TensorFlow for machine learning tasks

D.Shipilov, D.Zhurov

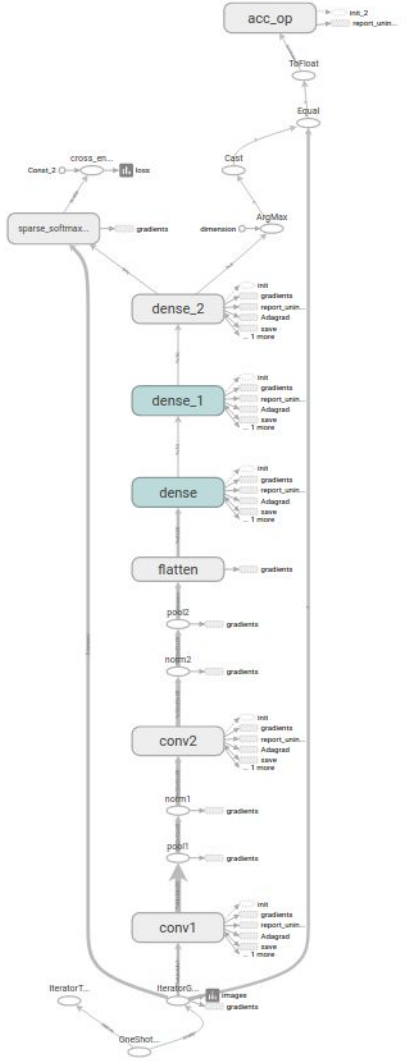
Intro

TensorFlow is an open-source software library for programming across a range of tasks. It is a symbolic math library, and also used for machine learning applications such as neural networks. It is used for both research and production at Google.

Dataflow Graph

The main difference from other libraries is **dataflow graph.** Graph calculation advantages:

- **Parallelism.** operations can execute in parallel.
- **Distributed execution.** partition your program across multiple devices (CPUs and GPUs)
- **Portability.** language-independent representation
- **Visibility.** represent the computation graph in a graphic form



TensorFlow API

TensorFlow API is “divided” into high and low level API.

- Low level API
  - Manage your TensorFlow Graph and runtime
  - Run TensorFlow operations
  - Build your training loop
- High level API
  - Provides a convenient interface for importing data
  - Greatly simplifies machine learning programming by using Estimator class

Solve ML task

Using High level API (Estimator class) it is easy to implement ML program.

1. Write function that read input data
2. Write function that build model graph
    a. Define the model
    b. Implement training, evaluation, and prediction
3. Use Estimator class for call train, evaluate or predict method

Data import

Tensor flow has convenient data import mechanism based on Dataset class.

- It is easy to implement reading and preprocessing from any data format
  - using TensorFlow API, or
  - using arbitrary Python logic
- Reading from single file or from multiple files

An Example for reading images for training

```
def train_input_fn(imfilenames, labels, batch_size=100):
    def _parse_function(filename, label):
        image_string = tf.read_file(filename)
        image_decoded = tf.image.decode_png(image_string)
        image_decoded = tf.cast(image_decoded, tf.float32)
        image_resized = tf.image.resize_images(image_decoded, (64, 64))
        return {'image':image_resized}, label
    dataset = tf.data.Dataset.from_tensor_slices((tf.constant(imfilenames),
                                                  tf.constant(labels)))
    dataset = dataset.map(_parse_function)
    dataset = dataset.shuffle(200).repeat().batch(batch_size)
    return dataset.make_one_shot_iterator().get_next()

classifier.train(input_fn=lambda: train_input_fn(x_train,y_train),
                 steps=5000)
```

Model function

To implement 3 steps are needed:

1. Write model make prediction
2. Calc losses and accuracy
3. Define optimizer for train

```
def my_model(features, labels, mode, params):

    # Create three fully connected nn
    net = tf.feature_column.input_layer(features, params['feature_columns'])
    for units in params['hidden_units']:
        net = tf.layers.dense(net, units=units, activation=tf.nn.relu)
    logits = tf.layers.dense(net, params['n_classes'], activation=None)
    predicted_classes = tf.argmax(logits, 1)

    if mode == tf.estimator.ModeKeys.PREDICT:
        predictions = {
            'class_ids': predicted_classes[:, tf.newaxis],
            'probabilities': tf.nn.softmax(logits),
            'logits': logits,
        }
        return tf.estimator.EstimatorSpec(mode, predictions=predictions)
    # Compute lossand accuracy
    loss = tf.losses.sparse_softmax_cross_entropy(labels=labels, logits=logits)
    accuracy = tf.metrics.accuracy(labels=labels,
                                   predictions=predicted_classes,
                                   name='acc_op')
    metrics = {'accuracy': accuracy}
    tf.summary.scalar('accuracy', accuracy[1])

    if mode == tf.estimator.ModeKeys.EVAL:
        return tf.estimator.EstimatorSpec(
            mode, loss=loss, eval_metric_ops=metrics)

    # Create training op.
    optimizer = tf.train.AdagradOptimizer(learning_rate=0.1)
    train_op = optimizer.minimize(loss, global_step=tf.train.get_global_step())
    return tf.estimator.EstimatorSpec(mode, loss=loss, train_op=train_op)
```

Estimator class

- Estimator class takes model function and params as arguments
- It provide train, evaluate and predict functions that simplifies work with the library
- Train, evaluate and predict functions takes input function as arguments

An Example for using Estimator class

```
classifier = tf.estimator.Estimator(
    model_fn=my_model_fn,
    params={
        'feature_columns': my_feature_columns,
        # Two hidden layers of 10 nodes each.
        'hidden_units': [10, 10],
        # The model must choose between 3 classes.
        'n_classes': 4,
    },
    model_dir='/tmp/test'
)
classifier.train(input_fn=lambda: train_input_fn(x_train,y_train,10), steps=500)
eval_result = classifier.evaluate(input_fn=lambda: eval_input_fn(x_test,y_test,10))
print('\nTest set accuracy: {accuracy:0.3f}\n'.format(**eval_result))
prediction =  classifier.predict(input_fn=lambda: eval_input_fn(x_test[:5],None,10))
```

Logging and saving model

TensorFlow provide mechanisms for logging, saving restoring

- **Checkpoints**. Saving versions of the model created during training
- **Saving variables** for future using.
- **Exporting and importing** models
- **Event files** contains information that TensorBoard uses to create visualizations

An example code for saving variables during train and visualize in TensorBoard

```
def variable_summaries(var):
  """Attach a lot of summaries to a Tensor (for TensorBoard visualization)."""
  with tf.name_scope('summaries'):
    mean = tf.reduce_mean(var)
    tf.summary.scalar('mean', mean)
    with tf.name_scope('stddev'):
      stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
    tf.summary.scalar('stddev', stddev)
    tf.summary.scalar('max', tf.reduce_max(var))
    tf.summary.scalar('min', tf.reduce_min(var))
    tf.summary.histogram('histogram', var)
```

