به نام خدا

درس: یادگیری عمیق

استاد: دکتر فاطمی‌زاده

گزارش تمرین عملی شماره ۴

سیّدمحمّدامین منصوری طهرانی

۹۴۱۰۵۱۷۴

توجه: فایل‌های event نیز به همراه گزارش و کد مسأله ۱ و ۴ پیوست شده‌اند.

۱. به طور خلاصه کد پیوست شده در فایل Deep_Learning_HW4.py را توضیح می‌دهم و نتایج پس از توضیحات قابل مشاهده‌اند.

```
# MNIST data input (img shape: 28*28)
n_input = 28

# MNIST total classes (0-9 digits)
n_classes = 10
n_classes_p4 = 2
std = 0.1

weights = {
    'weight_conv_layer_1': tf.get_variable('W0', shape=(5, 5, 1, 64),
                                           initializer=tf.initializers.random_normal(stddev=std)
                                           ),
    'weight_conv_layer_2': tf.get_variable('W1', shape=(5, 5, 64, 64),
                                           initializer=tf.initializers.random_normal(stddev=std)),
    # 'weight_conv_layer_3': tf.get_variable('W2', shape=(3, 3, 64, 128),
    # initializer=tf.contrib.layers.xavier_initializer()),
    'w_fc': tf.get_variable('W2', shape=(7 * 7 * 64, 256), initializer=tf.initializers.random_normal(stddev=std)),
    'out': tf.get_variable('W3', shape=(256, n_classes), initializer=tf.initializers.random_normal(stddev=std)),
    'out_p4': tf.get_variable('W4', shape=(256, n_classes_p4), initializer=tf.initializers.random_normal(stddev=std)),
}
biases = {
    'bias_conv_layer_1': tf.get_variable('B0', shape=64, initializer=tf.initializers.random_normal(stddev=std)),
    'bias_conv_layer_2': tf.get_variable('B1', shape=64, initializer=tf.initializers.random_normal(stddev=std)),
    # 'bias_conv_layer_3': tf.get_variable('B2', shape=128, initializer=tf.contrib.layers.xavier_initializer()),
    'b_fc': tf.get_variable('B2', shape=256, initializer=tf.initializers.random_normal(stddev=std)),
    'out': tf.get_variable('B3', shape=n_classes, initializer=tf.initializers.random_normal(stddev=std)),
    'out_p4': tf.get_variable('B4', shape=n_classes_p4, initializer=tf.initializers.random_normal(stddev=std)),
}
```

در قطعه فوق، پس از import کتاب‌خانه‌های مربوط، تعداد پیکسل‌های هر سطر و ستون و تعداد کلاس‌های سوال اول که ۱۰ می‌باشد به همراه تعداد کلاس‌های مسأله ۴ که ۲ می‌باشد تعریف می‌شوند. (دومی برای استفاده از restore استفاده می‌شود.) std نیز انحراف معیار وزن‌ها و بایاس‌ها در مقدار گاوسی آن‌ها برای initialize می‌باشد و برای خوب نتیجه گرفتن باید همین حدود تنظیم شود. در ادامه نیز وزن لایه‌های کانولوشنی و fully connected و لایه fc به لایه طبقه‌بند به همراه بایاس آن‌ها در دیکشنری تعریف می‌شوند تا دسترسی به آن‌ها تسهیل گردد.

```python
def conv2d(x, w, b, strides=1):
    # Conv2D wrapper, with bias and ReLU activation
    x = tf.nn.conv2d(x, w, strides=[1, strides, strides, 1], padding='SAME')
    x = tf.nn.bias_add(x, b)
    return tf.nn.relu(x)


def maxpool2d(x, k=2):
    return tf.nn.max_pool(x, ksize=[1, k, k, 1], strides=[1, k, k, 1], padding='SAME')


# both placeholders are of type float
x = tf.placeholder("float", [None, 28, 28, 1])
# x = tf.placeholder("float", [28, 28])
y = tf.placeholder("float", [None, n_classes])
y_trans_learn = tf.placeholder("float", [None, n_classes_p4])

# here we call the conv2d function we had defined above and pass the input image x, weights weight_conv_layer_1
# and bias bias_conv_layer_1.
conv1 = conv2d(x, weights['weight_conv_layer_1'], biases['bias_conv_layer_1'])
# Max Pooling (down-sampling), this chooses the max value from a 2*2 matrix window and outputs a 14*14 matrix.
conv1_maxpool = maxpool2d(conv1, k=2)

# print(x.shape, weights['weight_conv_layer_1'].shape, biases['bias_conv_layer_1'].shape, conv1.shape)

# Convolution Layer
# here we call the conv2d function we had defined above and pass the input image x, weights wc2 and bias bc2.
conv2 = conv2d(conv1_maxpool, weights['weight_conv_layer_2'], biases['bias_conv_layer_2'])
# Max Pooling (down-sampling), this chooses the max value from a 2*2 matrix window and outputs a 7*7 matrix.
conv2_maxpool = maxpool2d(conv2, k=2)
```

در قطعه کد فوق ابتدا عملیات کانولوشن دوبعدی به وسیله یک تابع تعریف شده و از تابع فعالیت relu عبور می‌کند و به عنوان خروجی بازگردانده می‌شود. لایه downsampling نیز به سادگی از نوع maxpool تعریف می‌شود. در ادامه placeholder هایی برای استفاده در شبکه عصبی تعریف می‌شود و آخرین آن‌ها برای تولید ساختار مسأله ۴ است و بعداً در آن مسأله استفاده خواهد شد.

در ادامه عملیات کانولوشن دو بعدی و اعمال فیلترها و maxpooling ها به سادگی انجام می‌گردند.

```
# Fully connected layer
# Reshape conv2 output to fit fully connected layer input
fc1 = tf.reshape(conv2_maxpool, [-1, weights['w_fc'].get_shape().as_list()[0]])
fc1 = tf.add(tf.matmul(fc1, weights['w_fc']), biases['b_fc'])
fc1 = tf.nn.relu(fc1)
# Output, class prediction
# finally we multiply the fully connected layer with the weights and add a bias term.
out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])


training_iterations = 10
learning_rate = 0.01
batch_size = 128
keep_prob = tf.placeholder("float")

data = input_data.read_data_sets('mnist', one_hot=True)

print("Training set (images) shape: {shape}".format(shape=data.train.images.shape))

curr_img = np.reshape(data.train.images[1], (28, 28))
curr_lbl = np.argmax(data.train.labels[1, :])
plt.imshow(curr_img)
plt.show()

# Reshape training and testing image
train_X = data.train.images.reshape(-1, 28, 28, 1)
test_X = data.test.images.reshape(-1, 28, 28, 1)

print([train_X.shape, test_X.shape])

train_y = data.train.labels
test_y = data.test.labels

print(train_y.shape, test_y.shape)

prediction = out

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=prediction, labels=y))
```

عملیات کانولوشن در لایه تمام‌متصل نیز انجام پذیرفته و از تابع فعالیت relu می‌گذرد و نهایتاً با ضرب شدن در وزن‌های لایه طبقه‌بند و جمع شدن با بایاس آن‌ها خروجی ۱۰ نورون نهایی تعیین می‌شوند.

