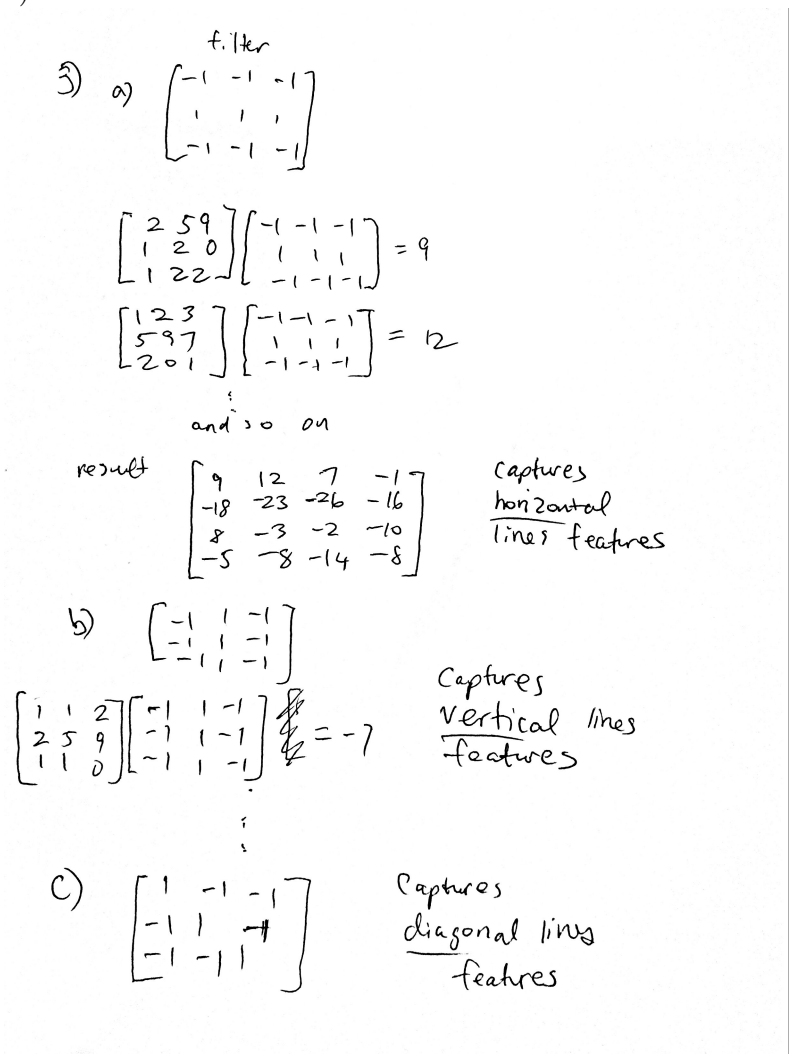
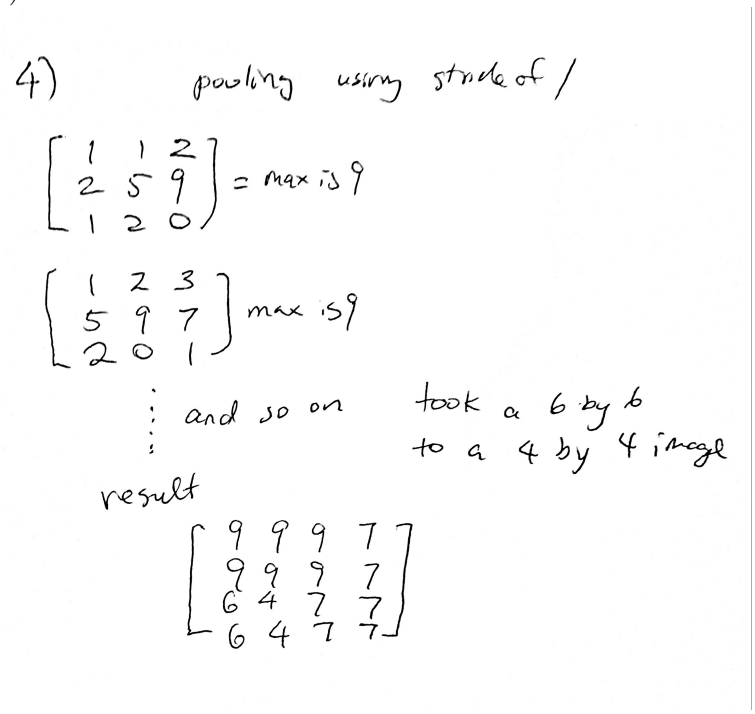
1)
Size = 6-3+1 = 4 By 4
Trainable parameters = ((Size of image) - (size of filter)) * stride + 1 = (6*6 - 2 *2) * 2 + 1 = 65

