Part 3 - Evaluating Models

In this notebook we will cover the following topics:

- Evaluating model performance
- Controlling overfitting with dropout
- Increasing model complexity

Evaluating Model Performance

One of the most important parts of the data science workflow is evaluating the performance of a trained model and deciding:

1. Is it good enough? (If so, stop!)
2. If not, how should it be changed?

Let's load up Keras and train an overly simple model on the CIFAR10 data.

In [1]:
```
import numpy as np
np.warnings.filterwarnings('ignore')  # Hide np.floating warning

import keras

from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D

# Prevent TensorFlow from grabbing all the GPU memory
import tensorflow as tf
config = tf.ConfigProto()
config.gpu_options.allow_growth=True
sess = tf.Session(config=config)

import holoviews as hv
hv.extension('bokeh')
```

```
Using TensorFlow backend.
```

Load the Data

Same data preparation as before.

(*Pro tip*: If this wasn't a tutorial, we'd move these repetitive code to a Python module and import it in the notebook to ensure we do it consistently in every experiment.)

In [3]:
```
from keras.datasets import cifar10
import keras.utils

(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Save an unmodified copy of y_test for later, flattened to one column
y_test_true = y_test[:,0].copy()

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# The data only has numeric categories so we also have the string labels below

cifar10_labels = np.array(['airplane', 'automobile', 'bird', 'cat', 'deer',
                           'dog', 'frog', 'horse', 'ship', 'truck'])
```

Create a Basic Model

This model resembles the one from the previous notebook, but we've removed one of the convolutional groups

In [4]:
```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=x_train.shape[1:]))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

In [5]:
```
history = model.fit(x_train, y_train,
          batch_size=128,
          epochs=8,
          verbose=1,
          validation_data=(x_test, y_test))
```

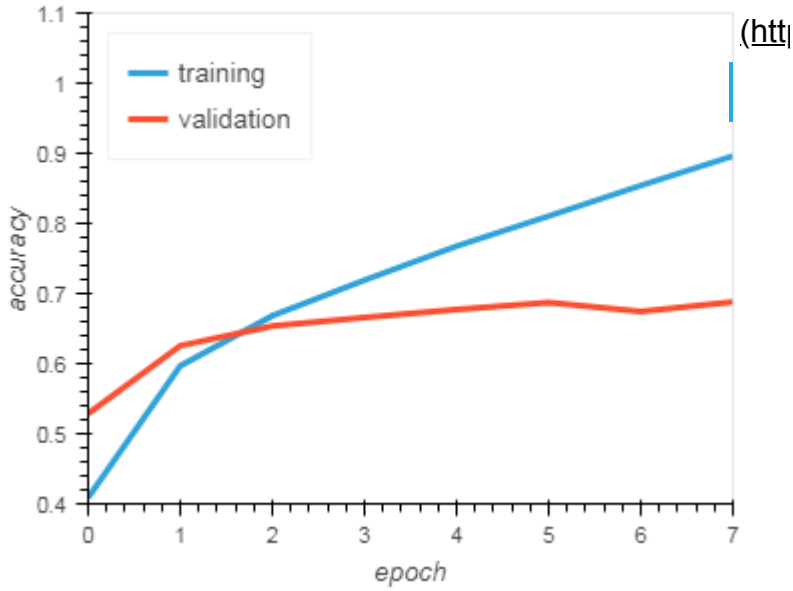
```
Train on 50000 samples, validate on 10000 samples
Epoch 1/8
50000/50000 [==============================] - 12s 249us/step - loss: 1.6504
- acc: 0.4090 - val_loss: 1.3074 - val_acc: 0.5283
Epoch 2/8
50000/50000 [==============================] - 11s 226us/step - loss: 1.1488
- acc: 0.5970 - val_loss: 1.0609 - val_acc: 0.6255
Epoch 3/8
50000/50000 [==============================] - 11s 225us/step - loss: 0.9486
- acc: 0.6683 - val_loss: 0.9939 - val_acc: 0.6537
Epoch 4/8
50000/50000 [==============================] - 11s 224us/step - loss: 0.8058
- acc: 0.7187 - val_loss: 0.9585 - val_acc: 0.6660
Epoch 5/8
50000/50000 [==============================] - 11s 219us/step - loss: 0.6718
- acc: 0.7675 - val_loss: 0.9536 - val_acc: 0.6769
Epoch 6/8
50000/50000 [==============================] - 11s 222us/step - loss: 0.5461
- acc: 0.8102 - val_loss: 0.9467 - val_acc: 0.6872
Epoch 7/8
50000/50000 [==============================] - 11s 223us/step - loss: 0.4250
- acc: 0.8540 - val_loss: 1.0321 - val_acc: 0.6742
Epoch 8/8
50000/50000 [==============================] - 11s 224us/step - loss: 0.3090
- acc: 0.8962 - val_loss: 1.1161 - val_acc: 0.6878
```

In [6]:
```
%%opts Curve [width=400 height=300]
%%opts Curve (line_width=3)
%%opts Overlay [legend_position='top_left']

