Day 3

陳泓仁

Tensorflow

An open source machine learning framework for everyone There are many tutorials that teach you how to implement an easy machine learning model.

Easy concept for tensorflow - Session

```python
import tensorflow as tf

m1 = tf.constant([[2, 2]])
m2 = tf.constant([[3],
                  [3]])
dot_operation = tf.matmul(m1, m2)

print(dot_operation)  # wrong! no result

# method1 use session
sess = tf.Session()
result = sess.run(dot_operation)
print(result)
sess.close()

# method2 use session
with tf.Session() as sess:
    result_ = sess.run(dot_operation)
    print(result_)
```

To activate the “product” and get the result

Easy concept for tensorflow - Placeholder

```python
x1 = tf.placeholder(dtype=tf.float32, shape=None)
y1 = tf.placeholder(dtype=tf.float32, shape=None)
z1 = x1 + y1

x2 = tf.placeholder(dtype=tf.float32, shape=[2, 1])
y2 = tf.placeholder(dtype=tf.float32, shape=[1, 2])
z2 = tf.matmul(x2, y2)

with tf.Session() as sess:
    # when only one operation to run
    z1_value = sess.run(z1, feed_dict={x1: 1, y1: 2})

    # when run multiple operations
    z1_value, z2_value = sess.run(
        [z1, z2],       # run them together
        feed_dict={
            x1: 1, y1: 2,
            x2: [[2], [2]], y2: [[3, 3]]
        })
    print(z1_value)
    print(z2_value)
```

Define your input data format

Easy concept for tensorflow - Placeholder

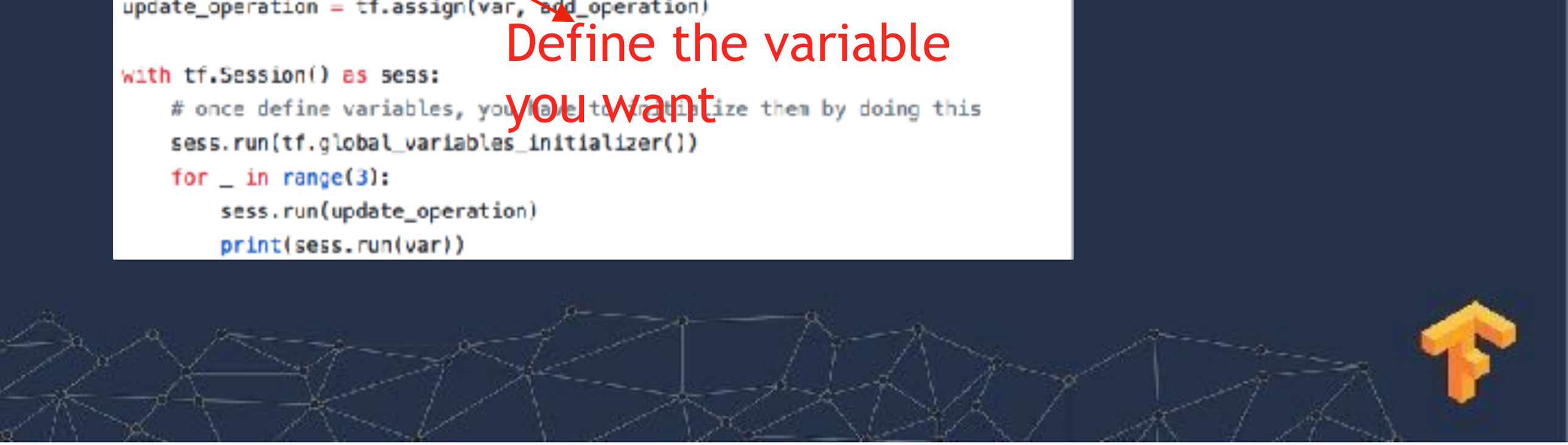
```python
import tensorflow as tf

var = tf.Variable(0)    # our first variable in the "global_variable" set

add_operation = tf.add(var, 1)
update_operation = tf.assign(var, add_operation)

with tf.Session() as sess:
    # once define variables, you have to initialize them by doing this
    sess.run(tf.global_variables_initializer())
    for _ in range(3):
        sess.run(update_operation)
        print(sess.run(var))
```

