In [1]:

```
from matplotlib import pylab
pylab.rcParams['figure.figsize'] = (10.0, 10.0)

from tensorflow.examples.tutorials.mnist import input_data

import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
sess = tf.InteractiveSession()
```

Initialisations

Let us write some helper functions to initialise weights and biases. We'll initialise weights as Gaussian random variables with mean 0 and variance 0.0025. For biases we'll initialise everything with a constant 0.1. This is because we're mainly going to be using ReLU non-linearities.

In [2]:

```
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.05)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

def conv2d(x, W, strides, padding):
    return tf.nn.conv2d(x, \
        W, strides=strides, padding=padding)

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], \
        strides=[1, 2, 2, 1], padding='SAME')
```

Model

Let's define the model. The model is defined as follows:

- An input that is 728 dimensional vector.
- Reshape the input as 28x28x1 images (only 1 because they are grey scale)
- A convolutional layer with 25 filters of shape 12x12x1 and a ReLU non-linearity (with stride (2, 2) and no padding)
- A convolutional layer with 64 filters of shape 5x5x25 and a ReLU non-linearity (with stride (1, 2) and padding to maintain size)
- A max_pooling layer of shape 2x2
- A fully connected layer taking all the outputs of the max_pooling layer to 1024 units and ReLU nonlinearity
- A fully connected layer taking 1024 units to 10 no activation function (the softmax non-linearity will be included in the loss function rather than in the model)

In [3]:
```
x = tf.placeholder(tf.float32, shape=[None, 784])
x_ = tf.reshape(x, [-1, 28, 28, 1])
y_ = tf.placeholder(tf.float32, shape=[None, 10])

# Define the first convolution layer here
W_conv1 = weight_variable([12,12,1,25])
b_conv1 = bias_variable([25])
h_conv1 = tf.nn.relu(conv2d(x_, W_conv1, 2, \
    [[0,0], [0,0], [0,0], [0,0]]) + b_conv1)
print(h_conv1.shape)
# Define the second convolution layer here
W_conv2 = weight_variable([5,5,25,64])
b_conv2 = bias_variable([64])
h_conv2 = tf.nn.relu(conv2d(h_conv1, W_conv2, \
    [1,1,1,1], "SAME") + b_conv2)

# Define maxpooling
h_pool2 = max_pool_2x2(h_conv2)

# All subsequent layers will be fully connected ignoring geometry so we'll f
# Flatten the h_pool2_layer (as it has a multidimensiona shape)
h_pool2_flat = tf.reshape(h_pool2, [-1,5*5*64])

# Define the first fully connected layer here
W_fc1 = weight_variable([5*5*64, 1024])
b_fc1 = bias_variable([1024])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

# Use dropout for this layer (should you wish)
# keep_prob = tf.placeholder(tf.float32)
# h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# The final fully connected layer
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])
y_conv = tf.matmul(h_fc1, W_fc2) + b_fc2
```

```
(?, 9, 9, 25)
```

Loss Function, Accuracy and Training Algorithm

- We'll use the cross entropy loss function. The loss function is called `tf.nn.cross_entropy_with_logits` in tensorflow.
- Accuray is simply defined as the fraction of data correctly classified.
- For training you should use the AdamOptimizer (read the documentation) and set the learning rate to be 1e-4. You are welcome, and in fact encouraged, to experiment with other optimisation procedures and learning rates.
- (Optional): You may even want to use different filter sizes once you are finished with experimenting with what is asked in this practical

In [4]:

```
# We'll use the cross entropy loss function
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logit

# And classification accuracy
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# And the Adam optimiser
train_step = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(cross_entro
```

```
WARNING:tensorflow:From <ipython-input-4-10e0b012a53a>:2: softmax_cross_ent
ropy_with_logits (from tensorflow.python.ops.nn_ops) is deprecated and will
be removed in a future version.
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.
```