train_acc = hv.Curve((history.epoch, history.history['acc']), 'epoch', 'accura
cy', label='training')
val_acc = hv.Curve((history.epoch, history.history['val_acc']), 'epoch', 'accu
racy', label='validation')

(train_acc * val_acc).redim(accuracy=dict(range=(0.4, 1.1)))
```

Out[6]:



This model shows a huge discrepancy in accuracy between the training and validation data, a sign of overfitting. After the epoch 2, additional training is not helping. The model is essentially memorizing the training data and not generalizing at all.

Plotting the Confusion Matrix

When dealing with models that predict categories, it is helpful to look at the confusion matrix as well. This will show which categories are being predicted poorly, and what kind of mispredictions are happening.

As the confusion matrix is a standard tool in all of machine learning, the `sklearn` package includes a function that computes it from an array of true category IDs and an array of predicted category IDs:

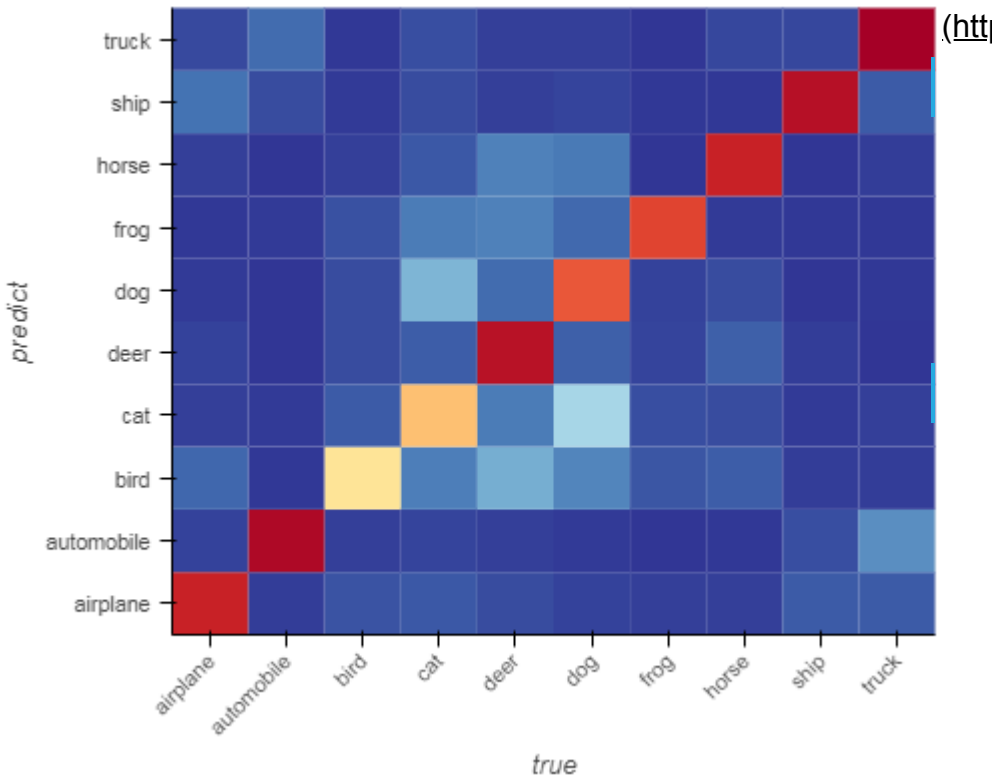
In [7]:
```
from sklearn.metrics import confusion_matrix