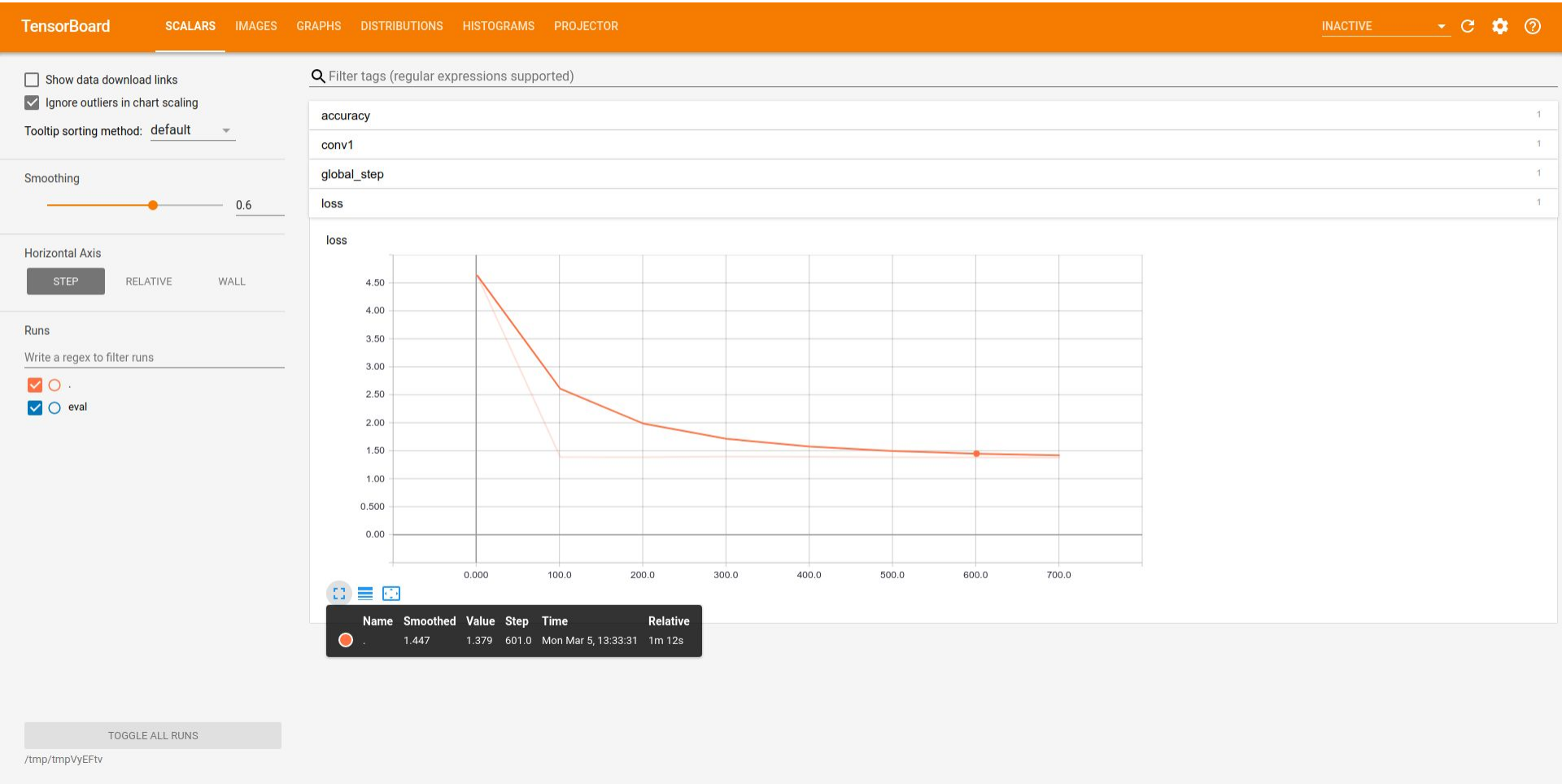
Tensorboard

Tensorboard is a web app to view information about your Tensorflow app. Data is written in Tensorflow and read by Tensorboard.