تعداد تکرار یادگیری ۱۰ بار تعیین شده و سایز دسته‌ها ۱۲۸ قرار داده شده‌است. نرخ یادگیری نیز برابر ۰٫۰۱ تنظیم می‌شود. یک متغیر نیز برای احتمال نگه داشتن هر نورون از جنس float تعریف می‌کنیم. در یادگیری ما، از هر ۵۵۰۰۰ عکس train، هر بار یک دهم آن‌ها یعنی ۵۵۰۰ عکس به ترتیب انتخاب شده و با بچ‌های ۱۲۸ تایی به طور کامل ترین می‌شود و سپس برای ۵۵۰۰ داده بعدی این کار انجام می‌شود تا انتها که از همه داده‌های ترین استفاده شود.(که طبیعتاً overfitting روی خطای train اتفاق می‌افتد که برای مهم نیست چون داده‌های تست fresh در اختیار داریم که مدل را با آن‌ها validate کنیم.)

سپس برای اطمینان از شرایط شکل(shape) داده ترین چاپ می‌شود و یک عکس هم از داده‌های ترین به عنوان نمونه چاپ می‌شود. سپس داده‌های ترین و تست reshape می‌شوند تا به صورت بچ قابل انتقال به شبکه کانولوشن شوند. برچسب آن‌ها نیز جدا شده و در متغیر ذخیره می‌شود. در آخر تابع هزینه بر اساس cross entropy بین پیش‌بینی خروجی ۱۰ نورون انتهایی و برچسب‌های واقعی تعیین می‌شود.

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)

# Here you check whether the index of the maximum value of the predicted image is equal to the actual labelled image.
#  and both will be a column vector.
correct_prediction = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))

# calculate accuracy across all the given images and average them out.
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# summary settings:

train_loss_summary = tf.summary.scalar('loss_train', cost)
train_accuracy_summary = tf.summary.scalar('train_accuracy', accuracy)
train_summaries = tf.summary.merge([train_loss_summary, train_accuracy_summary])
# file_writer = tf.summary.FileWriter('./Output', sess.graph)

validation_loss_summary = tf.summary.scalar('loss_validation', cost)
validation_accuracy_summary = tf.summary.scalar('validation_accuracy', accuracy)
validation_summaries = tf.summary.merge([validation_loss_summary, validation_accuracy_summary])
# validation_file_writer = tf.summary.FileWriter(output_folder)

# Initializing the variables
init = tf.global_variables_initializer()

saver = tf.train.Saver()
```

در ابتدای این قطعه بهینه‌ساز از نوع Gradient Descent با نرخ یادگیری تعیین شده ساخته می‌شود و وظیفه آن‌را مینیمم کردن تابع cost قرار می‌دهیم. سپس پیش‌بینی‌های درست با tf.equal شناخته شده و دقت مدل با درصد پیش‌بینی صحیح به وسیله میانگین correct prediction تعیین می‌شود چون این متغیر به تعداد پیش‌بینی صحیح مقدار ۱ دارد و سایر آن‌ها صفر است و پس میانگین آن همان دقت است.

در ادامه نیز شیءهای summary برای استفاده در تنسوربرد تعیین می‌شوند. برای اعتبارسنجی و آموزش summary جدا تعیین می‌کنیم.

نهایتاً initialize می‌کنیم و saver را برای ذخیره مدل و استفاده بعدی تعریف می‌کنیم.

```
# Batch Mode Learning

with tf.Session() as sess:
    sess.run(init)

    merge = tf.summary.merge_all()
    summary_writer_train = tf.summary.FileWriter('./Output_train', sess.graph)
    summary_writer_validation = tf.summary.FileWriter('./Output_validation', sess.graph)
    summary_writer_conv_layer_1 = tf.summary.FileWriter('./conv_1_filters')
    summary_writer_conv_layer_2 = tf.summary.FileWriter('./conv_2_filters')

    for i in range(training_iterations):
        for batch in tqdm(range(len(train_X) // batch_size // training_iterations)):
            batch_x = train_X[(batch + i * len(train_X) // batch_size // training_iterations) *
                              batch_size:min((batch + 1 + i * len(train_X) // batch_size // training_iterations) *
                                             batch_size, len(train_X))]
            batch_y = train_y[(batch + i * len(train_X) // batch_size // training_iterations) *
                              batch_size:min((batch + 1 + i * len(train_X) // batch_size // training_iterations) *
                                             batch_size, len(train_y))]
            # Run optimization op (backprop).
            # Calculate batch loss and accuracy
            opt = sess.run(optimizer, feed_dict={x: batch_x,
                                                 y: batch_y})
            loss, acc = sess.run([cost, accuracy], feed_dict={x: batch_x,
                                                              y: batch_y})

            summary_writer_train.add_summary(
                (sess.run(train_summaries, feed_dict={x: batch_x,
                                                      y: batch_y})), batch)

            print("Iter " + str(i) + ", Loss= " + "{:.6f}".format(loss) + ", Training Accuracy= " + "{:.5f}".format(acc)
                  )
            print("Optimization Finished!")

        # Calculate accuracy for all 10000 mnist test images
        test_acc, valid_loss = sess.run([accuracy, cost], feed_dict={x: test_X, y: test_y})
```

در این قسمت session تعریف می‌شود و initialization انجام می‌شود. سپس فولدرهای summary writer ها ساخته می‌شود تا فایل‌های event که متغیرهای آن‌ها را نگه می‌دارند در آن‌ها ذخیره شوند. سپس در حلقه‌ای به تعداد دفعات تکرار، هر بار کل داده‌های عکس‌ها یا همان ۵۵۰۰۰ عکس، به تعداد دفعات تکرار تقسیم شده (در مورد کد من به ۵۵۰۰ عکس در هر iteration) و با بچ‌های با اندازه ۱۲۸ تایی آموزش داده می‌شود و دفعه بعد ۵۵۰۰ عکس بعدی از ۵۵۰۰۰ عکس کل با بچ‌های ۱۲۸ تایی آموزش داده می‌شود. در ادامه تعریف بچ‌های هر به روز رسانی، بهینه‌ساز اجرا شده و بعد از آن هزینه و دقت به ازای این بچ با مدلی که تا به این لحظه به دست آمده محاسبه می‌شوند. این مقادیر به summary مربوط به train اضافه می‌شوند و دقت مدل ساخته شده تا این لحظه بر روی این بچ چاپ می‌شود. در انتهای هر تکرار نیز، مدلی که تا این لحظه ساخته شده با یک بچ از داده‌های test، اعتبارسنجی می‌شود و دقت و مقدار هزینه برای آن در validation summary ذخیره می‌شود.

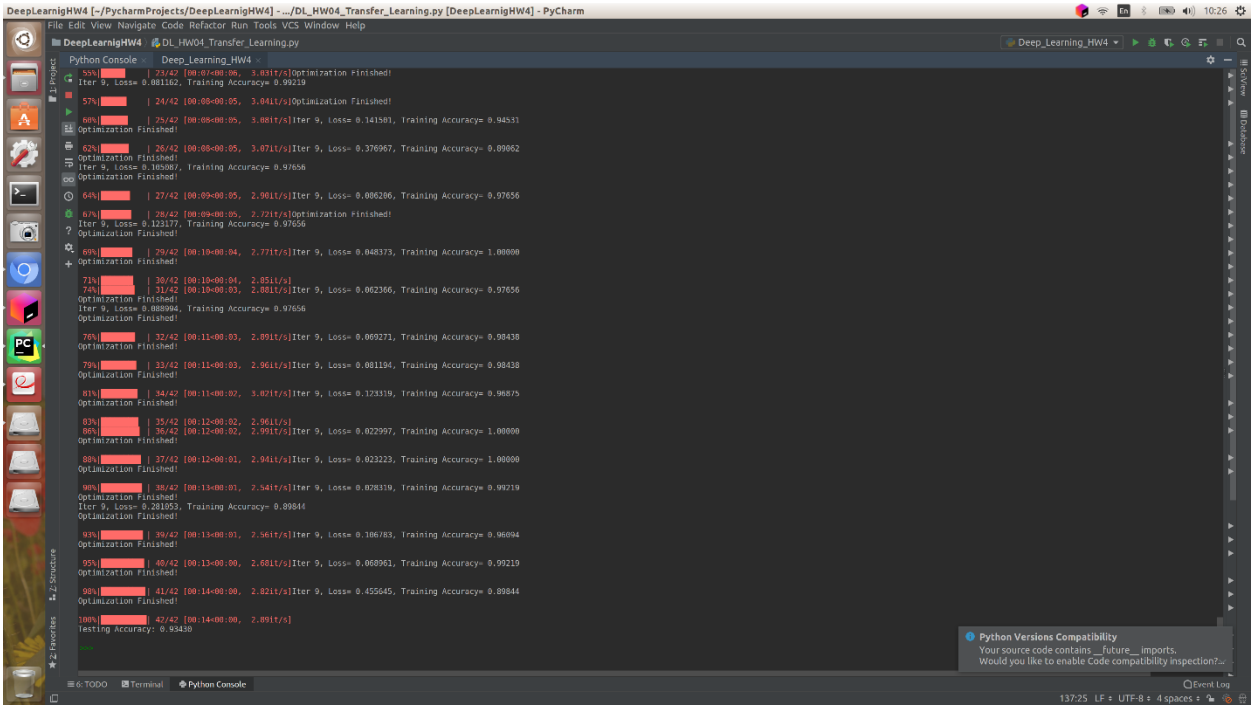
```
    test_acc, valid_loss = sess.run([accuracy, cost], feed_dict={x: test_X, y: test_y})