y_pred = model.predict_classes(x_test)
confuse = confusion_matrix(y_test_true, y_pred)
```

In [8]:
```
# Holoviews hack to tilt labels by 45 degrees
from math import pi
def angle_label(plot, element):
    plot.state.xaxis.major_label_orientation = pi / 4
```

In [9]:
```
%%opts HeatMap [width=500 height=400 tools=['hover'] finalize_hooks=[angle_lab
el]]
hv.HeatMap((cifar10_labels, cifar10_labels, confuse)).redim.label(x='true', y
='predict')
```

Out[9]:



From this we can see that dogs, deer, cats, and birds are particularly problematic classes, with the confusion between cats and dogs being especially high. Note that because the test data are already balanced to have equal examples from each class, we do not need to do any special normalization of the above.

Controlling Overfit with Dropout

Overfitting is more or less inevitable if we train long enough. The goal is to control it with tools like regularization or dropout. Dropout (https://keras.io/layers/core/#dropout) is a surprisingly effective technique where layer inputs are passed to the output, with a random subset of outputs forced to zero during training. The subset of zeroed outputs changes after every batch. When the model is used for prediction after training, the dropout layers have no effect.

For more details about dropout, see this paper (http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf).

In [11]:
```
model2 = Sequential()
model2.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=x_train.shape[1:]))
model2.add(Conv2D(64, (3, 3), activation='relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Dropout(0.25))

model2.add(Flatten())
model2.add(Dense(128, activation='relu'))

model2.add(Dropout(0.5))

model2.add(Dense(num_classes, activation='softmax'))

model2.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

In [12]:
```
history2 = model2.fit(x_train, y_train,
          batch_size=128,
          epochs=11,
          verbose=1,
          validation_data=(x_test, y_test))
```

