Authors: Burca Horia, Arias Rodrigo
Date: 10/12/2018

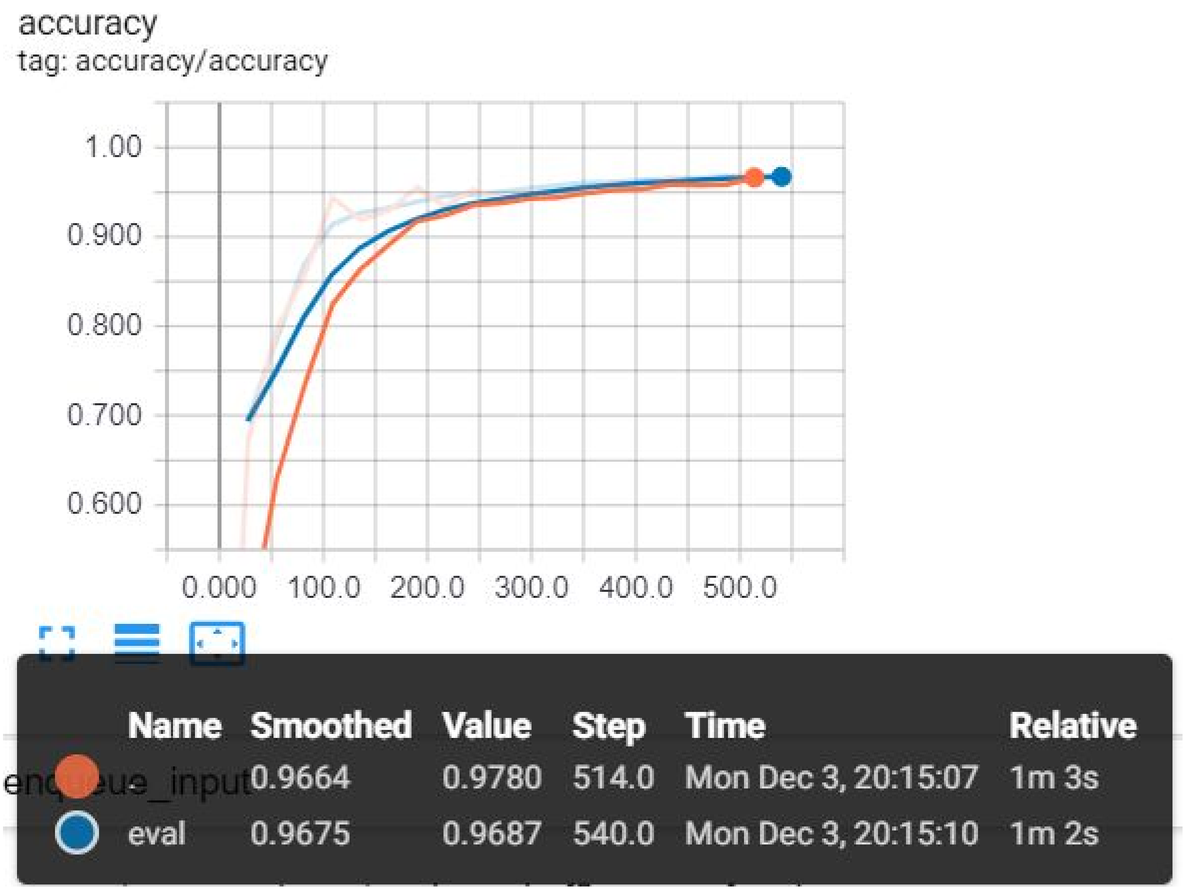
Lab 10: TensorFlow Estimator

Introduction

In this lab we have had the chance to discover the high-level TensorFlow API: Estimator. This API make it easy to quickly configure, train and evaluate a variety of models. The `tf.estimator.Estimator` interface can be customized with a `model_fn` which holds the logic of how the API behaves when training, evaluating and predicting, and an `input_fn` which describes how to handle data. The interface offers the `.train`, `.eval`, `.predict` methods.