    summary_writer_validation.add_summary(
        (sess.run(validation_summaries, feed_dict={x: test_X,
                                                   y: test_y})), i)

    print("Testing Accuracy:", "{:.5f}".format(test_acc))

    # summary_writer_conv_layer_1.add_summary(sess.run(filter_summary_conv_1, feed_dict={x: test_X,
    #                                                                                    y: test_y}), i)
    # summary_writer_conv_layer_2.add_summary(sess.run(filter_summary_conv_2, feed_dict={x: test_X,
    #                                                                                    y: test_y}), i)

save_path = saver.save(sess, './tmp/model.ckpt')

summary_writer_train.close()
summary_writer_validation.close()

# Plotting first layer's weights

weights_image_conv_1 = np.reshape(sess.run(weights['weight_conv_layer_1']).T, newshape=(64, 5, 5))

fig_1 = plt.figure()
plt.figure(num=None, figsize=(30, 30), dpi=100)
plt.title('First Convolution Layer Weights')

for j in np.arange(0, 64):
    plt.subplot(8, 8, j + 1)
    plt.imshow((weights_image_conv_1[j][:][:]).T)

plt.show()
```

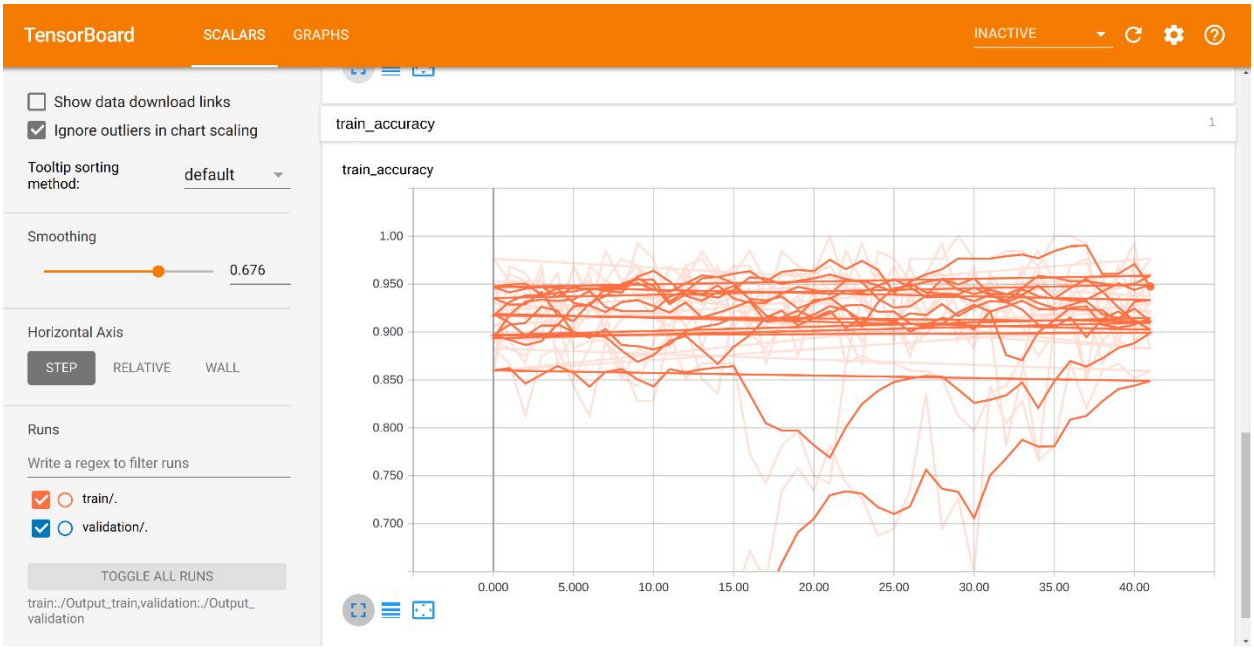
سپس مدل آموزش داده شده در پایان آخرین تکرار save می‌شود تا برای مسأله ۴ قابل استفاده باشد(transfer learning).

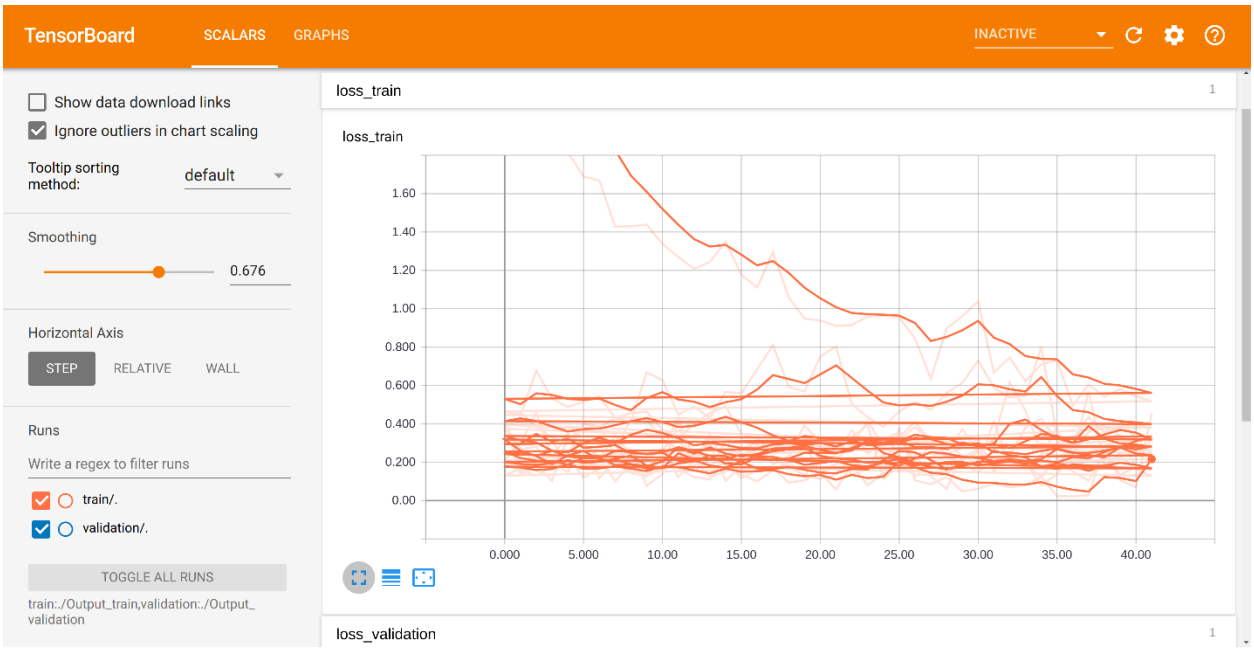
بعد از آن هم writer ها بسته می‌شوند. ادامه کد به سوال ۲ و ۳ مربوط است و توضیح آن‌ها را در سوال خودشان می‌آورم.

نتیجه دقت و هزینه برای داده‌های آموزش و اعتبارسنجی:

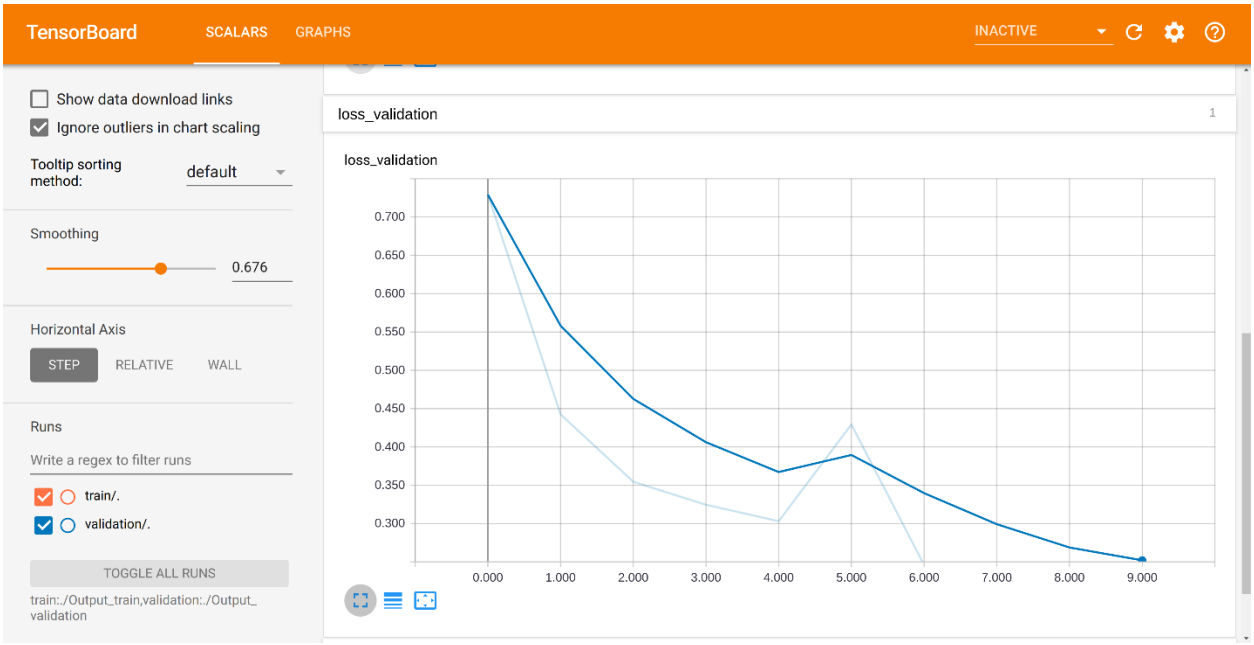
مشاهده می‌شود دقت اعتبارسنجی ۹۳,۴ درصد است.


DeepLearnigHW4 [~/PycharmProjects/DeepLearnigHW4] - .../DL_HW04_Transfer_Learning.py [DeepLearnigHW4] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help
DeepLearnigHW4 DL_HW04_Transfer_Learning.py
Deep_Learning_HW4
Python Console
Deep_Learning_HW4
Optimization Finished!
Iter 9, Loss= 0.081162, Training Accuracy= 0.99219
Optimization Finished!
Iter 9, Loss= 0.141581, Training Accuracy= 0.94531
Optimization Finished!
Iter 9, Loss= 0.376967, Training Accuracy= 0.89062
Optimization Finished!