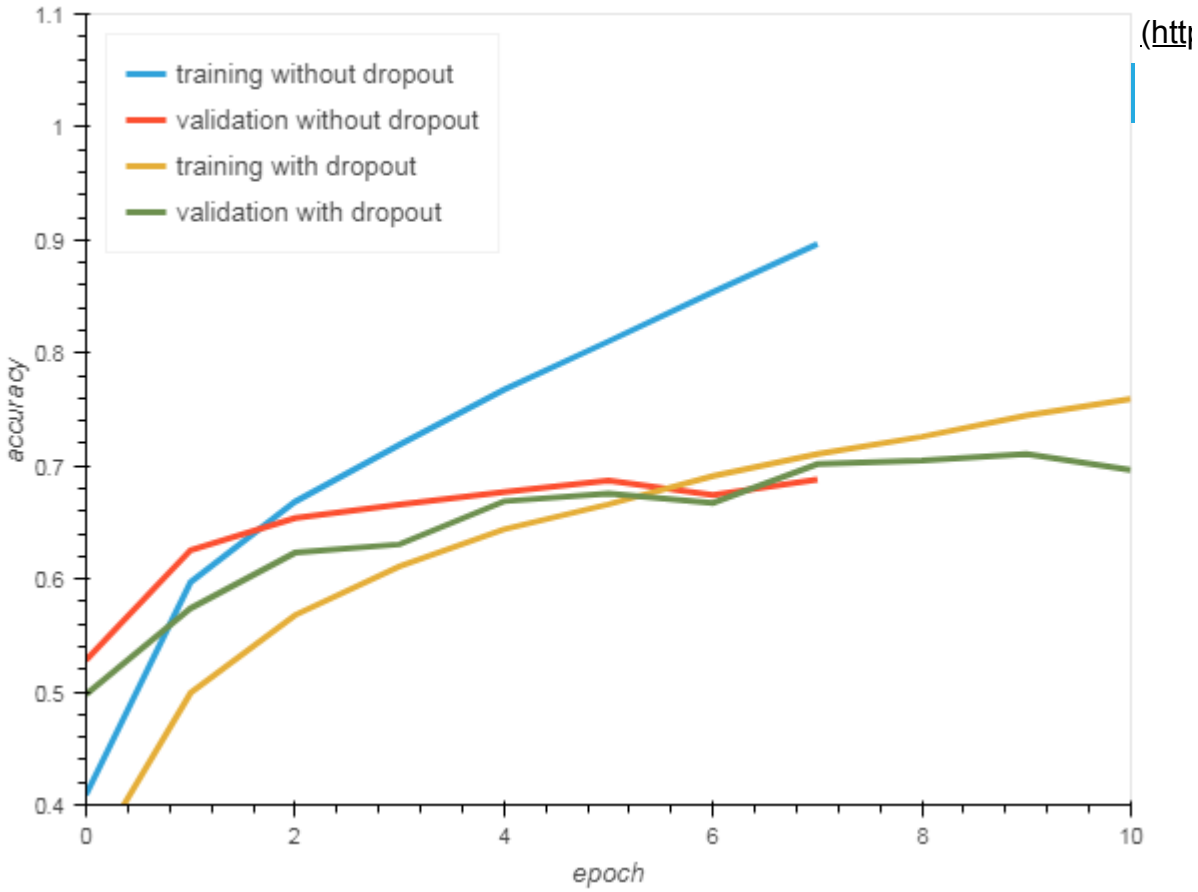
```
Train on 50000 samples, validate on 10000 samples
Epoch 1/11
50000/50000 [==============================] - 12s 243us/step - loss: 1.8259
- acc: 0.3439 - val_loss: 1.4563 - val_acc: 0.4977
Epoch 2/11
50000/50000 [==============================] - 12s 237us/step - loss: 1.4114
- acc: 0.4990 - val_loss: 1.2109 - val_acc: 0.5740
Epoch 3/11
50000/50000 [==============================] - 12s 234us/step - loss: 1.2212
- acc: 0.5680 - val_loss: 1.0627 - val_acc: 0.6232
Epoch 4/11
50000/50000 [==============================] - 12s 234us/step - loss: 1.1008
- acc: 0.6111 - val_loss: 1.0774 - val_acc: 0.6306
Epoch 5/11
50000/50000 [==============================] - 12s 231us/step - loss: 1.0173
- acc: 0.6439 - val_loss: 0.9455 - val_acc: 0.6687
Epoch 6/11
50000/50000 [==============================] - 12s 231us/step - loss: 0.9505
- acc: 0.6661 - val_loss: 0.9373 - val_acc: 0.6756
Epoch 7/11
50000/50000 [==============================] - 12s 231us/step - loss: 0.8866
- acc: 0.6912 - val_loss: 0.9464 - val_acc: 0.6671
Epoch 8/11
50000/50000 [==============================] - 11s 230us/step - loss: 0.8326
- acc: 0.7105 - val_loss: 0.8574 - val_acc: 0.7017
Epoch 9/11
50000/50000 [==============================] - 11s 229us/step - loss: 0.7800
- acc: 0.7259 - val_loss: 0.8557 - val_acc: 0.7050
Epoch 10/11
50000/50000 [==============================] - 12s 230us/step - loss: 0.7308
- acc: 0.7448 - val_loss: 0.8478 - val_acc: 0.7107
Epoch 11/11
50000/50000 [==============================] - 12s 234us/step - loss: 0.6881
- acc: 0.7594 - val_loss: 0.8949 - val_acc: 0.6965
```

In [13]:
```
%%opts Curve [width=600 height=450]
%%opts Curve (line_width=3)
%%opts Overlay [legend_position='top_left']

train_acc = hv.Curve((history.epoch, history.history['acc']), 'epoch', 'accura
cy', label='training without dropout')
val_acc = hv.Curve((history.epoch, history.history['val_acc']), 'epoch', 'accu
racy', label='validation without dropout')
train_acc2 = hv.Curve((history2.epoch, history2.history['acc']), 'epoch', 'acc
uracy', label='training with dropout')
val_acc2 = hv.Curve((history2.epoch, history2.history['val_acc']), 'epoch', 'a
ccuracy', label='validation with dropout')

(train_acc * val_acc * train_acc2 * val_acc2).redim(accuracy=dict(range=(0.4,
 1.1)))
```

Out[13]:



Here we can see some common features of a model with dropout:

- Training is slower (the noise introduced by dropout interferes with gradient descent)
- The onset of overfitting is delayed (from epoch 2 to epoch 7)
- We achieve a slightly higher accuracy on the validation data with additional training.