Define the variable you want



MNIST Dataset

Handwritten digits 0–9, formatted as 28x28-pixel monochrome images 50,000 training examples
10,000 test examples

```
mnist=tf.contrib.learn.datasets.mnist.read_data_sets(train_dir=LOG
DIR+ "data", one_hot=True)
```

Mnist

MNIST Architecture

Layer Definition

Size of filter

Number of feature map

```python
# Define a simple convolutional layer
def conv_layer(input, size_in, size_out):
    w = tf.Variable(tf.truncated_normal([5, 5, size_in, size_out], stddev=0.1))
    b = tf.Variable(tf.constant(0.1, shape=[size_out]))
    conv = tf.nn.conv2d(input, w, strides=[1, 1, 1, 1], padding="SAME")
    act = tf.nn.relu(conv + b)

    return tf.nn.max_pool(act, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="SAME")
```

```python
# And a fully connected layer
def fc_layer(input, size_in, size_out):

    w = tf.Variable(tf.truncated_normal([size_in, size_out], stddev=0.1))
    b = tf.Variable(tf.constant(0.1, shape=[size_out]))
    act = tf.matmul(input, w) + b

    return act
```

Feed-forward Setup

```
# Setup placeholders, and reshape the data
x = tf.placeholder(tf.float32, shape=[None, 784])
x_image = tf.reshape(x, [-1, 28, 28, 1])
y = tf.placeholder(tf.float32, shape=[None, 10])

#build your model
conv1 = conv_layer(x_image, 1, 32)
conv_out = conv_layer(conv1, 32, 64)
flattened = tf.reshape(conv_out, [-1, 7 * 7 * 64])
fc1 = fc_layer(flattened, 7 * 7 * 64, 1024)
relu = tf.nn.relu(fc1)
logits = fc_layer(relu, 1024, 10)
```

Loss & Training

```
#Loss&trainging
xent = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y))
train_step = tf.train.AdamOptimizer(learning_rate).minimize(xent)
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Train the Model

```
sess.run(tf.global_variables_initializer())
for i in range(1000):
  batch = mnist.train.next_batch(100)
  # Occasionally report accuracy  if i % 500 == 0:
  [train_accuracy] = sess.run([accuracy], feed_dict={x: batch[0], y: batch[1]})
  print("step %d, training accuracy %g" % (i, train_accuracy))

# Run the training step
sess.run(train_step, feed_dict={x: batch[0], y: batch[1]})
```

Hyperparameter Search

- What about different learning rates?
- What about different model architectures?

Hyperparameter Search

```
def main():
  # You can try adding some more learning rates
  for learning_rate in [1E-4]:
    conv_param="conv=2"
    fc_param= "fc=2"
    # Construct a hyperparameter string for each one (example: "lr_1E-3,fc=2,conv=2")
    hparam = "lr_%.0E,%s,%s" % (learning_rate, conv_param, fc_param)
    print('Starting run for %s' % hparam)

    # Actually run with the new settings
    mnist_model(learning_rate, hparam)
print('Done training!')
print('Run `tensorboard --logdir=%s` to see the results.' % LOGDIR)
print('Running on mac? If you want to get rid of the dialogue asking to give '
      'network permissions to TensorBoard, you can provide this flag: '
      '--host=localhost')
```

Number Plate Recognition

http://matthewearl.github.io/2016/05/06/cnn-anpr/

Package

conda install -c condo-forge opencv

conda install tensorflow

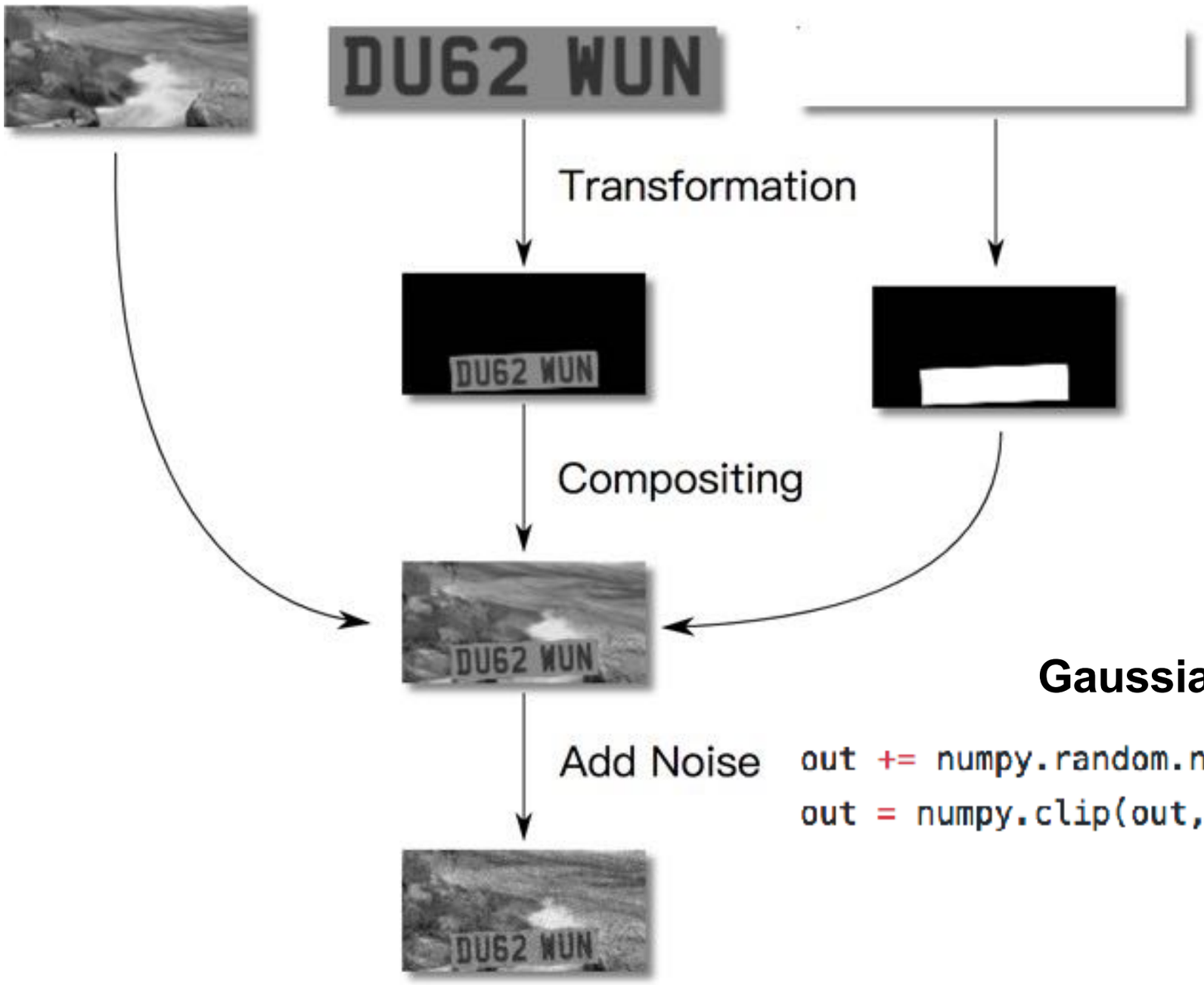
Outline

- Problem Formulation
- Generate Data
- Design Model
- Output Processing

Problem Formulation

- The training image is 64x128, the network should output:
    1. The probability a number plate is present in the input image.
    2. The probability of the digit in each position, ie. for each of the 7 possible positions it should return a probability distribution across the 36 possible characters.

Synthesizing images



Gaussian noise

```
out += numpy.random.normal(scale=0.05, size=out.shape)
out = numpy.clip(out, 0., 1.)
```

Data

Present

Too small

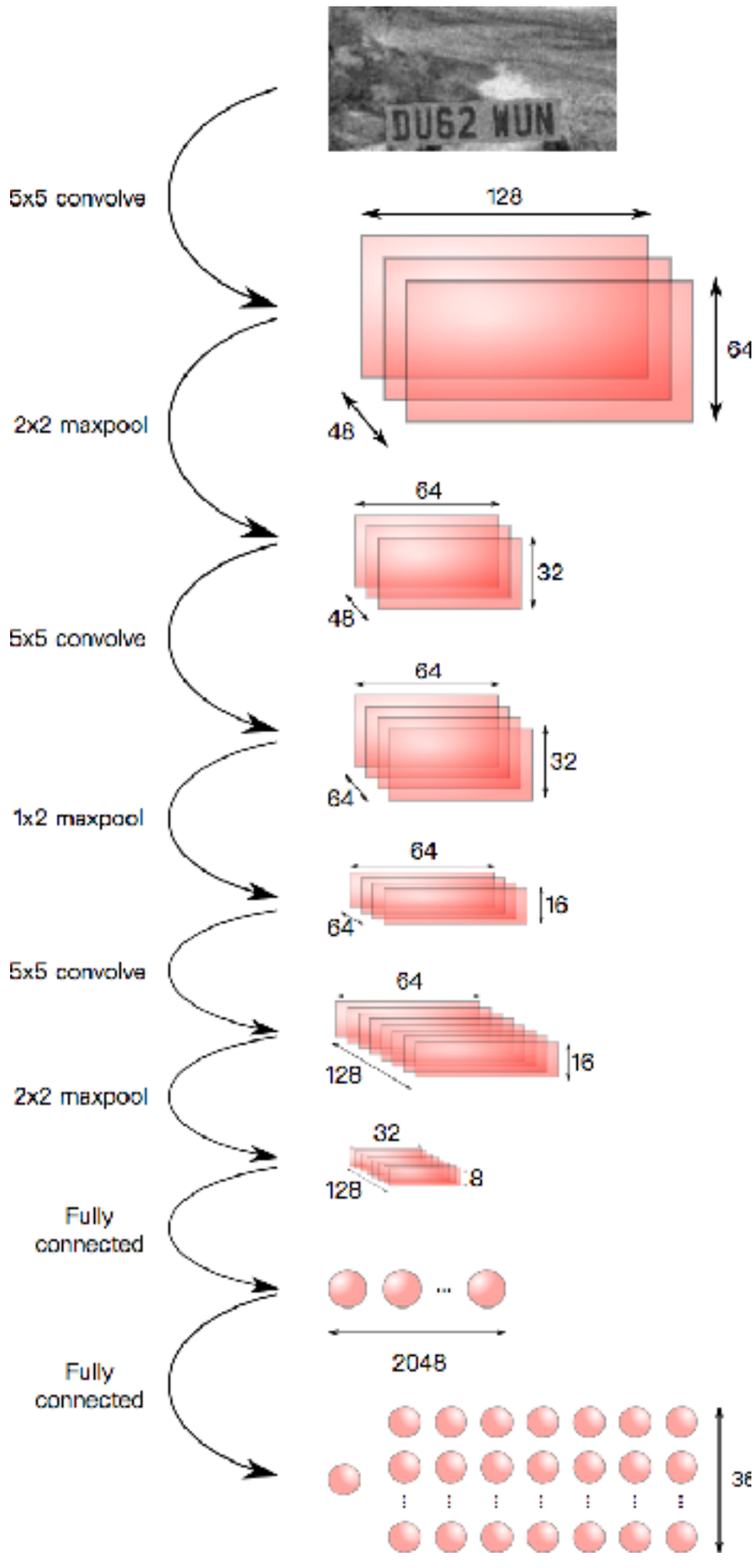
Too large

Not present

Truncated

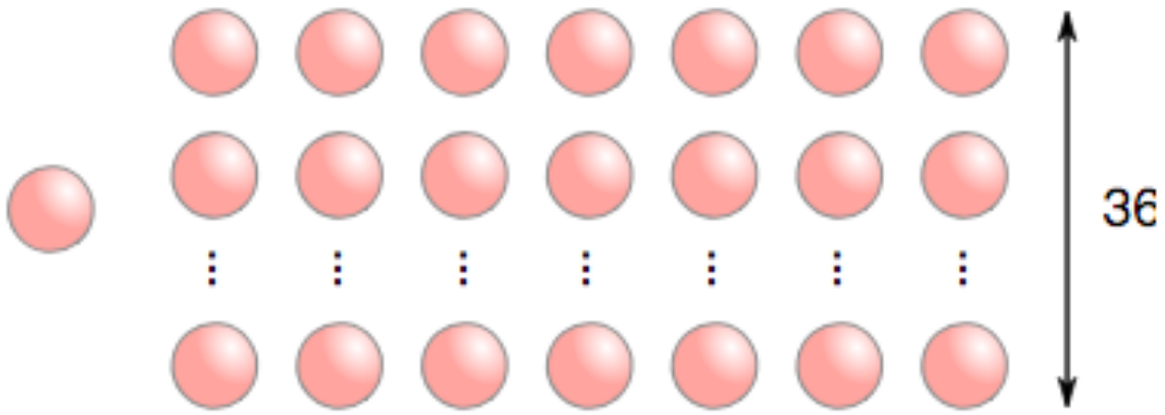
Model