Approach

We have used the Google Collaboratory working environment to configure and train our CNN. We worked on the MNIST dataset.
Using the default provided code we obtain an accuracy equal to 96.87%

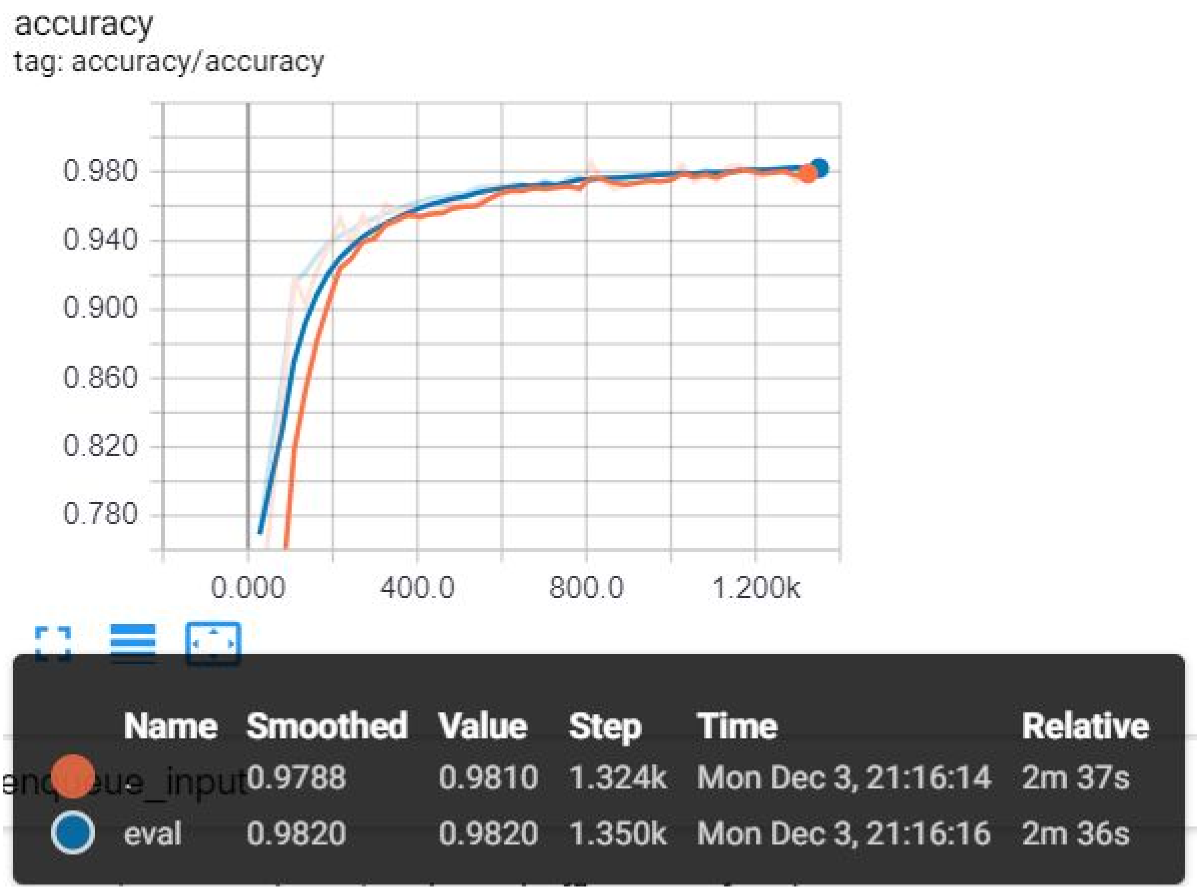


	Name	Smoothed	Value	Step	Time	Relative
●	enqueue_input	0.9664	0.9780	514.0	Mon Dec 3, 20:15:07	1m 3s
○	eval	0.9675	0.9687	540.0	Mon Dec 3, 20:15:10	1m 2s

Figure 1: accuracy using default code

In order to improve the accuracy of the model compared to the initial code that we were provided we can take the following actions:

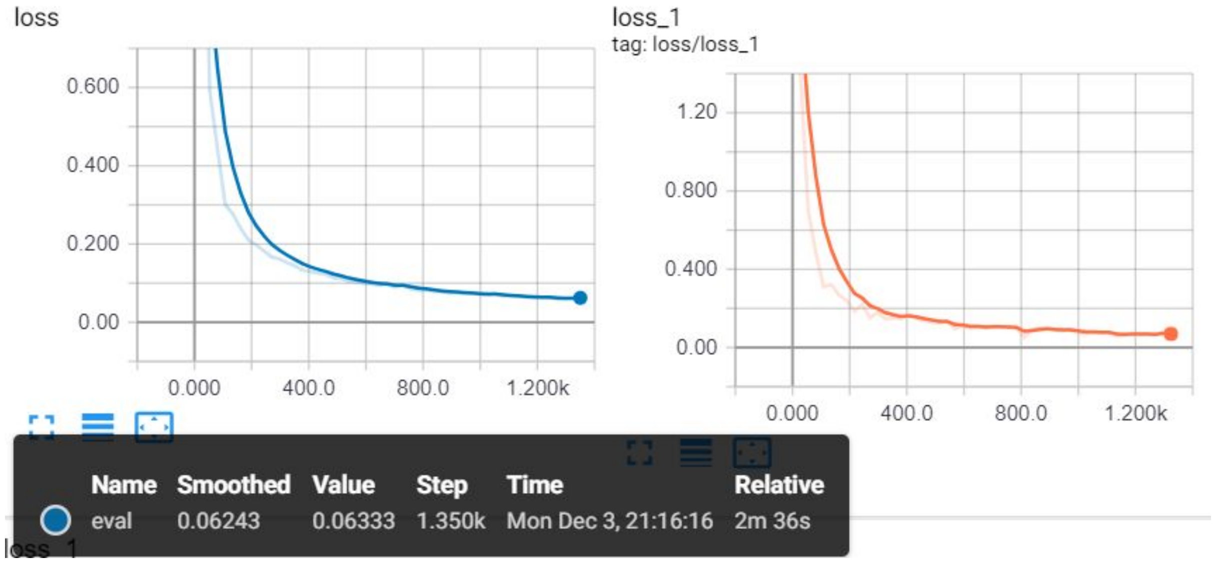
1. increase the number of epochs. We observed that by increasing the number of epochs from 20 to 50 we obtained an accuracy of 98.20%.



	Name	Smoothed	Value	Step	Time	Relative
●	enqueue_input	0.9788	0.9810	1.324k	Mon Dec 3, 21:16:14	2m 37s
○	eval	0.9820	0.9820	1.350k	Mon Dec 3, 21:16:16	2m 36s

Figure 2: accuracy with 50 epochs

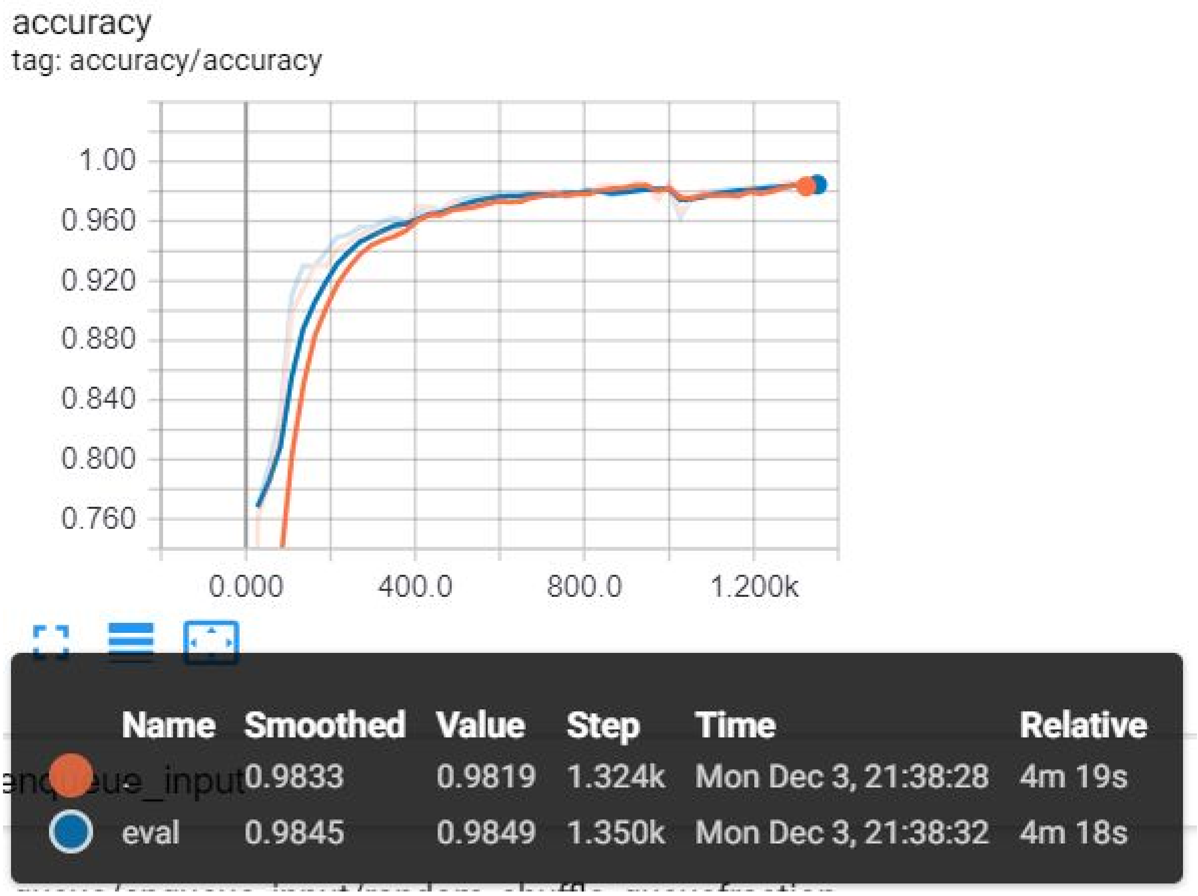
The associated loss is equal to 6.33%.