Unfortunately, the amount of improvement in this case is still not enough to increase accuracy by more than a few percent. It looks like we need a more complex model.

A More Complex Model

To increase the sophistication of this model, we're going to employ a few strategies:

- Add back the second round of convolutions (more like VGG16)
- Increase the size of the first dense layer

Unfortunately, this is the hardest thing to figure out in practice. Sometimes we need more layers, sometimes we need bigger layers, and sometimes we need a different model entirely. Looking at what others have done is your best guide here until you get some intuition.

In [14]:
```
model3 = Sequential()
model3.add(Conv2D(32, kernel_size=(3, 3), padding='same',
                 activation='relu',
                 input_shape=x_train.shape[1:]))
model3.add(Conv2D(32, (3, 3), activation='relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

# Second layer of convolutions
model3.add(Conv2D(64, kernel_size=(3, 3), padding='same',
                 activation='relu'))
model3.add(Conv2D(64, (3, 3), activation='relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

model3.add(Flatten())
model3.add(Dense(512, activation='relu'))
model3.add(Dropout(0.5))
model3.add(Dense(num_classes, activation='softmax'))

model3.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history3 = model3.fit(x_train, y_train,
          batch_size=128,
          epochs=15,
          verbose=1,
          validation_data=(x_test, y_test))
```

