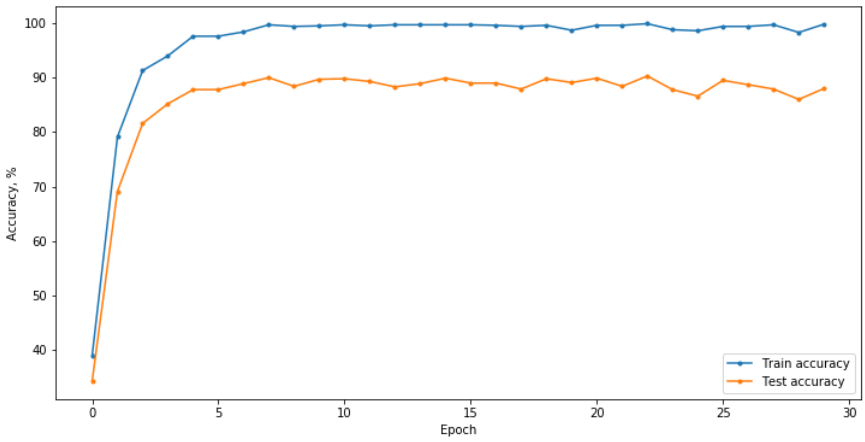
Stride controls how the filter shifts around the image. When stride = 1, means that the filter convolves around the input by shifting one pixel at a time over x-axis and y-axis. Usually stride should be tuned in such a way that the output volume is an integer. For our case, there was used default value of 'stride=1'.

Padding

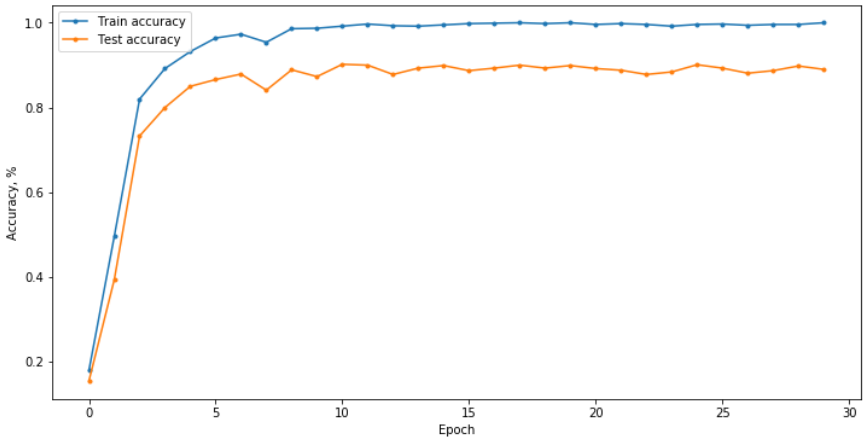
Zero-padding parameter adjusts the input by adding zeros to the edges. It helps to control the spatial size of the output volume.

Below we compare the results with and without zero-padding after executing epoch 30 with default filter size of 5x5, stride = 1, number of filters at Convolution layer 1 = 16, Convolution layer 2 = 32, applied dropouts.

With padding (padding = 'SAME')
Accuracy:
- Training Accuracy: 99.8%
- Test Accuracy: 88.0%
Time usage: 196 seconds



Without padding (padding='VALID')
Accuracy:
- Training Accuracy: 100%
- Test Accuracy: 89.0%
Time usage: 111 seconds



As we can see for our dataset there is no a significant difference using the padding in terms of accuracy, but the computational time increases by around 45% and the accuracy on

test dataset is also higher by 1%. Thus we can omit adding zero-padding for MNIST dataset.

