In [1]:
```
import DeepLearning as DL
import numpy as np
from keras import backend as K
from keras import models
from keras import layers
from keras import optimizers

# Use a MobileNet as convolutional base for feature extraction
from keras.applications import MobileNet

K.clear_session()

# MobileNet model width, one of [0.25, 0.5, 0.75, 1.0]
alpha = 0.75

# Image size, one of [128, 160, 192, 224]
input_shape = (160, 160, 3)

# Load a MobileNet with pre-trained ImageNet weights
# No classifier, only the convolutional base
conv_base = MobileNet(weights='imagenet',
                      include_top = False,
                      input_shape=input_shape,
                      alpha=alpha,
                      pooling='avg')

# Needed here when called from a webapp like jupyter notebook;
# tensorflow does not play well with threads
conv_base._make_predict_function()
```

```
/usr/local/lib/python2.7/dist-packages/h5py/__init__.py:36: FutureWarning:
Conversion of the second argument of issubdtype from `float` to `np.floati
ng` is deprecated. In future, it will be treated as `np.float64 == np.dtyp
e(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [2]:
```
conv_base.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 160, 160, 3)       0
_________________________________________________________________
conv1 (Conv2D)               (None, 80, 80, 24)        648
_________________________________________________________________
conv1_bn (BatchNormalization (None, 80, 80, 24)        96
_________________________________________________________________
conv1_relu (Activation)      (None, 80, 80, 24)        0
_________________________________________________________________
conv_dw_1 (DepthwiseConv2D)  (None, 80, 80, 24)        216
_________________________________________________________________
conv_dw_1_bn (BatchNormaliza (None, 80, 80, 24)        96
_________________________________________________________________
conv_dw_1_relu (Activation)  (None, 80, 80, 24)        0
_________________________________________________________________
conv_pw_1 (Conv2D)           (None, 80, 80, 48)        1152
_________________________________________________________________
conv_pw_1_bn (BatchNormaliza (None, 80, 80, 48)        192
```

In [3]:
```
train_dir = 'Pet_Dataset/train'
test_dir  = 'Pet_Dataset/test'

train_features, train_labels = DL.extract_features(conv_base,
                                                   train_dir,
                                                   4800)
test_features,  test_labels  = DL.extract_features(conv_base,
                                                   test_dir,
                                                   1200)
```

```
Found 4800 images belonging to 2 classes.
{'Dog': 1, 'Cat': 0}
Pet_Dataset/train: [=======================================>] 100.00% 181.
390801907s
Found 1200 images belonging to 2 classes.
{'Dog': 1, 'Cat': 0}
Pet_Dataset/test: [=======================================>] 100.00% 46.71
86050415s
```

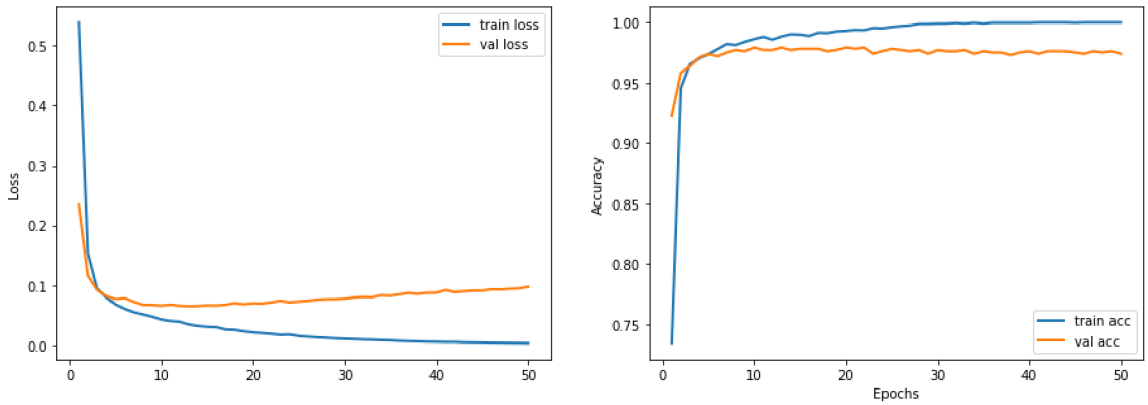
In [4]:
```
# Build a custom DNN classifier
def build_classifier():
    model = models.Sequential()
    model.add(layers.Dense(16, activation='relu',
                           input_dim=int(alpha*1024)))
    model.add(layers.Dense(16, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(optimizer=optimizers.sgd(momentum=0.8),
                  loss='binary_crossentropy',
                  metrics=['acc'])
    return model

classifier = build_classifier()
classifier.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 16)                12304
_________________________________________________________________
dense_2 (Dense)              (None, 16)                272
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 17
=================================================================
Total params: 12,593
Trainable params: 12,593
Non-trainable params: 0
_________________________________________________________________
```

In [5]:
```
# Train the classifier using the extracted features
batch_size = 200
history = classifier.fit(train_features, train_labels,
                         batch_size=batch_size, epochs=50,
                         validation_split=0.20)
```