Iter 9, Loss= 0.165087, Training Accuracy= 0.97656
Optimization Finished!
Optimization Finished!
Iter 9, Loss= 0.123177, Training Accuracy= 0.97656
Optimization Finished!
Optimization Finished!
Optimization Finished!
Optimization Finished!
Iter 9, Loss= 0.088994, Training Accuracy= 0.97656
Optimization Finished!
Optimization Finished!
Optimization Finished!
Optimization Finished!
Optimization Finished!
Optimization Finished!
Optimization Finished!
Optimization Finished!
Iter 9, Loss= 0.281053, Training Accuracy= 0.89844
Optimization Finished!
Optimization Finished!
Optimization Finished!
Testing Accuracy: 0.93438
Python Versions Compatibility
Your source code contains __future__ imports.
Would you like to enable Code compatibility inspection?...
6: TODO
Terminal
Python Console
Event Log
137:25 LF UTF-8 4 spaces


TensorBoard
SCALARS
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method:
default
Smoothing
0.676
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
train/.
validation/.
TOGGLE ALL RUNS
train:./Output_train,validation:./Output_validation
train_accuracy
1
train_accuracy
1.00
0.950
0.900
0.850
0.800
0.750
0.700
0.000
5.000
10.00
15.00
20.00
25.00
30.00
35.00
40.00


TensorBoard
SCALARS
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method:
default
Smoothing
0.676
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
train/.
validation/.
TOGGLE ALL RUNS
train:./Output_train,validation:./Output_validation
loss_train
1
loss_train
1.60
1.40
1.20
1.00
0.800
0.600
0.400
0.200
0.00
0.000
5.000
10.00
15.00
20.00
25.00
30.00
35.00
40.00
loss_validation
1


TensorBoard
SCALARS
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method:
default
Smoothing
0.676
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
train/.
validation/.
TOGGLE ALL RUNS
train:./Output_train,validation:./Output_validation
validation_accuracy
1
validation_accuracy
0.940
0.920
0.900
0.880
0.860
0.840
0.820
0.800
0.780
0.000
1.000
2.000
3.000
4.000
5.000
6.000
7.000
8.000
9.000

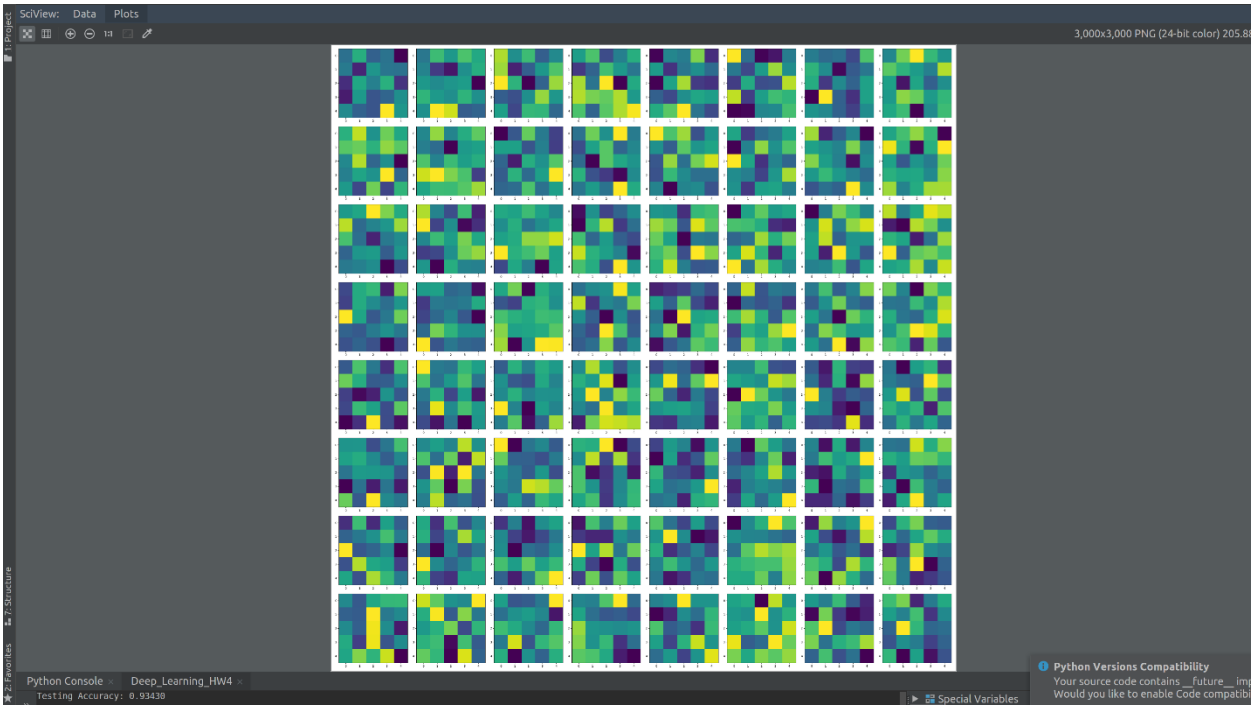


۲. ابتدا قطعه کد مربوط به آن را به طور خلاصه توضیح می‌دهیم.