	Name	Smoothed	Value	Step	Time	Relative
○	eval	0.06243	0.06333	1.350k	Mon Dec 3, 21:16:16	2m 36s

Figure 3: loss with 50 epochs

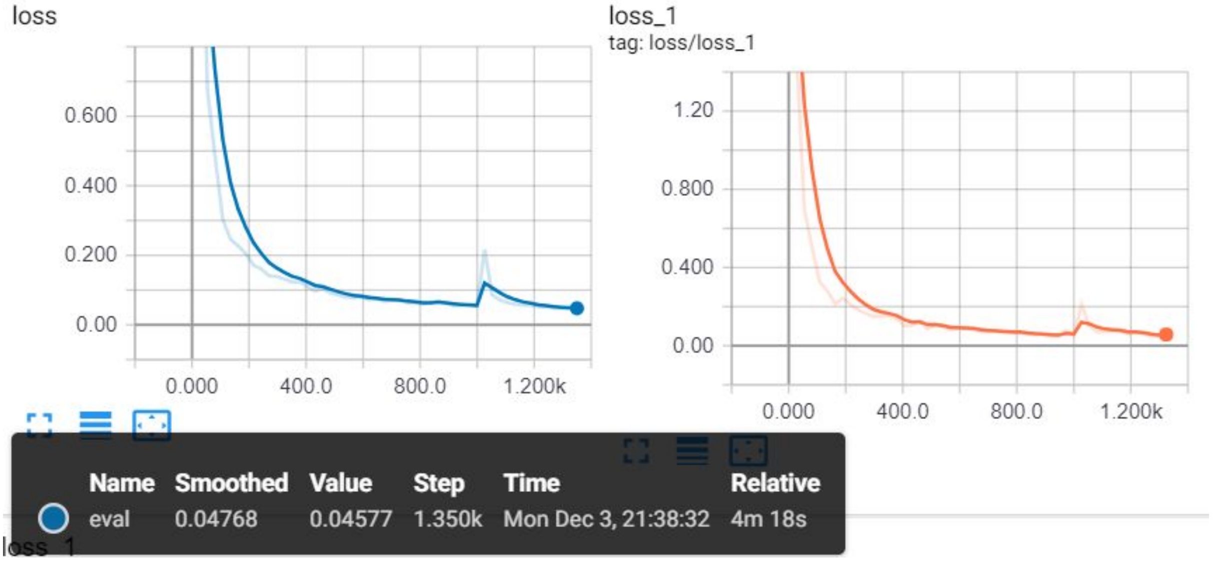
2. add a dropout layer to the model to prevent some overfitting. We observed that by doing this we further increased the accuracy to 98.49%.



	Name	Smoothed	Value	Step	Time	Relative
●	enqueue_input	0.9833	0.9819	1.324k	Mon Dec 3, 21:38:28	4m 19s
●	eval	0.9845	0.9849	1.350k	Mon Dec 3, 21:38:32	4m 18s

Figure 4: accuracy with 50 epochs and improved model

In turn, the associated loss decreased to 4.57%.



	Name	Smoothed	Value	Step	Time	Relative
●	eval	0.04768	0.04577	1.350k	Mon Dec 3, 21:38:32	4m 18s