Dropouts

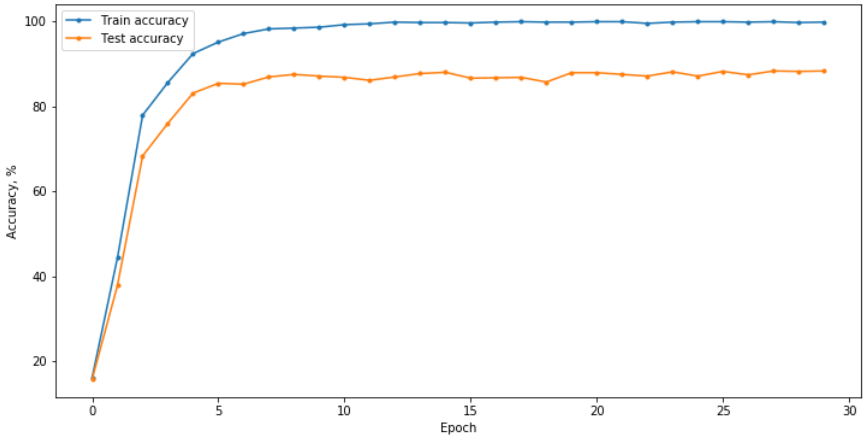
Dropout with probability of 0.6 was used - a technique where randomly selected neurons are ignored during training. This helps to avoid overfitting and is done only on the training step. Neurons are dropped-out randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass. We applied dropout function on the Fully-Connected layer using the Tensorflow function $tf.nn.dropout$.

Rotation

Tensorflow function $tf.contrib.image.rotate$ allows to rotate images at 1-3 degrees clockwise and contrclockwise.
This way we have increased the size of our input data from 1000 to 7000. Below are the results after executing epoch 30 with default filter size of 5x5, stride = 1, number of filters at Convolution layer 1 = 16, Convolution layer 2 = 32, applied dropouts, applied padding.

Accuracy:
- Training Accuracy: 99.8%
- Test Accuracy: 88.3%

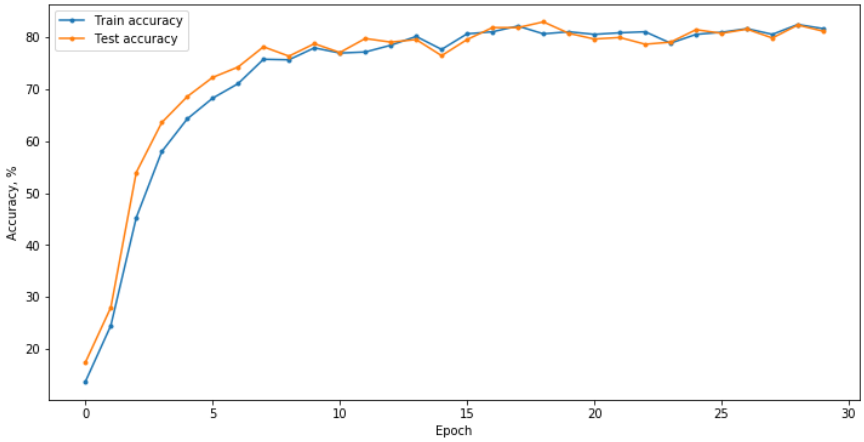
Time usage: 701 seconds



Shifting

We used Tensorflow function $tf.contrib.image.translate$ to shift images per 3 pixels at 8 different direction.
This technique also increases the number of input data, in our case it increased the dataset from 1000 to 9000. Below are the results, settings are the same as with the Rotate state, but the dataset is combined with rotation step, thus there are 15000 samples of training data:

Accuracy:
- Training Accuracy: 81.7%
- Test Accuracy: 81.2%
Time usage: 998 seconds



As we can see from the results rotation and shifting techniques work great with overfitting, for many cases the accuracy for test data was higher than for the train data at the same epoch. But the computation time increased by 4, comparing to using the original dataset, while keeping all other parameters the stable. At the same time the accuracy drops.

Thus we can see that for MNIST dataset the best results in terms of accuracy (89% on testing data) and computational time (111seconds) were shown while training a model without padding and adding modified training data.