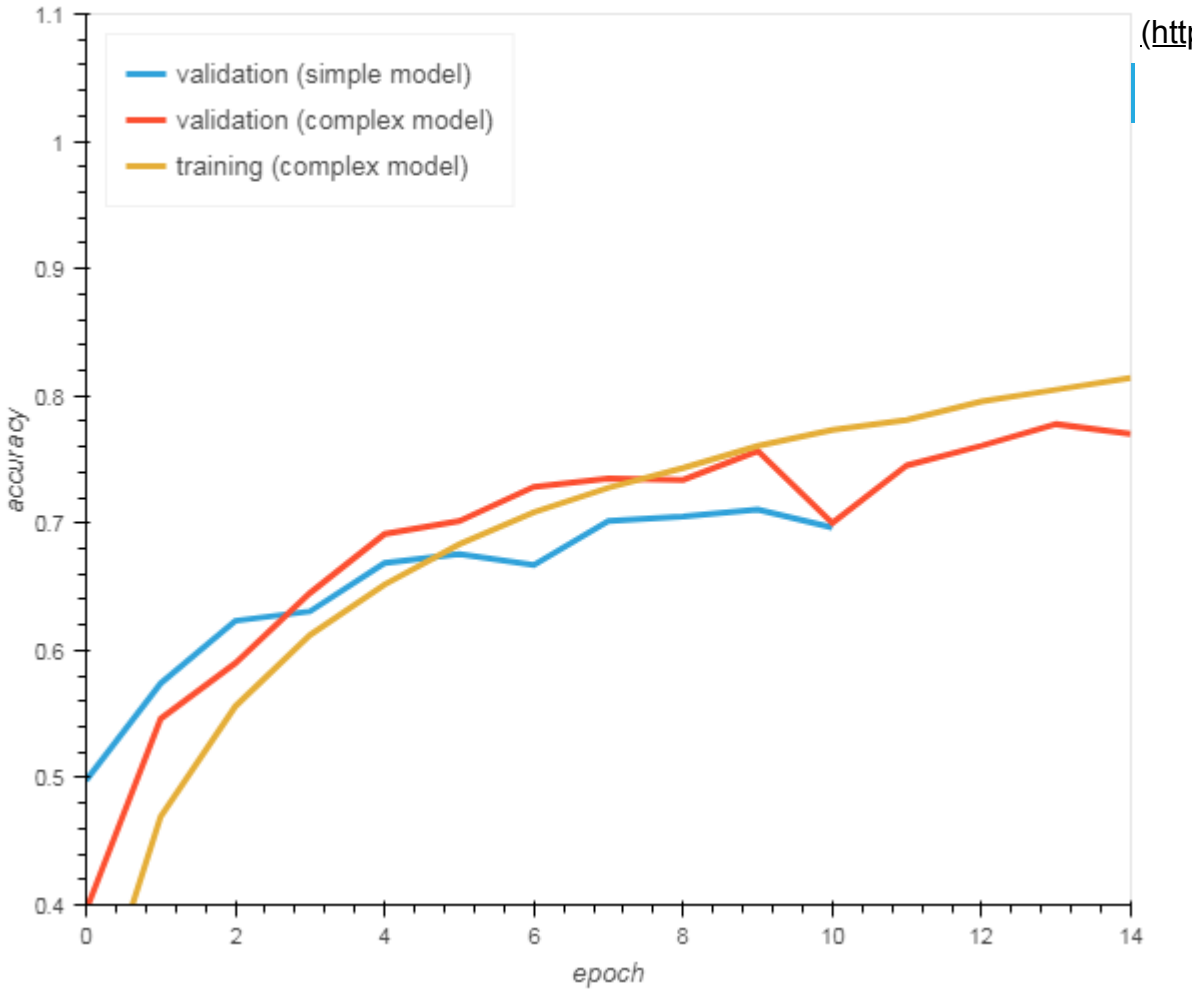
```
Train on 50000 samples, validate on 10000 samples
Epoch 1/15
50000/50000 [==============================] - 16s 329us/step - loss: 1.9558
- acc: 0.2861 - val_loss: 1.6612 - val_acc: 0.3957
Epoch 2/15
50000/50000 [==============================] - 16s 320us/step - loss: 1.4749
- acc: 0.4695 - val_loss: 1.2704 - val_acc: 0.5464
Epoch 3/15
50000/50000 [==============================] - 16s 321us/step - loss: 1.2483
- acc: 0.5560 - val_loss: 1.1467 - val_acc: 0.5899
Epoch 4/15
50000/50000 [==============================] - 16s 320us/step - loss: 1.0971
- acc: 0.6121 - val_loss: 0.9921 - val_acc: 0.6450
Epoch 5/15
50000/50000 [==============================] - 16s 319us/step - loss: 0.9858
- acc: 0.6518 - val_loss: 0.8788 - val_acc: 0.6914
Epoch 6/15
50000/50000 [==============================] - 16s 320us/step - loss: 0.8998
- acc: 0.6834 - val_loss: 0.8663 - val_acc: 0.7015
Epoch 7/15
50000/50000 [==============================] - 16s 320us/step - loss: 0.8321
- acc: 0.7085 - val_loss: 0.7788 - val_acc: 0.7284
Epoch 8/15
50000/50000 [==============================] - 16s 319us/step - loss: 0.7778
- acc: 0.7278 - val_loss: 0.7566 - val_acc: 0.7350
Epoch 9/15
50000/50000 [==============================] - 16s 321us/step - loss: 0.7331
- acc: 0.7432 - val_loss: 0.7655 - val_acc: 0.7339
Epoch 10/15
50000/50000 [==============================] - 16s 320us/step - loss: 0.6837
- acc: 0.7605 - val_loss: 0.7165 - val_acc: 0.7567
Epoch 11/15
50000/50000 [==============================] - 16s 320us/step - loss: 0.6467
- acc: 0.7731 - val_loss: 0.9267 - val_acc: 0.7001
Epoch 12/15
50000/50000 [==============================] - 16s 320us/step - loss: 0.6153
- acc: 0.7811 - val_loss: 0.7627 - val_acc: 0.7454
Epoch 13/15
50000/50000 [==============================] - 16s 315us/step - loss: 0.5832
- acc: 0.7956 - val_loss: 0.7052 - val_acc: 0.7608
Epoch 14/15
50000/50000 [==============================] - 15s 310us/step - loss: 0.5555
- acc: 0.8049 - val_loss: 0.6605 - val_acc: 0.7777
Epoch 15/15
50000/50000 [==============================] - 15s 307us/step - loss: 0.5334
- acc: 0.8141 - val_loss: 0.6862 - val_acc: 0.7699
```