```
last_weights = [p.eval() for p in params]
numpy.savez("weights.npz", *last_weights)
```



Output Layer

The output layer has one node (shown on the left) which is used as the presence indicator. The rest encode the probability of a particular number plate:
Each column as shown in the diagram corresponds with one of the digits in the number plate, and each node gives the probability of the corresponding character being present



Saving weights

The .npz file format is a zipped archive of files named after the variables they contain.
The archive is not compressed and
each file in the archive contains one variable in .npy format.

```
>>> a = np.array([[1,2,3],[4,5,6]])
>>> b = np.arange(0, 1.0, 0.1)
>>> np.savez("result.npz", a, b)
>>> r = np.load("result.npz")
>>> r["arr_0"] # 数组a
array([[1, 2, 3],
       [4, 5, 6]])
>>> r["arr_1"] # 数组b
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9])
```

Output Processing

Output Processing

Checkpoint

Output Processing

```
# Load the model which detects number plates over a sliding window.
x, y, params = model.get_detect_model()

# Execute the model at each scale.
with tf.Session() as sess:
    y_vals = []
    for scaled_im in scaled_ims:
        feed_dict = {x: numpy.stack([scaled_im])}
        feed_dict.update(dict(zip(params, param_vals)))
        y_vals.append(sess.run(y, feed_dict=feed_dict))
```

Output Processing

Task

```
def get_detect_model():
    """
    The same as the training model, except it acts on an arbitrarily sized
    input, and slides the 128x64 window across the image in 8x8 strides.

    The output is of the form `v`, where `v[i, j]` is equivalent to the output
    of the training model, for the window at coordinates `(8 * i, 4 * j)`.

    """
    x, conv_layer, conv_vars = convolutional_layers()

    # Fourth layer
    W_fc1 = weight_variable([8 * 32 * 128, 2048])
    W_conv1 = tf.reshape(W_fc1, [8,  32, 128, 2048])
    b_fc1 = bias_variable([2048])
    h_conv1 = tf.nn.relu(conv2d(conv_layer, W_conv1,
                                stride=(1, 1), padding="VALID") + b_fc1)
    # Fifth layer
    W_fc2 = weight_variable([2048, 1 + 7 * len(common.CHARS)])
    W_conv2 = tf.reshape(W_fc2, [1, 1, 2048, 1 + 7 * len(common.CHARS)])
    b_fc2 = bias_variable([1 + 7 * len(common.CHARS)])
    h_conv2 = conv2d(h_conv1, W_conv2) + b_fc2

    return (x, h_conv2, conv_vars + [W_fc1, b_fc1, W_fc2, b_fc2])
```

Output Processing

```
im = cv2.imread(sys.argv[1])
im_gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY) / 255.

f = numpy.load(sys.argv[2])
param_vals = [f[n] for n in sorted(f.files, key=lambda s: int(s[4:]))]

for pt1, pt2, present_prob, letter_probs in post_process(
                                                detect(im_gray, param_vals)):
    pt1 = tuple(reversed(list(map(int, pt1))))
    pt2 = tuple(reversed(list(map(int, pt2))))

    code = letter_probs_to_code(letter_probs)

    color = (0.0, 255.0, 0.0)
    cv2.rectangle(im, pt1, pt2, color)

    cv2.putText(im,
                code,
                pt1,
                cv2.FONT_HERSHEY_PLAIN,
                1.5,
                (0, 0, 0),
                thickness=5)

    cv2.putText(im,
                code,
                pt1,
                cv2.FONT_HERSHEY_PLAIN,
                1.5,
                (255, 255, 255),
                thickness=2)

cv2.imwrite(sys.argv[3], im)
```

Output Processing

Python detect.py [name].jpg weights.npz [new name].jpg