```
# Plotting first layer's weights

weights_image_conv_1 = np.reshape(sess.run(weights['weight_conv_layer_1']).T, newshape=(64, 5, 5))

fig_1 = plt.figure()
plt.figure(num=None, figsize=(30, 30), dpi=100)
plt.title('First Convolution Layer Weights')

for j in np.arange(0, 64):
    plt.subplot(8, 8, j + 1)
    plt.imshow((weights_image_conv_1[j][:][:]).T)

plt.show()
```

وزن‌ها با دستور اجرا به ماتریس تبدیل شده و ترانهاده شده و از ۱ در ۶۴ در ۵ در ۵ به شکل ۶۴ فیلتر ۵ در ۵ در می‌آیند و در یک subplot نمایش داده می‌شوند. خروجی وزن‌های یادگرفته شده پس تکرارها در شکل زیر مشاهده می‌شود.

در وزن‌ها الگوی خاصی مشاهده نمی‌شود به جز این‌که در بعضی از آن‌ها نوعی کاهش اندازه پیکسل‌ها در یک جهت مشاهده می‌شود که این مشتق‌گیری جهت‌دار را بیان می‌کند و طبیعی است که درتشخیص و دسته‌بندی کلاس‌های عددی به مشتق‌های جهتی مختلف نیاز داشته باشیم. در تعداد اندکی از فیلترها نیز میانگین حول یک منطقه خاصی دیده می‌شود و ممکن است اثر blur کردن برای تشخیص بعضی اعداد مورد نیاز بوده باشد.



۳. توضیح مختصر کد این قسمت:

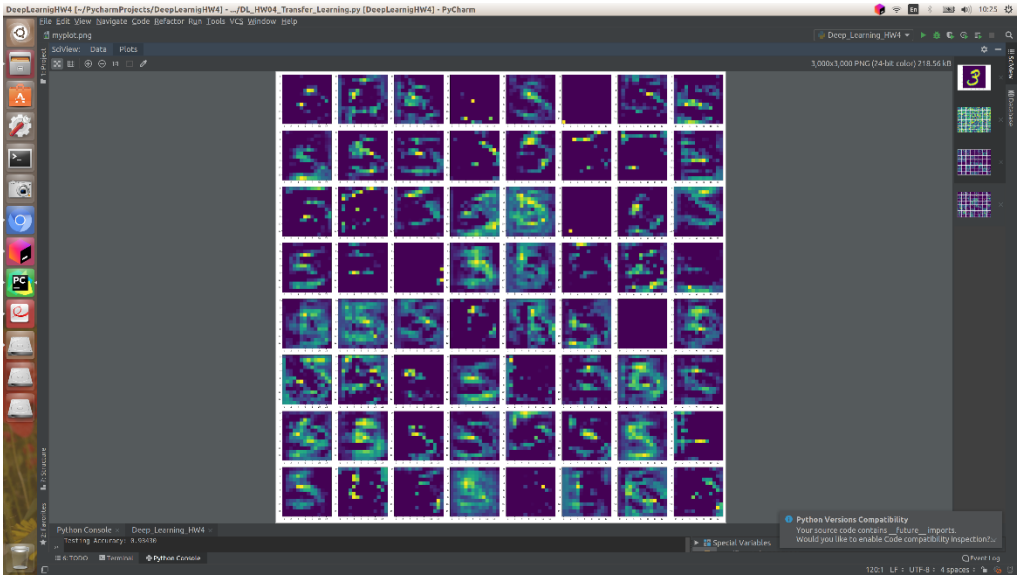
```
# Plotting first cnn layer's output for class '5'

feed_x = test_X[15]
feed_x = feed_x.reshape(-1, 28, 28, 1)
first_layer_output_class_5 = np.reshape(sess.run(conv1, feed_dict={x: feed_x}).T, newshape=(64, 28, 28))

fig_2 = plt.figure()
plt.figure(num=None, figsize=(30, 30), dpi=100)
plt.title('First Convolution Layer Outputs')

for j in np.arange(0, 64):
    plt.subplot(8, 8, j + 1)
    plt.imshow((first_layer_output_class_5[j][:][:]).T)

plt.show()