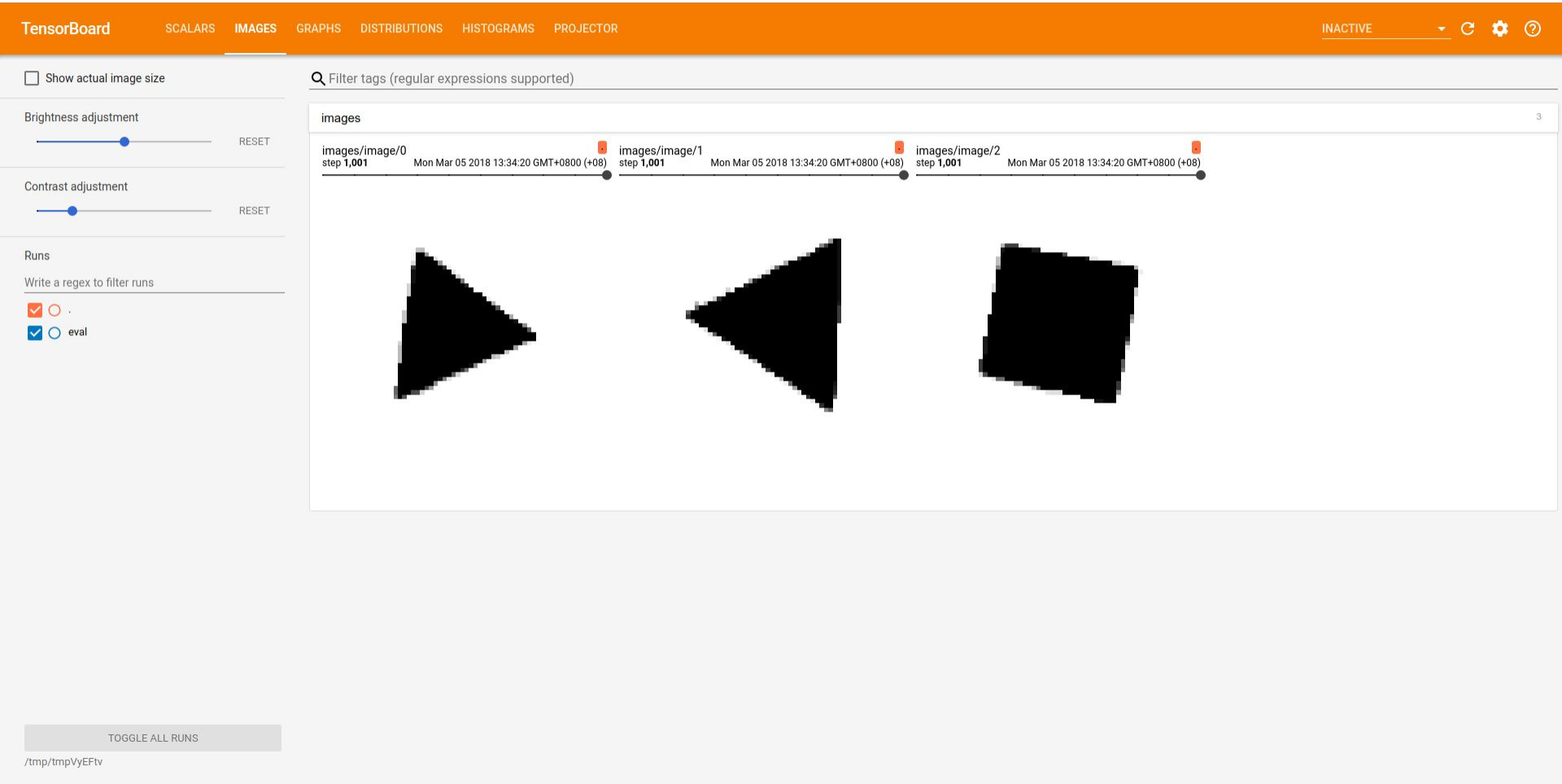
It allows to build the following types of graphics:

- Visualization learning
- Graph visualization
- Histogram
- Precision-recall curve