2)
with padding 1 around, it would be a 8 * 8.
size = 8 - 4 + 1 = 5 by 5
Trainable parameters = ((Size of image) - (size of filter)) * stride + 1 = (8*8 - 2 *2) * 2 + 1 = 121

3)



4)



5)

a) 3*4*4 = 48 weights.

b) After pooling, there's 3x5x5, so 75 reLu operations performed on the forward pass.

c) 48 + 4*75 = 348 weights.

d) It seems logical the answer is true. That another neural network with same architecture can represent the same classifier. Not sure how to explain in depth.

e) It takes too long to train as there will be too many parameters with fully connected.

6) code

```
reset_graph()

X = tf.placeholder(tf.float32, shape=(None, height, width, 1))
feature_maps = tf.constant(fmap)
# TODO :: Apply feature_maps to input image X with tf.nn.conv2d, the strides is 1 for each dimension of input. Expect 1
# line of code
convolution = tf.nn.conv2d(X, feature_maps, strides=[1,1,1,1], padding="SAME")
```

```
# TODO :: create a session and evaluate the convolution operation
with tf.Session() as sess:
    output = convolution.eval(feed_dict={X: images})
```

```
# TODO :: Can you build two filters that can sharpen and blur an image respectively. Test your feature by applying
# it on the china.jpg. Expect 3 lines of code
fmap = np.zeros(shape=(7, 7, 1, 2), dtype=np.float32)
# Dr. Straach showed this in live session notes
fmap[3, 3, 0, 0] = 1
fmap[:, :, 0, 1] = 1/49

plot_image(fmap[:, :, 0, 0])
plt.show()
plot_image(fmap[:, :, 0, 1])
plt.show()
```

```
reset_graph()

with tf.name_scope("inputs"):
    X = tf.placeholder(tf.float32, shape=[None, n_inputs], name="X")
    X_reshaped = tf.reshape(X, shape=[-1, height, width, channels])
    y = tf.placeholder(tf.int32, shape=[None], name="y")

# TOOD : build two convolution layers here, use the hyperparameters we defined earlier, use relu as activation function
# name them as "conv1" and "conv2" respectively. Expect 2 lines of code.
conv1 = tf.layers.conv2d(X_reshaped, filters=conv1_fmaps, kernel_size=conv1_ksize,strides=conv1_stride, padding=conv1_pad,
                         activation=tf.nn.relu, name="conv1")
conv2 = tf.layers.conv2d(conv1, filters=conv2_fmaps, kernel_size=conv2_ksize, strides=conv2_stride, padding=conv2_pad,
                         activation=tf.nn.relu, name="conv2")

with tf.name_scope("pool3"):
    # TODO : define max pooling layer here, the kernel size is 2 * 2, and the strides is 2 for both vertical and horizontal,
    # Use "VALID" as padding method (remember to flatten the pooling result), expect 2 lines of code.
    pool3 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="VALID")
    pool3 = tf.reshape(pool3, shape=[-1, pool3_fmaps * 49])

with tf.name_scope("fc1"):
    # TODO : define a fully connected layer for flattened pooling result. Namse it as "fc1", expect 1 line of code.
     fc1 = tf.layers.dense(pool3, n_fc1, activation=tf.nn.relu, name="fc1")

with tf.name_scope("output"):
    # TODO : define the output layer and the result goes into softmax layer to get the prediction probability.
    # name the output layer as "output", name the softmax layer as "Y_proba", expect 2 lines of code.
    output = tf.layers.dense(fc1, n_outputs, name="output")
    Y_proba = tf.nn.softmax(output, name="Y_proba")


with tf.name_scope("train"):
    # TODO : define a loss function with sparse_softmax_cross_entropy_with_logits, and optimize the loss with AdamOptimizer
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=output, labels=y)
    loss = tf.reduce_mean(xentropy)
    optimizer = tf.train.AdamOptimizer()
    training_op = optimizer.minimize(loss)

with tf.name_scope("eval"):
    # TODO : define the accuracy of the model with tf.reduce_mean, you may want to use tf.nn.in_top_k , expect 2 lines of the
    correct = tf.nn.in_top_k(output, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

with tf.name_scope("init_and_save"):
    init = tf.global_variables_initializer()
    saver = tf.train.Saver()
```