In [15]:
```
%%opts Curve [width=600 height=500]
%%opts Curve (line_width=3)
%%opts Overlay [legend_position='top_left']

train_acc = hv.Curve((history2.epoch, history2.history['val_acc']), 'epoch',
 'accuracy', label='validation (simple model)')
train_acc2 = hv.Curve((history3.epoch, history3.history['acc']), 'epoch', 'a
ccuracy', label='training (complex model)')
val_acc = hv.Curve((history3.epoch, history3.history['val_acc']), 'epoch',
 'accuracy', label='validation (complex model)')

(train_acc * val_acc * train_acc2).redim(accuracy=dict(range=(0.4, 1.1)))
```

Out[15]:



Experiments to Try

- We changed two things to make the more complex model: extra convolutional layers and making the dense layer bigger. Was it necessary to do both?
- Add a callback to the first fit to end training early once the validation data accuracy stops improving.
- What does the confusion matrix look like for the more complex model? Do we still have problems with cats and dogs?

If you screw everything up, you can use File / Revert to Checkpoint to go back to the first version of the notebook and restart the Jupyter kernel with Kernel / Restart.

In [16]:
```
early_stop = keras.callbacks.EarlyStopping(monitor='val_acc', min_delta=0.05,
patience=2, verbose=1)

model3 = Sequential()
model3.add(Conv2D(32, kernel_size=(3, 3), padding='same',
                 activation='relu',
                 input_shape=x_train.shape[1:]))
model3.add(Conv2D(32, (3, 3), activation='relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

# Second layer of convolutions
model3.add(Conv2D(64, kernel_size=(3, 3), padding='same',
                 activation='relu'))
model3.add(Conv2D(64, (3, 3), activation='relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

model3.add(Flatten())
model3.add(Dense(512, activation='relu'))
model3.add(Dropout(0.5))
model3.add(Dense(num_classes, activation='softmax'))

model3.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

history3 = model3.fit(x_train, y_train,
          batch_size=128,
          epochs=15,
          verbose=1,
          validation_data=(x_test, y_test),callbacks=[early_stop])
```