# Plotting second cnn layer's output for class '5'

second_layer_output_class_5 = np.reshape(sess.run(conv2, feed_dict={x: feed_x}).T, newshape=(64, 14, 14))

fig_3 = plt.figure()
plt.figure(num=None, figsize=(30, 30), dpi=100)
plt.title('Second Convolution Layer Outputs')

for j in np.arange(0, 64):
    plt.subplot(8, 8, j + 1)
    plt.imshow((second_layer_output_class_5[j][:][:]).T)

plt.show()
```

یکی از داده‌های تست که مربوط به ۵ است را انتخاب می‌کنیم و شکل آن را مناسب ورود به شبکه کانولوشن می‌کنیم سپس با اجرا کردن کانولوشن لایه اول، شکل خروجی آن‌را تغییر می‌دهیم تا مناسب رسم شود و در

subplot ای آن را نمایش می‌دهیم. پس از آن نیز با اجرای کانولوشن لایه دوم به ازای همان خروجی و تغییر شکل آن، خروجی آن را در یک subplot دیگر رسم می‌کنیم.

خروجی لایه اول کانولوشن:

خروجی لایه دوم کانولوشن:



در تصاویر بالا مشخص است که مطابق حدس قسمت قبل، در بعضی تصاویر خروجی لایه اول مرزها روشن شده و مشخص‌اند(مشتق‌گیرها) و بعضی دیگر داخل مرزها را پررنگ‌تر کرده‌اند و اثر خارج آن را کمتر کرده‌اند. آن‌چه مشخص است این است که اکثر فیلترها خروجی‌شان به ازای ورودی ۵ کاملاً به شکل آن مرتبط است و می‌توان این را نوعی تشخیص در نظر گرفت و نکته جالب توجه دیگر این‌که لایه اول شکل ۵ و ویژگی‌های ظاهری آن را به خوبی تشخیص داده حال آن‌که لایه دوم تصاویر چندان مرتبطی با ۵ ندارد و می‌توان حدس زد که ویژگی‌های پیچیده‌تری که چشم ما برای دسته‌بندی از آن استفاده نمی‌کند (از آن‌جا که ما این تصاویر را نمی‌فهمیم!) در این فیلترها encode شده‌است. البته خروجی بعضی از این فیلترها شبیه ۵ چاق شده است و بیان‌گر ویژگی‌های سطح پایینی مثل مشتق و moving average است.

۴. این قسمت (که یافتن روش دقیق انجام آن سخت هم بود!) از یادگیری انتقالی استفاده می‌کند و از مدل مسأله ۱ برای آموزش دسته‌بند دوتایی بین کلاس ۱ و ۴ بهره می‌گیرد. فقط وزن‌های لایه fully connected و وزن این لایه به طبقه‌بند دو نورونی بهینه‌سازی می‌شود و وزن لایه‌های قبل از یادگیری با حجم زیادی داده جایگزین می‌شود.

برای انجام این کار ابتدا در دیکشنری وزن‌ها و بایاس‌ها دو وزن بدون استفاده برای لایه fc به لایه دو نورونی تعریف و initialize کردیم که در این مدل استفاده شود. هم‌چنین placeholder خروجی این دو نورون را نیز تعریف کردیم. در آخرین تکرار یادگیری مسأله یک نیز مدل را save کردیم.

```
import urllib.request


class AppURLopener(urllib.request.FancyURLopener):
    version = "Mozilla/5.0"