Figure 4: loss with 50 epochs and improved model

Annex 1: code

```
import os
import sys
import time
import tensorflow as tf
import numpy as np

_NUM_CLASSES = 10
_MODEL_DIR = "model_name"
_NUM_CHANNELS = 1
_IMG_SIZE = 28
_LEARNING_RATE = 0.05
_NUM_EPOCHS = 50
_BATCH_SIZE = 2048

class Model(object):

    def __call__(self, inputs):
        net = tf.layers.conv2d(inputs, 32, [5, 5], padding='same',
                                   activation=tf.nn.relu, name='conv1')
        net = tf.layers.max_pooling2d(net, [2, 2], 2, name='pool1')
        net = tf.layers.conv2d(net, 64, [5, 5], padding='same',
                                   activation=tf.nn.relu, name='conv2')
        net = tf.layers.max_pooling2d(net, [2, 2], 2, name='pool2')

        net = tf.layers.flatten(net)
        net = tf.layers.dense(net, 1024, activation=tf.nn.relu)
        net = tf.layers.dropout(net, rate=0.4)

        logits = tf.layers.dense(net, _NUM_CLASSES, activation=None,
                                     name='fc1')
        return logits

def model_fn(features, labels, mode):

    model = Model()
    global_step=tf.train.get_global_step()

    images = tf.reshape(features, [-1, _IMG_SIZE, _IMG_SIZE,
                                   _NUM_CHANNELS])


    logits = model(images)
```

```
predicted_logit = tf.argmax(input=logits, axis=1,
                            output_type=tf.int32)
probabilities = tf.nn.softmax(logits)


#PREDICT
predictions = {
  "predicted_logit": predicted_logit,
  "probabilities": probabilities
}
if mode == tf.estimator.ModeKeys.PREDICT:
  return tf.estimator.EstimatorSpec(mode=mode,
                                    predictions=predictions)

with tf.name_scope('loss'):
    cross_entropy = tf.losses.sparse_softmax_cross_entropy(
        labels=labels, logits=logits, scope='loss')
    tf.summary.scalar('loss', cross_entropy)

with tf.name_scope('accuracy'):
    accuracy = tf.metrics.accuracy(
        labels=labels, predictions=predicted_logit, name='acc')
    tf.summary.scalar('accuracy', accuracy[1])

#EVAL
if mode == tf.estimator.ModeKeys.EVAL:
    return tf.estimator.EstimatorSpec(
        mode=mode,
        loss=cross_entropy,
        eval_metric_ops={'accuracy/accuracy': accuracy},
        evaluation_hooks=None)


# Create a SGR optimizer
optimizer = tf.train.GradientDescentOptimizer(
            learning_rate=_LEARNING_RATE)
train_op = optimizer.minimize(
            cross_entropy,global_step=global_step)

# Create a hook to print acc, loss & global step every 100 iter.
train_hook_list= []
train_tensors_log = {'accuracy': accuracy[1],
                     'loss': cross_entropy,
                     'global_step': global_step}
train_hook_list.append(tf.train.LoggingTensorHook(
        tensors=train_tensors_log, every_n_iter=100))

    if mode == tf.estimator.ModeKeys.TRAIN:
      return tf.estimator.EstimatorSpec(
          mode=mode,
          loss=cross_entropy,
          train_op=train_op,
          training_hooks=train_hook_list)

def MNIST_classifier_estimator(_):

    # Load training and eval data

    mnist = tf.contrib.learn.datasets.load_dataset('mnist')
    train_data = mnist.train.images  # Returns a np.array
    train_labels = np.asarray(mnist.train.labels, dtype=np.int32)
    eval_data = mnist.test.images  # Returns a np.array
    eval_labels = np.asarray(mnist.test.labels, dtype=np.int32)

    # Create a input function to train

    train_input_fn = tf.estimator.inputs.numpy_input_fn(x=train_data,
            y=train_labels, batch_size=_BATCH_SIZE, num_epochs=1,
            shuffle=True)

    # Create a input function to eval

    eval_input_fn = tf.estimator.inputs.numpy_input_fn(x=eval_data,
            y=eval_labels, batch_size=_BATCH_SIZE, num_epochs=1,
            shuffle=False)

    # Create a estimator with model_fn

    image_classifier = tf.estimator.Estimator(model_fn=model_fn,
            model_dir=_MODEL_DIR)

    # Finally, train and evaluate the model after each epoch

    for _ in range(_NUM_EPOCHS):
        image_classifier.train(input_fn=train_input_fn)
        metrics = image_classifier.evaluate(input_fn=eval_input_fn)

if __name__ == '__main__':

    tf.logging.set_verbosity(tf.logging.INFO)
    tf.app.run(MNIST_classifier_estimator)

!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
!unzip ngrok-stable-linux-amd64.zip
get_ipython().system_raw(
    'tensorboard --logdir {} --host 0.0.0.0 --port 6006 &'
    .format(_MODEL_DIR)
)
get_ipython().system_raw('./ngrok http 6006 &')
!curl -s http://localhost:4040/api/tunnels
```