```
Train on 3840 samples, validate on 960 samples
Epoch 1/50
3840/3840 [==============================] - 0s 99us/step - loss: 0.5385 -
acc: 0.7344 - val_loss: 0.2348 - val_acc: 0.9229
Epoch 2/50
3840/3840 [==============================] - 0s 23us/step - loss: 0.1538 -
acc: 0.9456 - val_loss: 0.1162 - val_acc: 0.9583
Epoch 3/50
3840/3840 [==============================] - 0s 25us/step - loss: 0.0951 -
acc: 0.9654 - val_loss: 0.0927 - val_acc: 0.9635
Epoch 4/50
3840/3840 [==============================] - 0s 24us/step - loss: 0.0803 -
acc: 0.9701 - val_loss: 0.0826 - val_acc: 0.9708
Epoch 5/50
3840/3840 [==============================] - 0s 25us/step - loss: 0.0682 -
acc: 0.9737 - val_loss: 0.0772 - val_acc: 0.9740
Epoch 6/50
3840/3840 [==============================] - 0s 25us/step - loss: 0.0610 -
acc: 0.9779 - val_loss: 0.0788 - val_acc: 0.9719
Epoch 7/50
```

In [6]:
```
DL.plot_history(history.history)
```

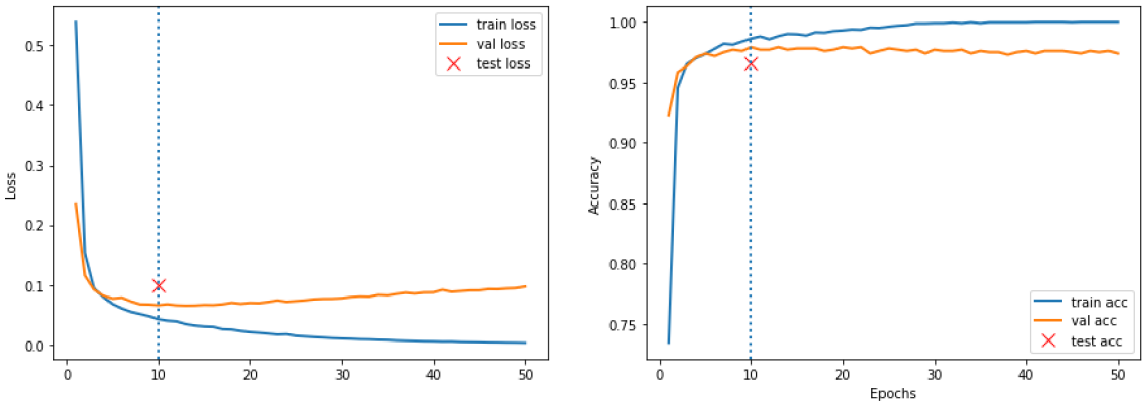


In [7]:
```
# Train the final classifier at the onset of overfitting
final_epochs = 10

classifier = build_classifier()
classifier.fit(train_features, train_labels,
               batch_size=batch_size, epochs=final_epochs)

test_loss, test_acc = classifier.evaluate(test_features, test_labels)
print
print 'Test loss:', test_loss
print 'Test accuracy:', test_acc
history.history['epochs'] = final_epochs
history.history['test_loss'] = test_loss
history.history['test_acc'] = test_acc
```

```
Epoch 1/10
4800/4800 [==============================] - 0s 78us/step - loss: 0.3755 -
acc: 0.8356
Epoch 2/10
4800/4800 [==============================] - 0s 19us/step - loss: 0.1238 -
acc: 0.9554
Epoch 3/10
4800/4800 [==============================] - 0s 20us/step - loss: 0.0901 -
acc: 0.9652
Epoch 4/10
4800/4800 [==============================] - 0s 21us/step - loss: 0.0750 -
acc: 0.9710
Epoch 5/10
4800/4800 [==============================] - 0s 23us/step - loss: 0.0665 -
acc: 0.9735
Epoch 6/10
4800/4800 [==============================] - 0s 24us/step - loss: 0.0589 -
acc: 0.9775
Epoch 7/10
4800/4800 [==============================] - 0s 22us/step - loss: 0.0543 -
acc: 0.9777
Epoch 8/10
4800/4800 [==============================] - 0s 21us/step - loss: 0.0517 -
acc: 0.9815
Epoch 9/10
4800/4800 [==============================] - 0s 20us/step - loss: 0.0452 -
acc: 0.9838
Epoch 10/10
4800/4800 [==============================] - 0s 20us/step - loss: 0.0438 -
acc: 0.9848
1200/1200 [==============================] - 0s 122us/step

Test loss: 0.10073441644509633
Test accuracy: 0.9658333333333333
```

In [8]:
```
DL.plot_history(history.history)
```

```
Test loss: 0.10073441644509633
Test accuracy: 0.9658333333333333
Final training epochs: 10
```



In [9]:
```
# Generate final model with both convolutional base & classifier
model = models.Sequential()
model.add(conv_base)
model.add(classifier)
model.compile(optimizer=optimizers.sgd(momentum=0.5),
              loss='binary_crossentropy',
              metrics=['acc'])
model.summary()

# Save model to disk
model.save('./cats_and_dogs_trained.h5')
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenet_0.75_160 (Model)   (None, 768)               1832976
_________________________________________________________________
sequential_2 (Sequential)    (None, 1)                 12593
=================================================================
Total params: 1,845,569
Trainable params: 1,829,153
Non-trainable params: 16,416
_________________________________________________________________
```