opener = AppURLopener()
response = opener.open('http://httpbin.org/user-agent')
```

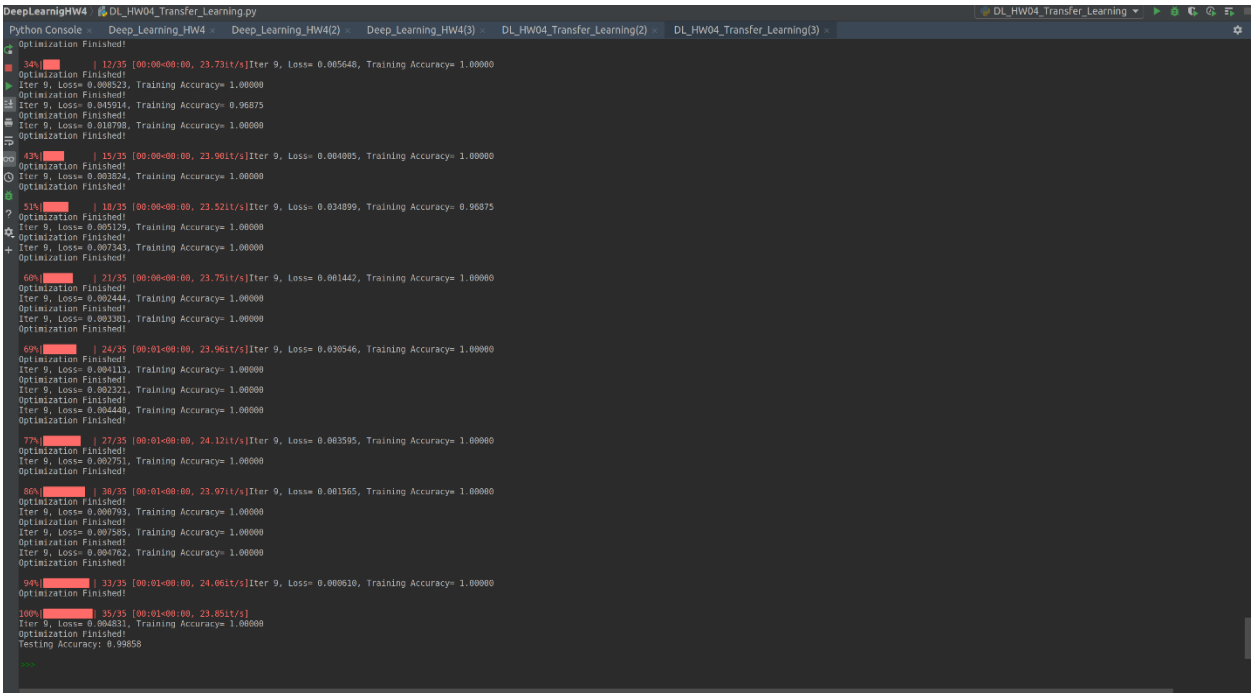
در این قسمت به علت وجود یک خطای HTTP و جستجو در گوگل، این قطعه‌کدها قرار داده شدند.

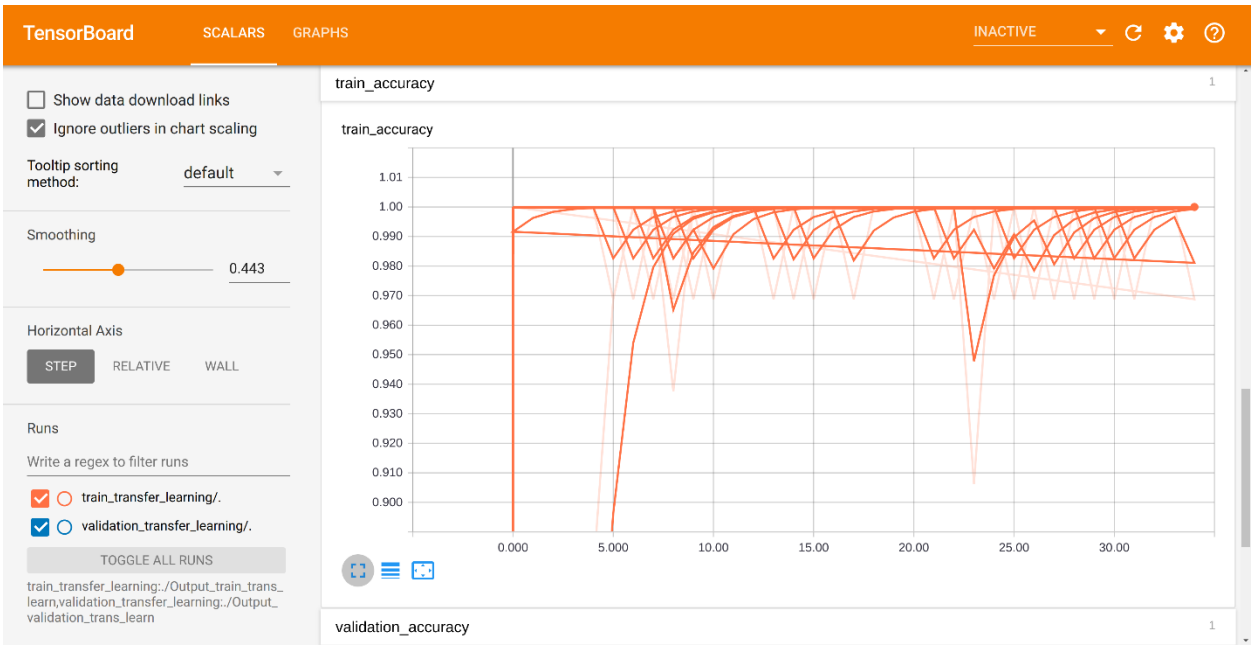
در ادامه همه کارهایی که در مسأله ۱ انجام شد، با تغییر نام متغیرها انجام شد(اضافه کردن trans_learning_ به انتهای هر متغیر!) هم‌چنین در ابتدای آموزش session جدید، مدل restore شد و برای به روز رسانی مدل با استفاده از پارامتر varlist در بهینه‌ساز، فقط وزن و بایاس‌های دو لایه آخر را قابل تغییر کردن قرار دادم. هم‌چنین چون تعداد داده‌های این دو کلاس خیلی کمتر از کل داده‌هاست باید اندازه بچ نیز کاهش یابد و آن را برابر ۳۲ قرار دادم.

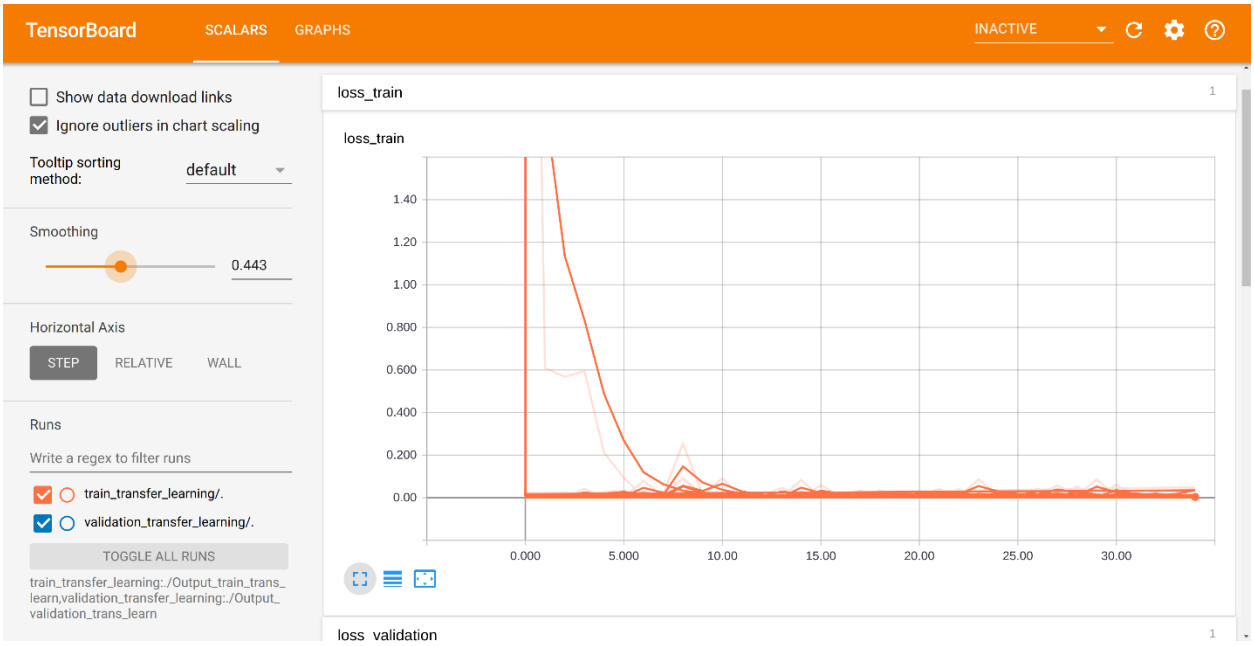
```
X_train = np.array([[]])
Y_train = np.array([[]])

t = 0

for i in range(0, len(data.train.images)):
    r = data.train.labels[i]
    if np.all(r == np.array([0, 1, 0, 0, 0, 0, 0, 0, 0, 0])) or np.all(r == np.array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0])):
        t = t + 1
        if np.all(X_train == np.array([[]])):
            X_train = [data.train.images[i]]
            Y_train = [data.train.labels[i, [1, 4]]]
        else:
            X_train = np.append(X_train, [data.train.images[i]], axis=0)
            Y_train = np.append(Y_train, [data.train.labels[i, [1, 4]]], axis=0)

X_test = np.array([[]])
Y_test = np.array([[]])

for i in range(0, len(data.test.images)):
    r = data.test.labels[i]
    if np.all(r == np.array([0, 1, 0, 0, 0, 0, 0, 0, 0, 0])) or np.all(r == np.array([0, 0, 0, 0, 1, 0, 0, 0, 0, 0])):
        if np.all(X_test == np.array([[]])):
            X_test = [data.test.images[i]]
            Y_test = [data.test.labels[i, [1, 4]]]
        else:
            X_test = np.append(X_test, [data.test.images[i]], axis=0)
            Y_test = np.append(Y_test, [data.test.labels[i, [1, 4]]], axis=0)
```

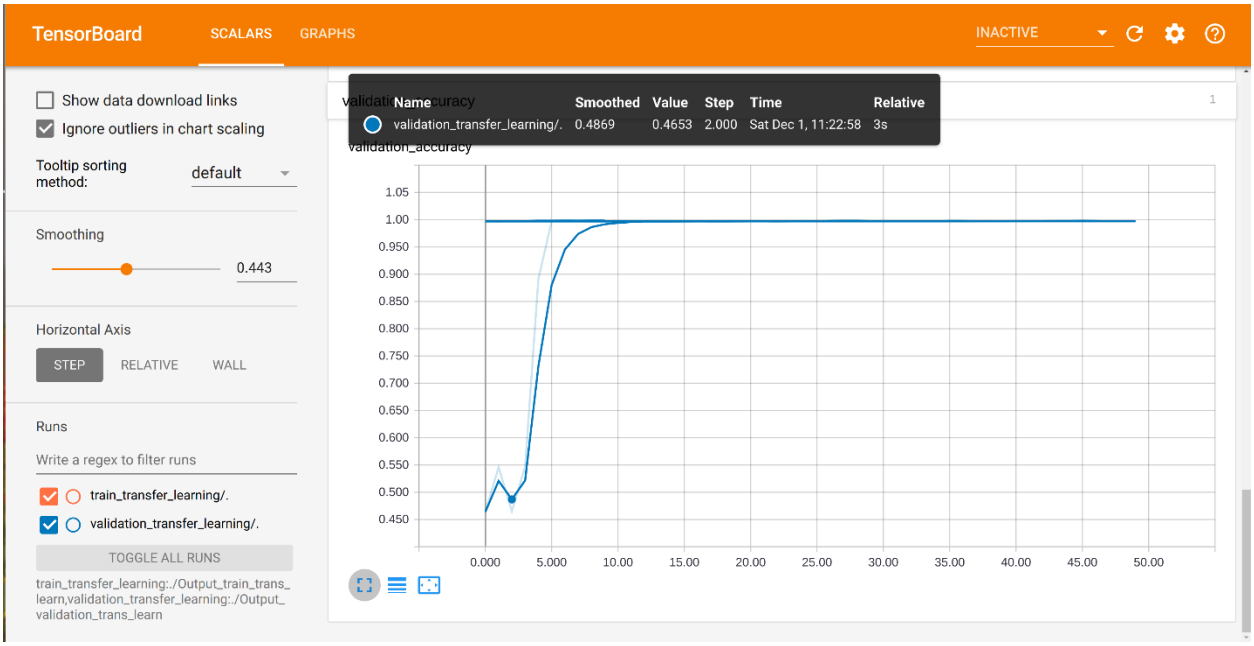
در این قطعه کد داده‌های مربوط به دو کلاس ۱ و ۴ جدا می‌شود. (هم داده‌ها هم برچسب آن‌ها)

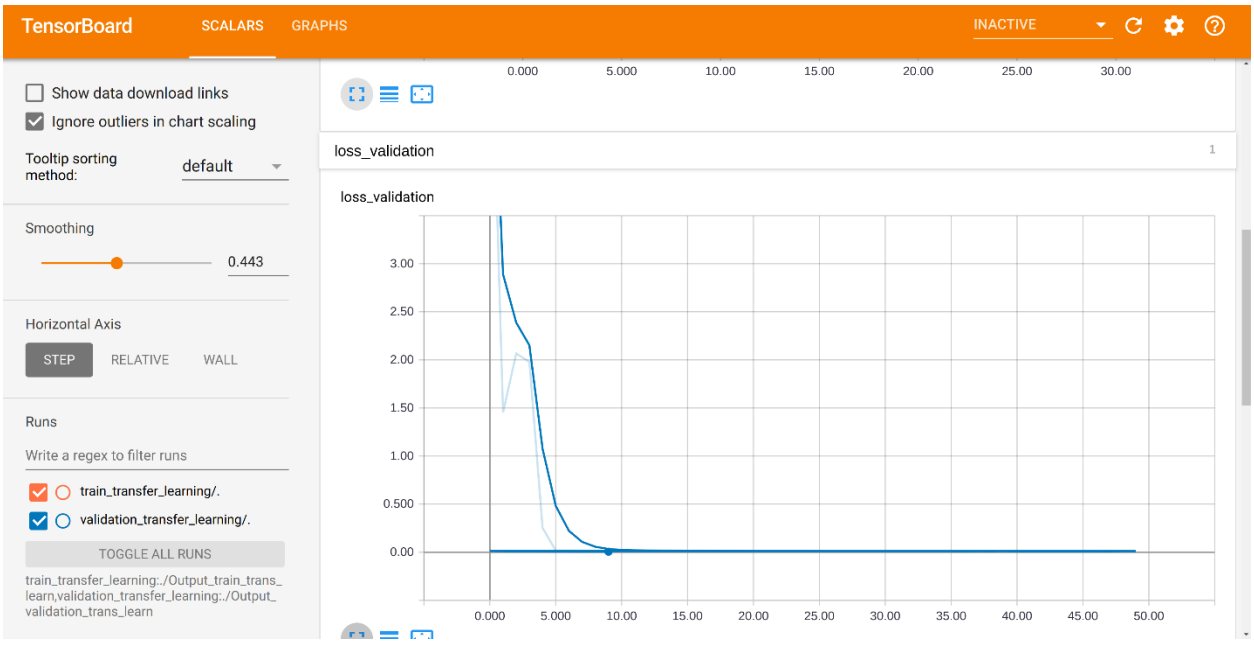
نتیجه دقت و هزینه برای داده‌های اعتبارسنجی و آموزش این مدل (فقط داده‌های آموزش و اعتبارسنجی کلاس ۱ و ۴) در شکل‌های زیر مشاهده می‌شود و دقت اعتبارسنجی بسیار نزدیک ۱ می‌شود.(۹۹/۸۶ درصد)




TensorBoard
SCALARS
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method:
default
Smoothing
0.443
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
train_transfer_learning/.
validation_transfer_learning/.
TOGGLE ALL RUNS
train_transfer_learning:./Output_train_trans_learn,validation_transfer_learning:./Output_validation_trans_learn
train_accuracy
train_accuracy
1.01
1.00
0.990
0.980
0.970
0.960
0.950
0.940
0.930
0.920
0.910
0.900
0.000
5.000
10.00
15.00
20.00
25.00
30.00
validation_accuracy


TensorBoard
SCALARS
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method:
default
Smoothing
0.443
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
train_transfer_learning/.
validation_transfer_learning/.
TOGGLE ALL RUNS
train_transfer_learning:./Output_train_trans_learn,validation_transfer_learning:./Output_validation_trans_learn
loss_train
loss_train
1.40
1.20
1.00
0.800
0.600
0.400
0.200
0.00
0.000
5.000
10.00
15.00
20.00
25.00
30.00
loss_validation


TensorBoard
SCALARS
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method:
default
Smoothing
0.443
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
train_transfer_learning/.
validation_transfer_learning/.
TOGGLE ALL RUNS
train_transfer_learning:./Output_train_trans_learn,validation_transfer_learning:./Output_validation_trans_learn
Name
Smoothed
Value
Step
Time
Relative
validation_transfer_learning/.
0.4869
0.4653
2.000
Sat Dec 1, 11:22:58
3s
validation_accuracy
1.05
1.00
0.950
0.900
0.850
0.800
0.750
0.700
0.650
0.600
0.550
0.500
0.450
0.000
5.000
10.00
15.00
20.00
25.00
30.00
35.00
40.00
45.00
50.00


TensorBoard
SCALARS
GRAPHS
INACTIVE
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method:
default
Smoothing
0.443
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
train_transfer_learning/.
validation_transfer_learning/.
TOGGLE ALL RUNS
train_transfer_learning:./Output_train_trans_learn,validation_transfer_learning:./Output_validation_trans_learn
loss_validation
3.00
2.50
2.00
1.50
1.00
0.500
0.00
0.000
5.000
10.00
15.00
20.00
25.00
30.00
35.00
40.00
45.00
50.00