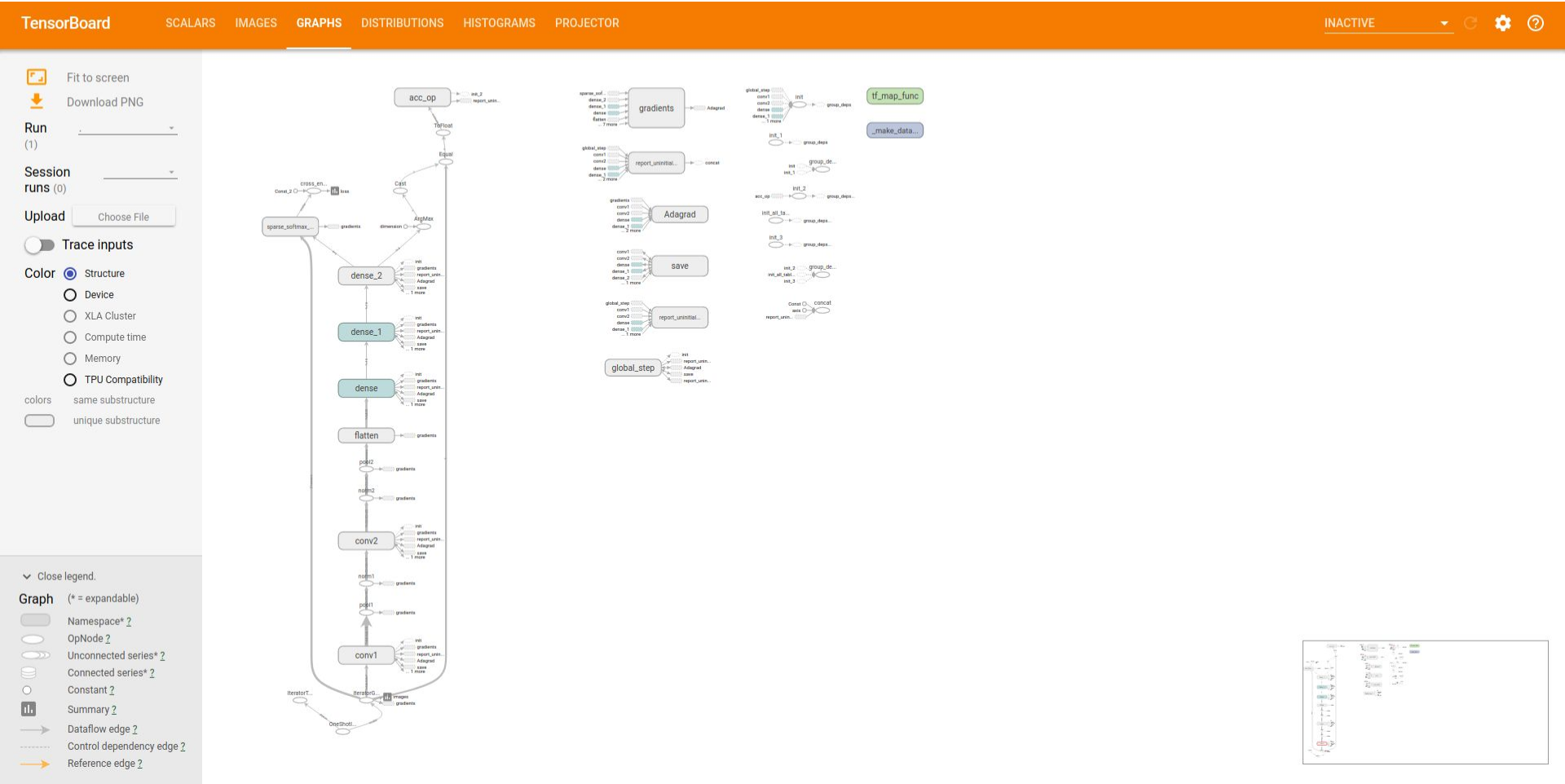
Tensorboard interface



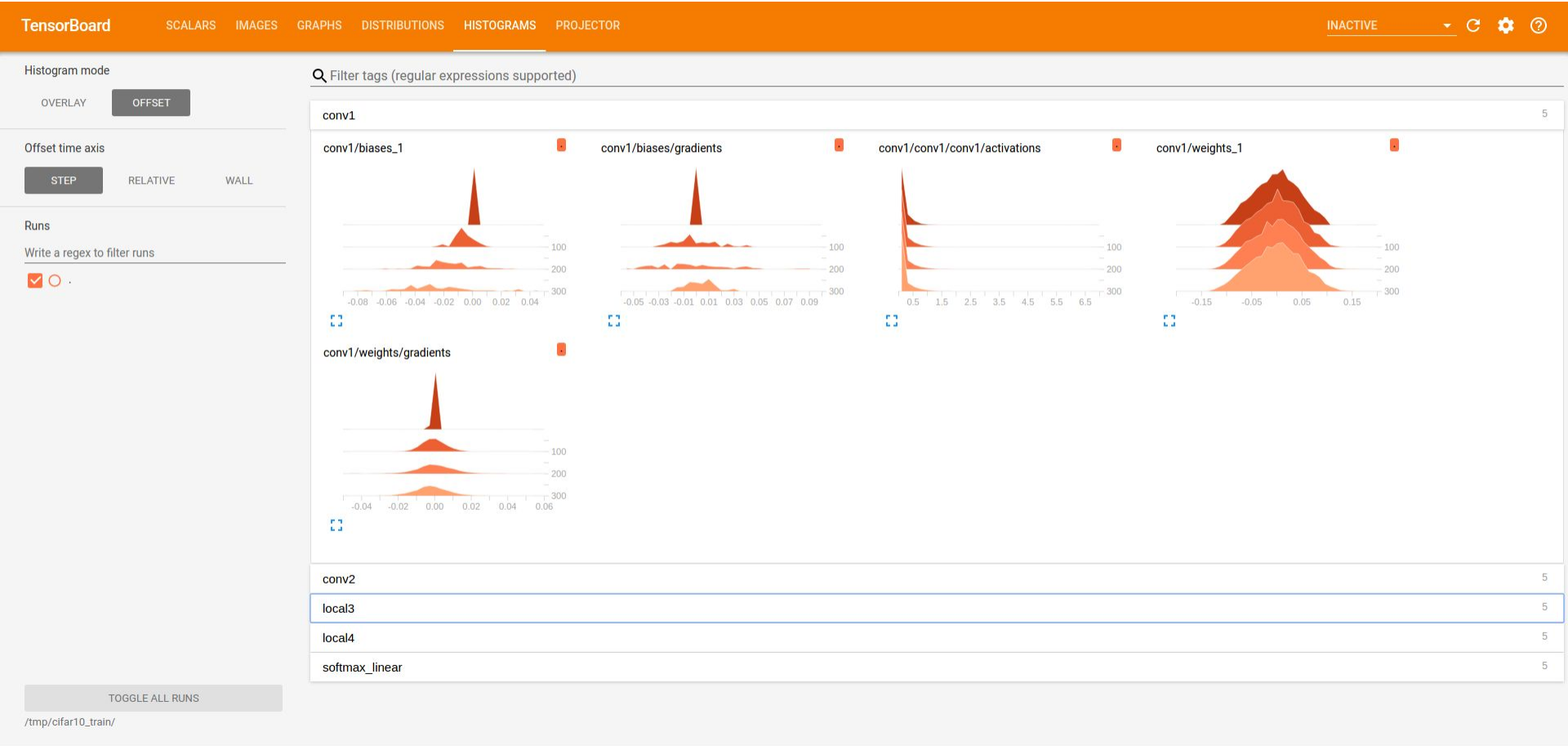
Tensorboard interface



Tensorboard interface



Tensorboard interface



Summary

Positive points

- Computing on GPU (using NVIDIA CUDA)
- Flexibility
- TensorBoard
- Detailed documentation
- Working with Keras

Negative points

- For our project convolution on hexagonal images should be implemented

First experience of using TensorFlow library

- The library is convenient for use
- An attempt has been made to implement the solution of the simple image recognition
- A fully connected neural network was established (works well)
- A Convolutional neural network was implemented (doesn't works well yet - overfitting)
- Working with hexagonal images in process

Keras

Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or MXNet. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

Last year Google's TensorFlow team decided to support Keras in TensorFlow's core library.

Keras

Positive points:

- Easy to learn, easy to use
- Quick build of the model and its implementation
- Contains implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers
- Can be used together with TensorFlow for ML tasks

Negative points:

- Not enough flexible (but extendable)

Keras model example

Keras allows write complex NN model in several code lines.

1. Define model
2. Compile model
3. Fit and evaluate

```
model = Sequential()
# input: 100x100 images with 3 channels -> (100, 100, 3) tensors.
# this applies 32 convolution filters of size 3x3 each.
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd)

model.fit(x_train, y_train, batch_size=32, epochs=10)
score = model.evaluate(x_test, y_test, batch_size=32)
```

Thank you for your attention