Part 4a - Saving a Model

In this notebook we will cover the following topics:

- Saving a model to disk

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

In [2]:
```
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 bel
ow
cifar10_labels = np.array(['airplane', 'automobile', 'bird', 'cat', 'deer',
                           'dog', 'frog', 'horse', 'ship', 'truck'])
```

Train the Model

Let's quickly train our simple model so we can save the results

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(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
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=10,
          verbose=1,
          validation_data=(x_test, y_test))
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 13s 260us/step - loss: 1.8158
- acc: 0.3433 - val_loss: 1.3895 - val_acc: 0.5053
Epoch 2/10
50000/50000 [==============================] - 12s 232us/step - loss: 1.3872
- acc: 0.5058 - val_loss: 1.1614 - val_acc: 0.5863
Epoch 3/10
50000/50000 [==============================] - 12s 232us/step - loss: 1.2121
- acc: 0.5728 - val_loss: 1.0876 - val_acc: 0.6228
Epoch 4/10
50000/50000 [==============================] - 12s 232us/step - loss: 1.0956
- acc: 0.6164 - val_loss: 0.9977 - val_acc: 0.6480
Epoch 5/10
50000/50000 [==============================] - 12s 233us/step - loss: 1.0074
- acc: 0.6476 - val_loss: 1.0207 - val_acc: 0.6457
Epoch 6/10
50000/50000 [==============================] - 12s 232us/step - loss: 0.9396
- acc: 0.6731 - val_loss: 0.8923 - val_acc: 0.6937
Epoch 7/10
50000/50000 [==============================] - 12s 233us/step - loss: 0.8795
- acc: 0.6910 - val_loss: 0.9609 - val_acc: 0.6625
Epoch 8/10
50000/50000 [==============================] - 12s 232us/step - loss: 0.8348
- acc: 0.7081 - val_loss: 0.9148 - val_acc: 0.6880
Epoch 9/10
50000/50000 [==============================] - 12s 231us/step - loss: 0.7808
- acc: 0.7263 - val_loss: 0.9091 - val_acc: 0.6881
Epoch 10/10
50000/50000 [==============================] - 12s 232us/step - loss: 0.7347
- acc: 0.7439 - val_loss: 0.8508 - val_acc: 0.7108
```

Saving the model is as easy as calling the `save()` method. This records the weights and the structure of the model so that it can be recreated by another program entirely from this file. (Assuming that program is using the same version of Keras.)

In [12]:
```
model.save('cifar10_model.hdf5')
```

Note that the file format is HDF5, which is a common data format for numerical data. Keras requires the `h5py` Python package be present in order to read and write HDF5 files.

Depending on the number of weights in the model, this file can get very big:

In [7]:
```
! ls -lh cifar10_model.hdf5
```

```
-rw-r--r-- 1 jovyan users 19M Apr  8 20:51 cifar10_model.hdf5
```

We can poke around to see the structure of it using the HDF5 command line tools:

In [8]:
```
! h5ls -r cifar10_model.hdf5
```

```
/                        Group
/model_weights           Group
/model_weights/conv2d_3  Group
/model_weights/conv2d_3/conv2d_3 Group
/model_weights/conv2d_3/conv2d_3/bias:0 Dataset {32}
/model_weights/conv2d_3/conv2d_3/kernel:0 Dataset {3, 3, 3, 32}
/model_weights/conv2d_4  Group
/model_weights/conv2d_4/conv2d_4 Group
/model_weights/conv2d_4/conv2d_4/bias:0 Dataset {64}
/model_weights/conv2d_4/conv2d_4/kernel:0 Dataset {3, 3, 32, 64}
/model_weights/dense_3   Group
/model_weights/dense_3/dense_3 Group
/model_weights/dense_3/dense_3/bias:0 Dataset {128}
/model_weights/dense_3/dense_3/kernel:0 Dataset {12544, 128}
/model_weights/dense_4   Group
/model_weights/dense_4/dense_4 Group
/model_weights/dense_4/dense_4/bias:0 Dataset {10}
/model_weights/dense_4/dense_4/kernel:0 Dataset {128, 10}
/model_weights/dropout_3 Group
/model_weights/dropout_4 Group
/model_weights/flatten_2 Group
/model_weights/max_pooling2d_2 Group
/optimizer_weights       Group
/optimizer_weights/training Group
/optimizer_weights/training/Adadelta Group
/optimizer_weights/training/Adadelta/Variable:0 Dataset {3, 3, 3, 32}
/optimizer_weights/training/Adadelta/Variable_10:0 Dataset {3, 3, 32, 64}
/optimizer_weights/training/Adadelta/Variable_11:0 Dataset {64}
/optimizer_weights/training/Adadelta/Variable_12:0 Dataset {12544, 128}
/optimizer_weights/training/Adadelta/Variable_13:0 Dataset {128}
/optimizer_weights/training/Adadelta/Variable_14:0 Dataset {128, 10}
/optimizer_weights/training/Adadelta/Variable_15:0 Dataset {10}
/optimizer_weights/training/Adadelta/Variable_1:0 Dataset {32}
/optimizer_weights/training/Adadelta/Variable_2:0 Dataset {3, 3, 32, 64}
/optimizer_weights/training/Adadelta/Variable_3:0 Dataset {64}
/optimizer_weights/training/Adadelta/Variable_4:0 Dataset {12544, 128}
/optimizer_weights/training/Adadelta/Variable_5:0 Dataset {128}
/optimizer_weights/training/Adadelta/Variable_6:0 Dataset {128, 10}
/optimizer_weights/training/Adadelta/Variable_7:0 Dataset {10}
/optimizer_weights/training/Adadelta/Variable_8:0 Dataset {3, 3, 3, 32}
/optimizer_weights/training/Adadelta/Variable_9:0 Dataset {32}
```

Interestingly, we can see that the HDF5 file also records the state of the optimizer, so that we can resume training from a saved model. This is a useful way to checkpoint your work. In fact, Keras has a ModelCheckpoint callback (https://keras.io/callbacks/#modelcheckpoint) that does this automatically after every epoch.

Experiments to Try

- Try using the `ModelCheckpoint` callback to save the model in every epoch.

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 [15]:
```
keras.callbacks.ModelCheckpoint('cifar10_model_chkpoint.hdf5', monitor='val_loss', verbose=0, save_best_only=False, save_weights_only=False, mode='auto', period=1)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-15-718e95cb9a42> in <module>()
----> 1 history.callbacks.ModelCheckpoint('cifar10_model_chkpoint.hdf5', monitor='val_loss', verbose=0, save_best_only=False, save_weights_only=False, mode='auto', period=1)

AttributeError: 'History' object has no attribute 'callbacks'
```