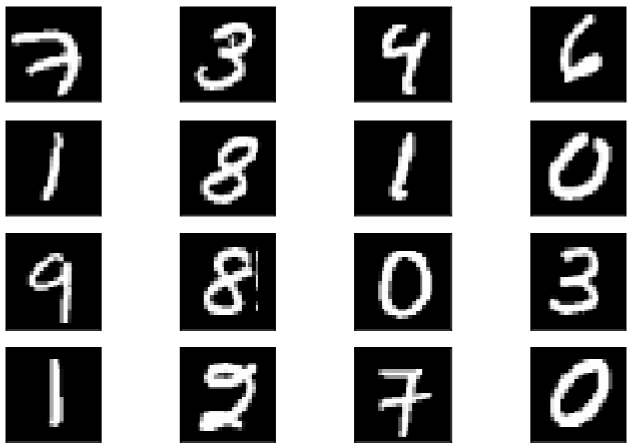
In [5]:

```python
# Load the mnist data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

```
WARNING:tensorflow:From <ipython-input-5-19c914ec135d>:2: read_data_sets (f
rom tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and
will be removed in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/m
odels.
WARNING:tensorflow:From C:\Users\ychen\Anaconda3\lib\site-packages\tensorfl
ow_core\contrib\learn\python\learn\datasets\mnist.py:260: maybe_download (f
rom tensorflow.contrib.learn.python.learn.datasets.base) is deprecated and
will be removed in a future version.
Instructions for updating:
Please write your own downloading logic.
WARNING:tensorflow:From C:\Users\ychen\Anaconda3\lib\site-packages\tensorfl
ow_core\contrib\learn\python\learn\datasets\mnist.py:262: extract_images (f
rom tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and
will be removed in a future version.
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting MNIST_data\train-images-idx3-ubyte.gz
WARNING:tensorflow:From C:\Users\ychen\Anaconda3\lib\site-packages\tensorfl
ow_core\contrib\learn\python\learn\datasets\mnist.py:267: extract_labels (f
rom tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and
will be removed in a future version.
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting MNIST_data\train-labels-idx1-ubyte.gz
WARNING:tensorflow:From C:\Users\ychen\Anaconda3\lib\site-packages\tensorfl
ow_core\contrib\learn\python\learn\datasets\mnist.py:110: dense_to_one_hot
(from tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated a
nd will be removed in a future version.
Instructions for updating:
Please use tf.one_hot on tensors.
Extracting MNIST_data\t10k-images-idx3-ubyte.gz
Extracting MNIST_data\t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From C:\Users\ychen\Anaconda3\lib\site-packages\tensorfl
ow_core\contrib\learn\python\learn\datasets\mnist.py:290: DataSet.__init__
(from tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated a
nd will be removed in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/m
odels.
```

In [6]:
```
# Let us visualise the first 16 data points from the MNIST training data

fig = plt.figure()
for i in range(16):
    ax = fig.add_subplot(4, 4, i + 1)
    ax.set_xticks(())
    ax.set_yticks(())
    ax.imshow(mnist.train.images[i].reshape(28, 28), cmap='Greys_r')
```



In [22]:

```
# Start a tf session and run the optimisation algorithm
sess.run(tf.global_variables_initializer())

# Optimization
for i in range(2000):
    batch = mnist.train.next_batch(50)
    if i%100 == 0:
        train_accuracy = accuracy.eval(feed_dict={
            x:batch[0], y_: batch[1]})
        print("step %d, training accuracy %g"%(i, train_accuracy))
    train_step.run(feed_dict={x: batch[0], y_: batch[1]})
```

```
step 0, training accuracy 0.12
step 100, training accuracy 0.74
step 200, training accuracy 0.86
step 300, training accuracy 0.96
step 400, training accuracy 0.86
step 500, training accuracy 0.9
step 600, training accuracy 0.94
step 700, training accuracy 0.98
step 800, training accuracy 1
step 900, training accuracy 0.9
step 1000, training accuracy 0.96
step 1100, training accuracy 1
step 1200, training accuracy 0.94
step 1300, training accuracy 0.96
step 1400, training accuracy 0.98
step 1500, training accuracy 0.92
step 1600, training accuracy 0.98
step 1700, training accuracy 0.98
step 1800, training accuracy 1
step 1900, training accuracy 0.94
```

In [23]:

```
# Print accuracy on the test set
print ('Test accuracy: %g' % sess.run(accuracy, feed_dict={x: mnist.test.ima
```

```
Test accuracy: 0.9796
```

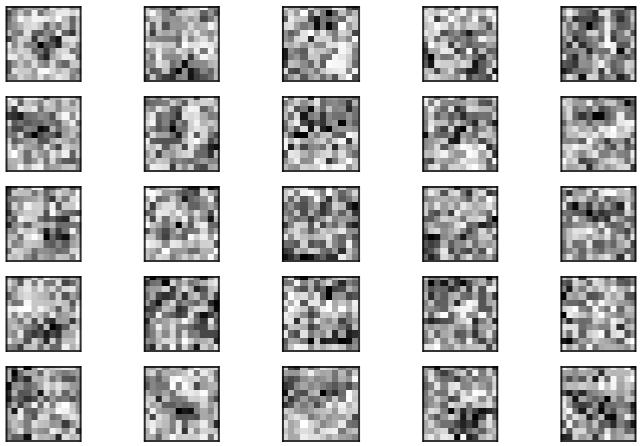
Visualising the Filters

We'll now visualise all the 32 filters in the first convolution layer. As they are each of shape 12x12x1, they may themselves be viewed as greyscale images. Visualising filters in further layers is more complicated and involves modifying the neural network. See the paper (https://arxiv.org/pdf/1311.2901.pdf (https://arxiv.org/pdf/1311.2901.pdf)) by Matt Zeiler and Rob Fergus if you are interested.

In [9]:

```
# Visualise the filters in the first convolutional layer
with sess.as_default():
    W = W_conv1.eval()

# Add code to visualise filters here
fig = plt.figure()
for i in range(25):
    ax = fig.add_subplot(5, 5, i + 1)
    ax.set_xticks(())
    ax.set_yticks(())
    ax.imshow(np.squeeze(np.reshape(W[:,:,:,i], (1,12,12))), cmap='Greys_r')
```



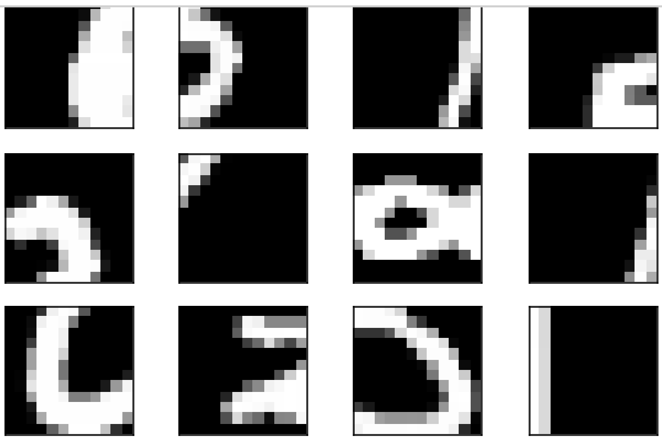
Identifying image patches that activate the filters

For this part you'll find the 12 patches in the test-set that activate each of the first 5 filters that maximise the activation for that filter.

In [77]:

```
H =  sess.run(h_conv1, feed_dict={x: mnist.test.images})
for filterNum in range(0,5):
    rankings = np.argsort(H[:,:,:,filterNum], axis=None)[-12:]

    images = mnist.train.images.reshape(-1,28, 28,1)
    fig = plt.figure()
    for i in range(12):
        ax = fig.add_subplot(3, 4, i + 1)
        ax.set_xticks(())
        ax.set_yticks(())
        plt.suptitle("Filter " + str(filterNum))
        index = np.unravel_index(rankings[i], H.shape)
        ax.imshow(images[index[0], \
            index[1]*2:(index[1]*2)+12, index[2]*2:(index[2]*2)+12,
            0], cmap='Greys_r')
```



Filter 1