German-Russian Institute of Advanced Technologies

TU-Ilmenau (Germany) and KNTRU-KAI (Kazan, Russia)

**Practice 3**

**«Implementing CNN for recognizing of handwritten digits»**

Student: Marushkai N. S.

Lecturer: Makmutova A.Z.

Kazan 2018

During this work, I have searched for the examples and documentation of proper building of CNN in TensorFlow. According to my researches, I found a tutorial from TensorFlow itself, which provides a detailed explanation of how to do such kind of tasks. So, according to this, a model for recognizing handwritten digits was implemented. This model supports 3 modes: training, evaluation and prediction, which are making it’s functions respectively to their names. During implementation of the CNN, there were used such things as:

* Estimators – a TF tool, which provides a high level control of the model, which allows to run it in either training, prediction or evaluation modes.
* I’ve used 2 convolutional (5x5; 32 filters and 5x5 64 filters), 2 pooling, and 1 dense layers for building this model
* The Adam optimizer was used
* LoggingTensorHook was used in order to watch the progress of training on every 50 training steps.

It was suggested by TF tutorial to use 20000 steps to train the model properly, but such training took a lot of time, so I used only 500 steps to train and evaluate the model. And I’ve got the results as following:

{'accuracy': 0.9798, 'loss': 0.06049383, 'global\_step': 500}

According to this results, we can see that accuracy is rather high on such a small amount of steps. So, this approach of using CNN for image recognition is really worth doing it.

Here is the code listing:

**from** \_\_future\_\_ **import** absolute\_import

**from** \_\_future\_\_ **import** division

**from** \_\_future\_\_ **import** print\_function

*# Imports*

**import** numpy **as** np

**import** tensorflow **as** tf

tf.logging.set\_verbosity(tf.logging.INFO)

**def** cnn\_model\_fn(features, labels, mode):

"""Model function for CNN."""

*# Input Layer*

input\_layer = tf.reshape(features["x"], [-1, 28, 28, 1])

*# Convolutional Layer #1*

conv1 = tf.layers.conv2d(

inputs=input\_layer,

filters=32,

kernel\_size=[5, 5],

padding="same",

activation=tf.nn.relu)

*# Pooling Layer #1*

pool1 = tf.layers.max\_pooling2d(inputs=conv1, pool\_size=[2, 2], strides=2)

*# Convolutional Layer #2 and Pooling Layer #2*

conv2 = tf.layers.conv2d(

inputs=pool1,

filters=64,

kernel\_size=[5, 5],

padding="same",

activation=tf.nn.relu)

pool2 = tf.layers.max\_pooling2d(inputs=conv2, pool\_size=[2, 2], strides=2)

*# Dense Layer*

pool2\_flat = tf.reshape(pool2, [-1, 7 \* 7 \* 64])

dense = tf.layers.dense(inputs=pool2\_flat, units=1024, activation=tf.nn.relu)

dropout = tf.layers.dropout(

inputs=dense, rate=0.4, training=mode == tf.estimator.ModeKeys.TRAIN)

*# Logits Layer*

logits = tf.layers.dense(inputs=dropout, units=10)

predictions = {

*# Generate predictions (for PREDICT and EVAL mode)*

"classes": tf.argmax(input=logits, axis=1),

*# Add `softmax\_tensor` to the graph. It is used for PREDICT and by the*

*# `logging\_hook`.*

"probabilities": tf.nn.softmax(logits, name="softmax\_tensor")

}

**if** mode == tf.estimator.ModeKeys.PREDICT:

**return** tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)

*# Calculate Loss (for both TRAIN and EVAL modes)*

loss = tf.losses.sparse\_softmax\_cross\_entropy(labels=labels, logits=logits)

*# Configure the Training Op (for TRAIN mode)*

**if** mode == tf.estimator.ModeKeys.TRAIN:

optimizer = tf.train.AdamOptimizer(learning\_rate=0.001)

train\_op = optimizer.minimize(

loss=loss,

global\_step=tf.train.get\_global\_step())

**return** tf.estimator.EstimatorSpec(mode=mode, loss=loss, train\_op=train\_op)

*# Add evaluation metrics (for EVAL mode)*

eval\_metric\_ops = {

"accuracy": tf.metrics.accuracy(

labels=labels, predictions=predictions["classes"])}

**return** tf.estimator.EstimatorSpec(

mode=mode, loss=loss, eval\_metric\_ops=eval\_metric\_ops)

**def** main(unused\_argv):

*# Load training and eval data*

mnist = tf.contrib.learn.datasets.load\_dataset("mnist")

train\_data = mnist.train.images *# Returns np.array*

train\_labels = np.asarray(mnist.train.labels, dtype=np.int32)

eval\_data = mnist.test.images *# Returns np.array*

eval\_labels = np.asarray(mnist.test.labels, dtype=np.int32)

*# Create the estimator*

mnist\_classifier = tf.estimator.Estimator(

model\_fn=cnn\_model\_fn, model\_dir='/tmp/mnist\_convnet\_model'

)

*# Set up logging*

tensors\_to\_log = {"probabilities": "softmax\_tensor"}

logging\_hook = tf.train.LoggingTensorHook(tensors=tensors\_to\_log, every\_n\_iter=50)

*# Train the model*

train\_input\_fn = tf.estimator.inputs.numpy\_input\_fn(

x={"x": train\_data},

y=train\_labels,

batch\_size=100,

num\_epochs=None,

shuffle=True

)

mnist\_classifier.train(

input\_fn=train\_input\_fn,

steps=500,

hooks=[logging\_hook]

)

*# Evaluate the model*

eval\_input\_fn = tf.estimator.inputs.numpy\_input\_fn(

x={"x": eval\_data},

y=eval\_labels,

num\_epochs=1,

shuffle=False

)

eval\_results = mnist\_classifier.evaluate(input\_fn=eval\_input\_fn)

**print**(eval\_results)

**if** \_\_name\_\_ == "\_\_main\_\_":

tf.app.run()