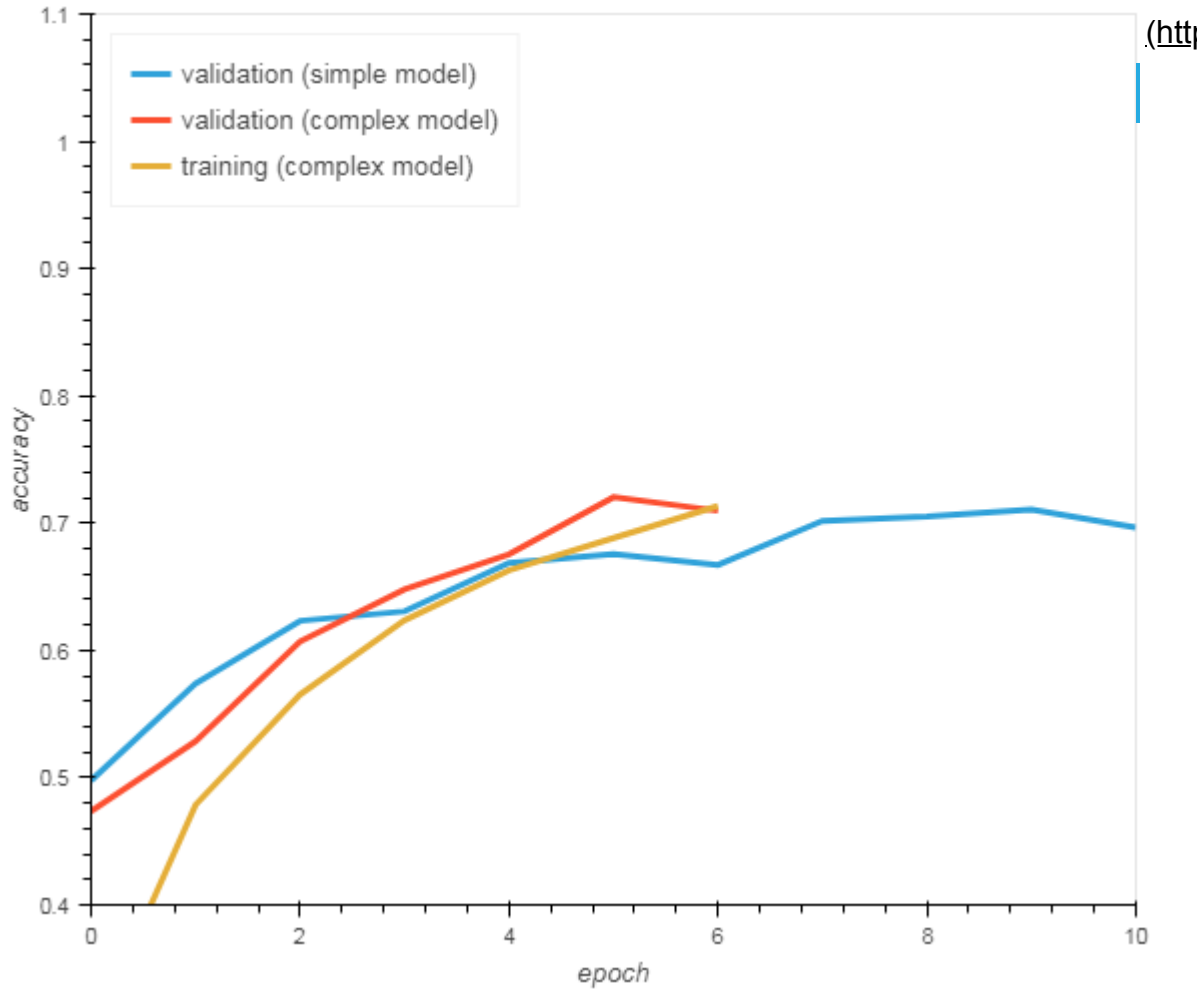
```
Train on 50000 samples, validate on 10000 samples
Epoch 1/15
50000/50000 [==============================] - 16s 317us/step - loss: 1.9292
- acc: 0.2951 - val_loss: 1.4828 - val_acc: 0.4735
Epoch 2/15
50000/50000 [==============================] - 15s 307us/step - loss: 1.4491
- acc: 0.4785 - val_loss: 1.3242 - val_acc: 0.5285
Epoch 3/15
50000/50000 [==============================] - 15s 309us/step - loss: 1.2214
- acc: 0.5652 - val_loss: 1.1013 - val_acc: 0.6069
Epoch 4/15
50000/50000 [==============================] - 15s 308us/step - loss: 1.0643
- acc: 0.6235 - val_loss: 1.0125 - val_acc: 0.6480
Epoch 5/15
50000/50000 [==============================] - 15s 309us/step - loss: 0.9591
- acc: 0.6629 - val_loss: 0.9083 - val_acc: 0.6756
Epoch 6/15
50000/50000 [==============================] - 15s 309us/step - loss: 0.8855
- acc: 0.6883 - val_loss: 0.8035 - val_acc: 0.7203
Epoch 7/15
50000/50000 [==============================] - 15s 307us/step - loss: 0.8208
- acc: 0.7135 - val_loss: 0.8150 - val_acc: 0.7097
Epoch 00007: early stopping
```

In [17]:
```
%%opts Curve [width=600 height=500]
%%opts Curve (line_width=3)
%%opts Overlay [legend_position='top_left']

train_acc = hv.Curve((history2.epoch, history2.history['val_acc']), 'epoch',
 'accuracy', label='validation (simple model)')
train_acc2 = hv.Curve((history3.epoch, history3.history['acc']), 'epoch', 'acc
uracy', label='training (complex model)')
val_acc = hv.Curve((history3.epoch, history3.history['val_acc']), 'epoch', 'ac
curacy', label='validation (complex model)')

(train_acc * val_acc * train_acc2).redim(accuracy=dict(range=(0.4, 1.1)))
```

Out[17]:



In [20]:
```
from sklearn.metrics import confusion_matrix

y_pred = model.predict_classes(x_test)
confuse = confusion_matrix(y_test_true, y_pred)
```

In [21]:
```
# Holoviews hack to tilt labels by 45 degrees
from math import pi
def angle_label(plot, element):
    plot.state.xaxis.major_label_orientation = pi / 4
```

In [22]:
```
%%opts HeatMap [width=500 height=400 tools=['hover'] finalize_hooks=[angle_lab
el]]
hv.HeatMap((cifar10_labels, cifar10_labels, confuse)).redim.label(x='true', y
='predict')
```

Out[22]: