SDSC Summer Institute
Deep Learning

SDSC SAN DIEGO SUPERCOMPUTER CENTER

UC San Diego

CNN Transfer Learning Hands-On

Mai H. Nguyen, Ph.D.

Transfer Learning Hands-On

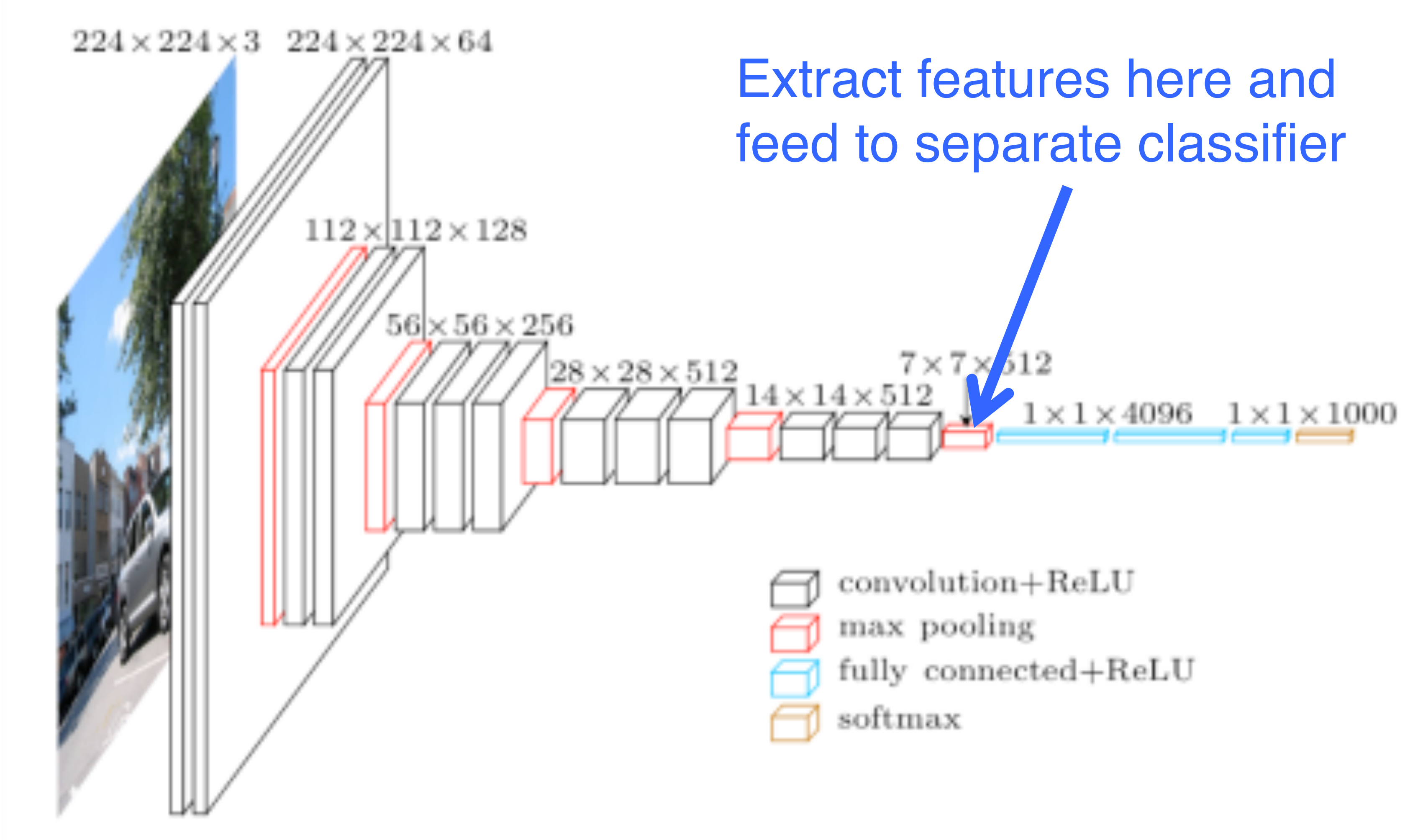
- **Data**
  - Cats and dogs images from Kaggle

- **Exercises**
  - Feature extraction
    - Use pre-trained CNN to extract features from images
    - Train neural network to classify cats/dogs using extract features
  - Fine tune
    - Adjust weights of last few layers of pre-trained CNN through training

Feature Extraction

- **Data**
  - Cats and dogs images from Kaggle
- **Method**
  - Use VGG16 trained on ImageNet data as pre-trained model. Remove last fully connected layer.
  - Extract features from pre-trained model and save
  - Neural network then trained on extracted features to classify cats vs. dogs

Transfer Learning – Feature Extraction



Extract features here and feed to separate classifier

Source: https://www.cs.toronto.edu/~frossard/post/vgg16/

Get Latest from Github Repo

- **If haven’t cloned Summer Institute repo**
  - git clone <URL>
- **If already cloned Summer Institute repo**
  - git pull <URL>
- **<URL>**

https://github.com/sdsc/sdsc-summer-institute-2019

Server Setup

- **Go to keras directory**
  - *cd <SI2019_dir>/datasci4_deep_learning/keras*
- **Request GPU node and start jupyter**
  - *sbatch --res=SI2019DAY4 keras.slrm*

Set Link to Data

- **Create soft link to data**
  - ln –s ~/ml/data data

- **Look at dataset**
  - ls –l data/train/cats/* | wc
  - ls –l data/train/dogs/* | wc
  - ls –l data/validation/cats/* | wc
  - ls –l data/validation/dogs/* | wc

Server Setup (cont.)

- **Check queue**
  - *squeue –u $USER*
  - Note compute node name (e.g., comet-14-43)
    - 18223060 compute bash <user> R 1:19 1 comet-14-43
- **Check that Jupyter Notebook has started**
  - ls –l keras*.out

-rw-r--r-- 1 <user> sds148 840 Aug 4 09:32 keras.18317995.comet-14-43.out

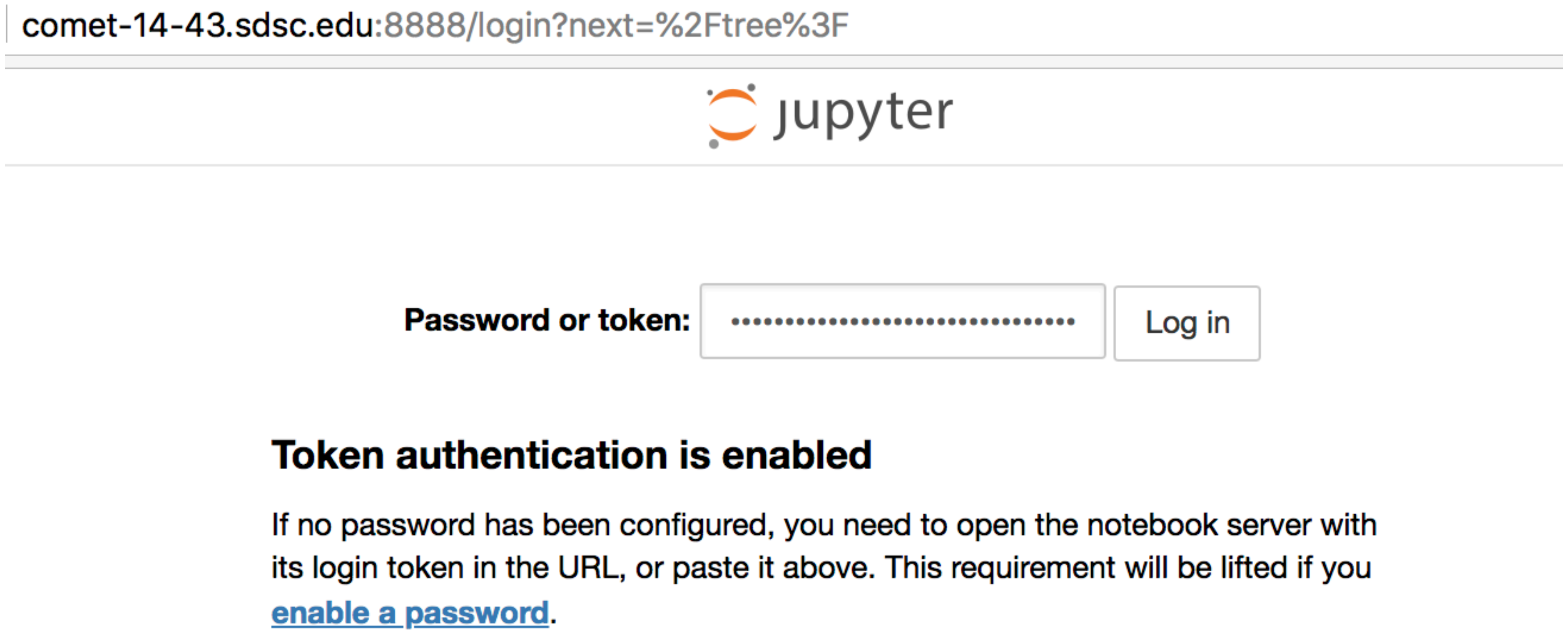
Should be > 0

  - tail keras*.out
  - Copy token
    - *Copy/paste this URL into your browser when you connect for the first time, to login with a token:*

http://localhost:8888/?token=395fe3d456cb951e34ef125adf27e6a62

Browser Setup

- **In browser, type:**
  - comet-xx-xx.sdsc.edu:8888
  - Paste token

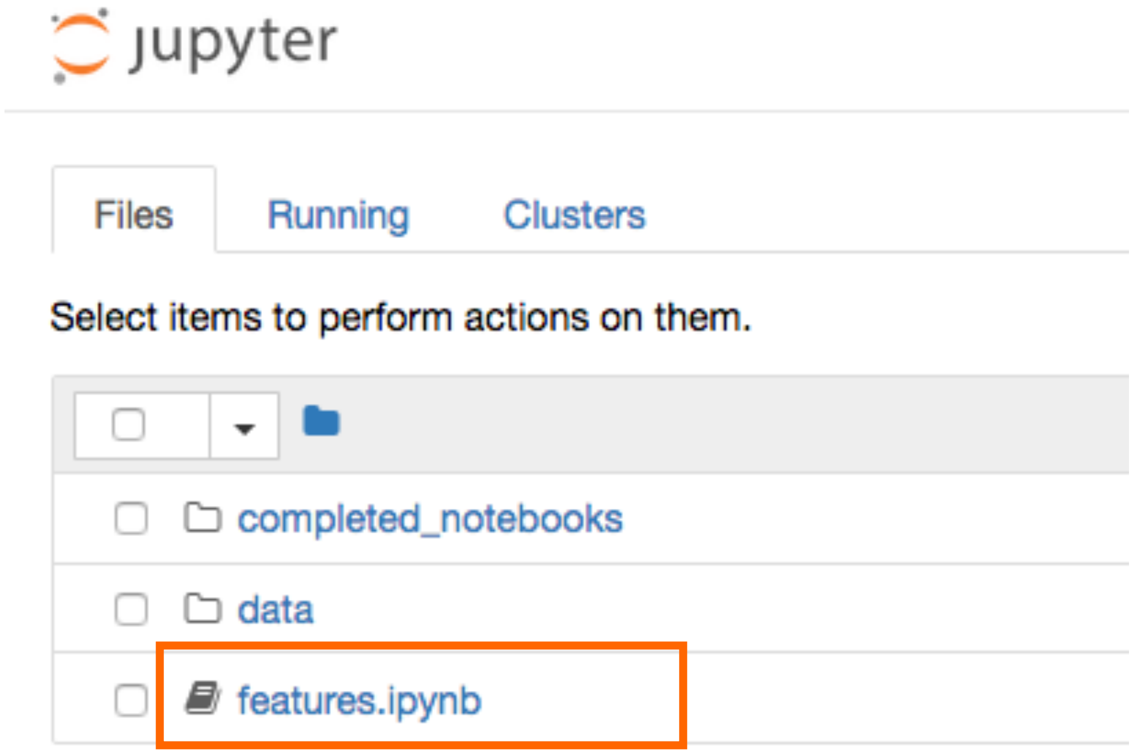


comet-14-43.sdsc.edu:8888/login?next=%2Ftree%3F

Token authentication is enabled

If no password has been configured, you need to open the notebook server with its login token in the URL, or paste it above. This requirement will be lifted if you enable a password.

Open features.ipynb Notebook



jupyter

Files Running Clusters

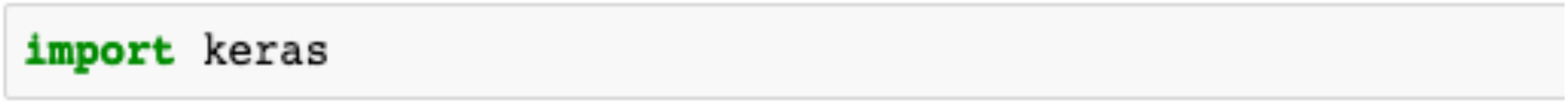
Select items to perform actions on them.

completed_notebooks

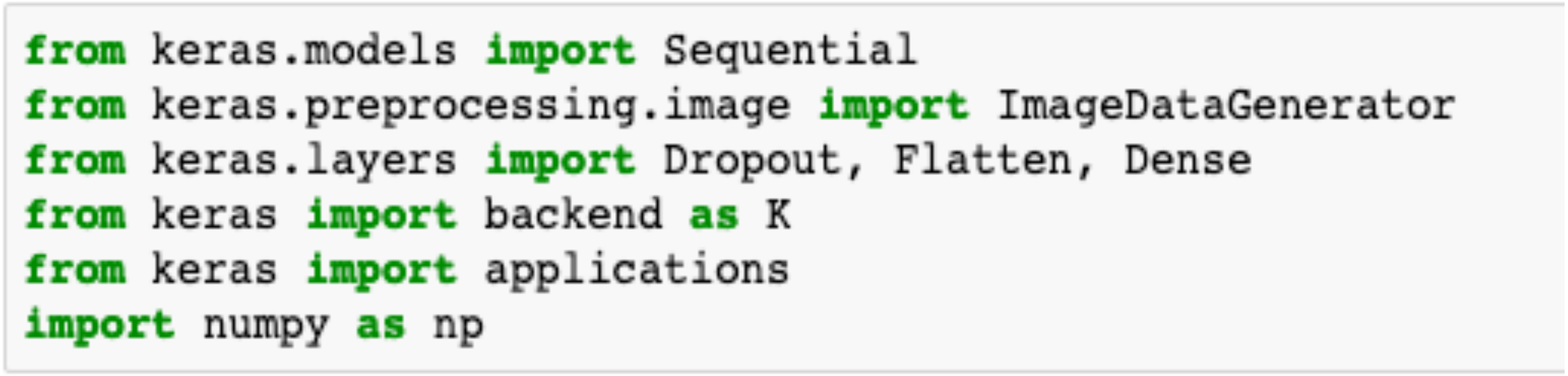
data

features.ipynb

Import Modules

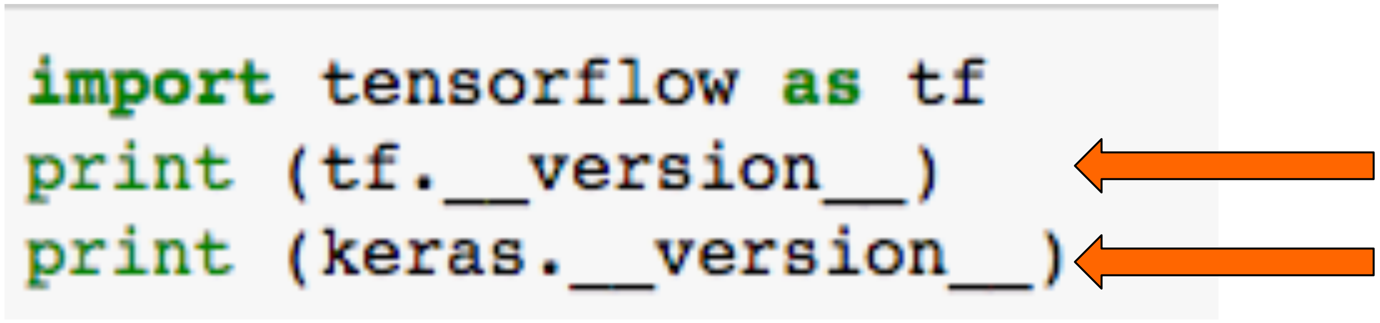


```python
import keras
```



```python
from keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dropout, Flatten, Dense
from keras import backend as K
from keras import applications
import numpy as np
```

Print Keras & TensorFlow Versions



```
import tensorflow as tf
print (tf.__version__)
print (keras.__version__)
```

Set Data Parameters

- Set image dimensions
  - *img_width, img_height = 150, 150*
- Set data location
  - *train_data_dir = ‘data/train’*
  - *validation_data_dir = ‘data/validation’*
- Set number of images
  - *nb_train_samples = 2000*
  - *nb_validation_samples = 800*

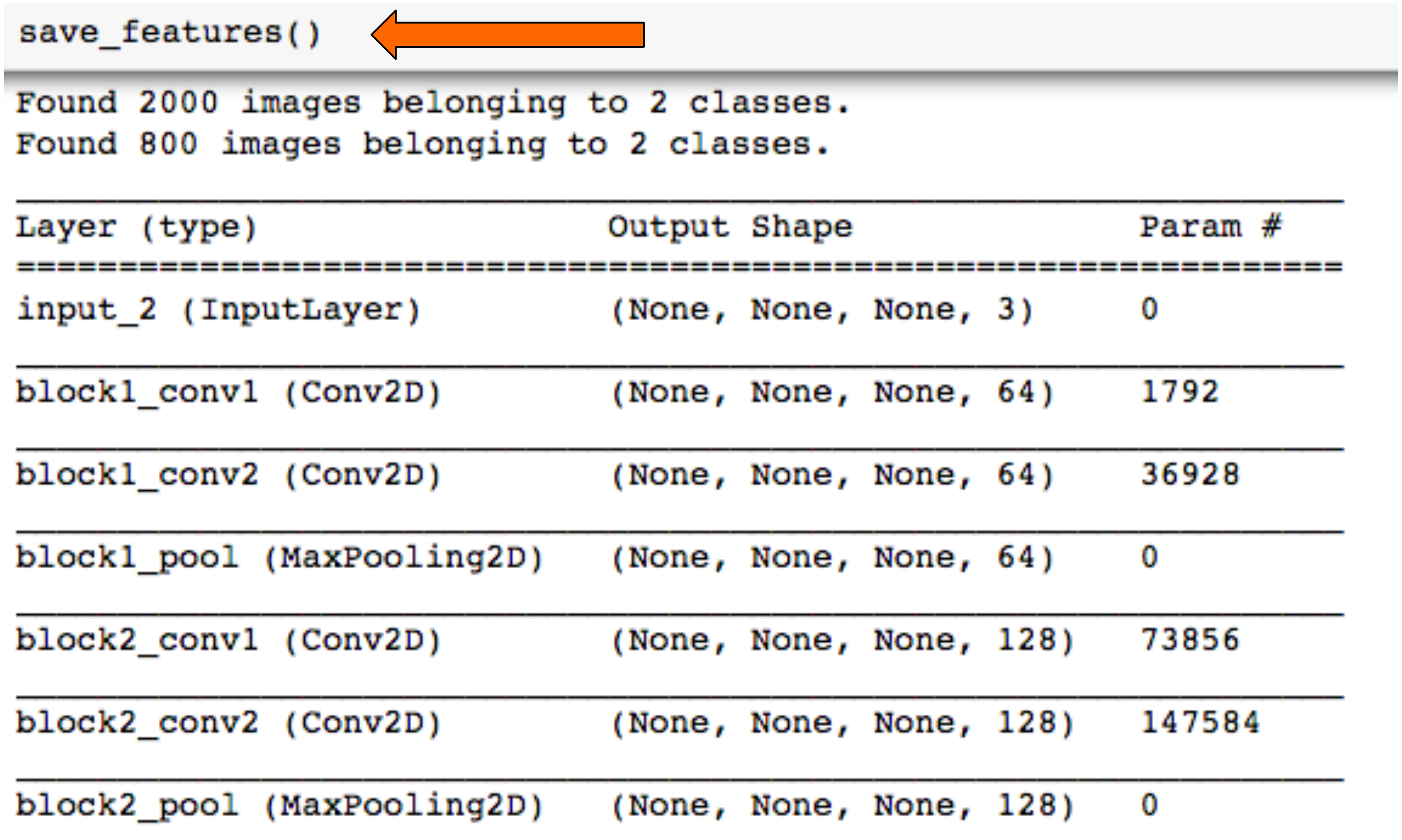
(150, 150, 3)

Method to Extract Features from Pre-Trained Network

```
def save_features():
        …
```

1. Scale pixel values in each image
2. Load weights for pre-trained network without top classifier
3. Generator reads images from subdir, batch_size number of images at a time.
4. Feed images through pre-trained network and extract features
5. Save features
6. Repeat 3-5 for validation data

Call Method to Extract & Save Features



```
save_features()
```

```
Found 2000 images belonging to 2 classes.
Found 800 images belonging to 2 classes.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, None, None, 3)     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, None, None, 64)    36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, None, None, 64)    0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, None, None, 128)   73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, None, None, 128)   147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, None, None, 128)   0
```

Load Saved Features

- **Add name of file containing saved features**
  - For train data

    train_data = np.load ('features_train.npy')

  - For validation data

    validation_data = np.load ('features_validation.npy')

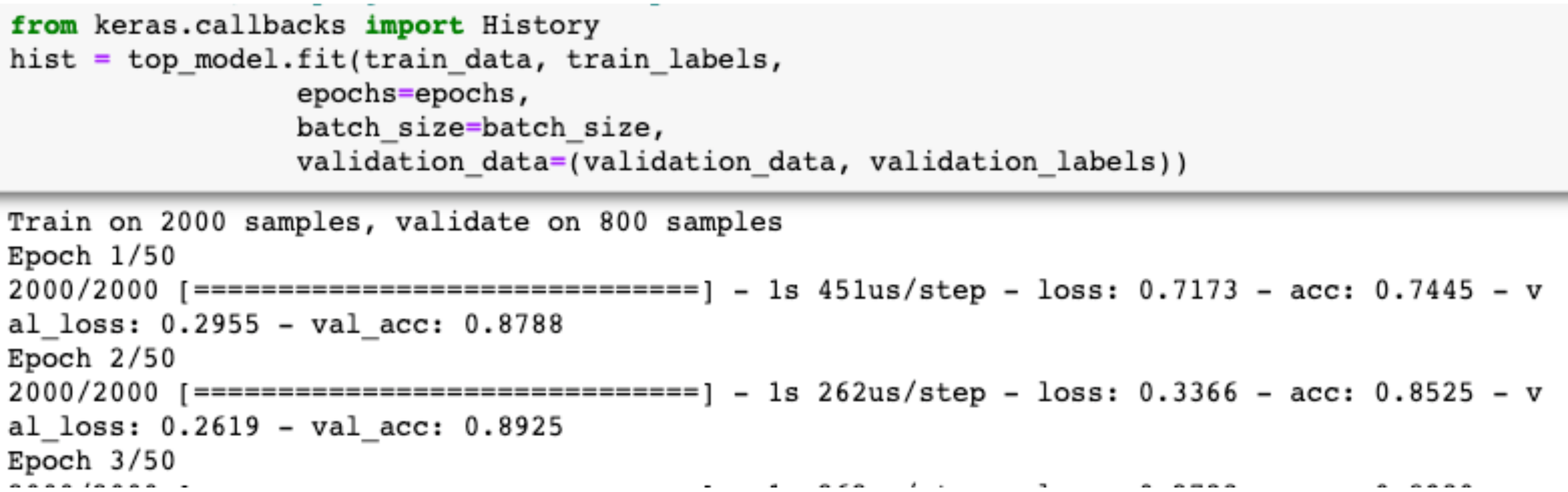
(2000,) (800,)

Create Top Model to Classify Extracted Features

- **Model**
  - Fully connected layer from input to hidden
    - 256 nodes in hidden layer
    - Rectified linear activation function
  - Fully connected layer from hidden to output
    - 1 node in output layer (cat or dog)
    - Sigmoid activation function

Train Top Model

- **Set number of training iterations**
  - epochs = 50
- **Train model, keeping track of history**

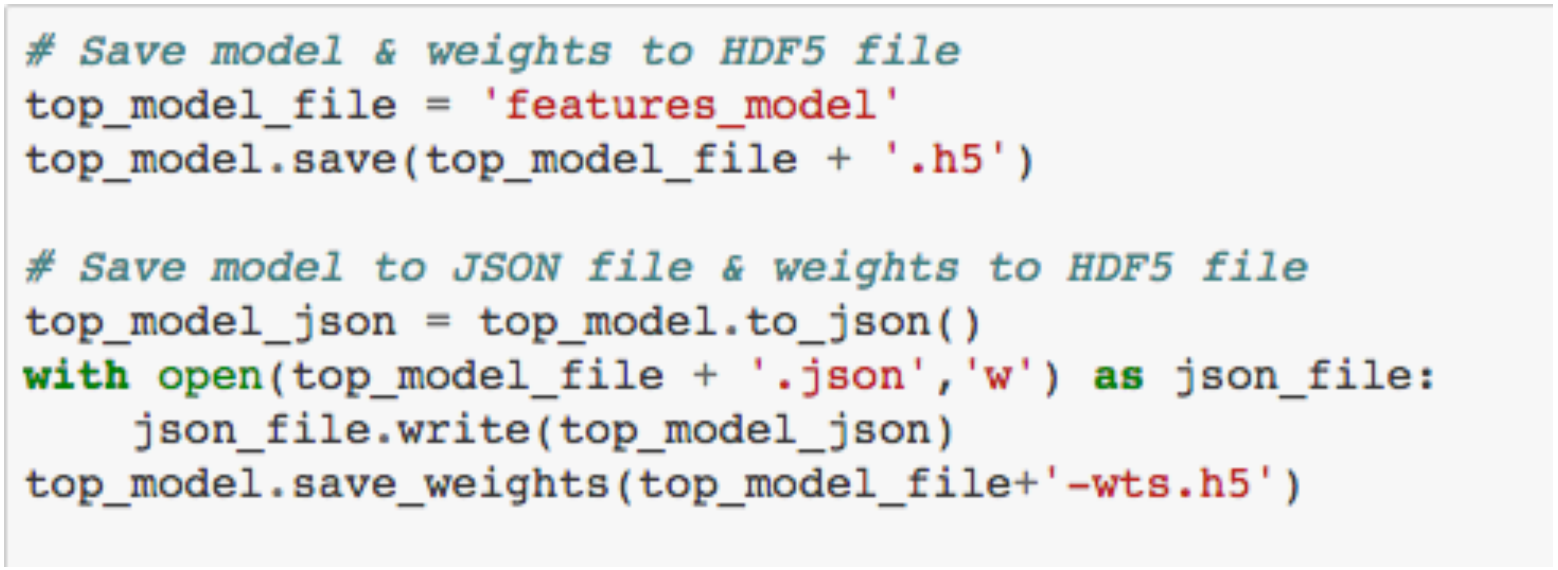


```
from keras.callbacks import History
hist = top_model.fit(train_data, train_labels,
                     epochs=epochs,
                     batch_size=batch_size,
                     validation_data=(validation_data, validation_labels))
```

```
Train on 2000 samples, validate on 800 samples
Epoch 1/50
2000/2000 [==============================] - 1s 451us/step - loss: 0.7173 - acc: 0.7445 - v
al_loss: 0.2955 - val_acc: 0.8788
Epoch 2/50
2000/2000 [==============================] - 1s 262us/step - loss: 0.3366 - acc: 0.8525 - v
al_loss: 0.2619 - val_acc: 0.8925
Epoch 3/50
```

Save Model and Weights

- **Add name for model files**
  - top_model_file = ‘features_model’
- **Save model and weights**

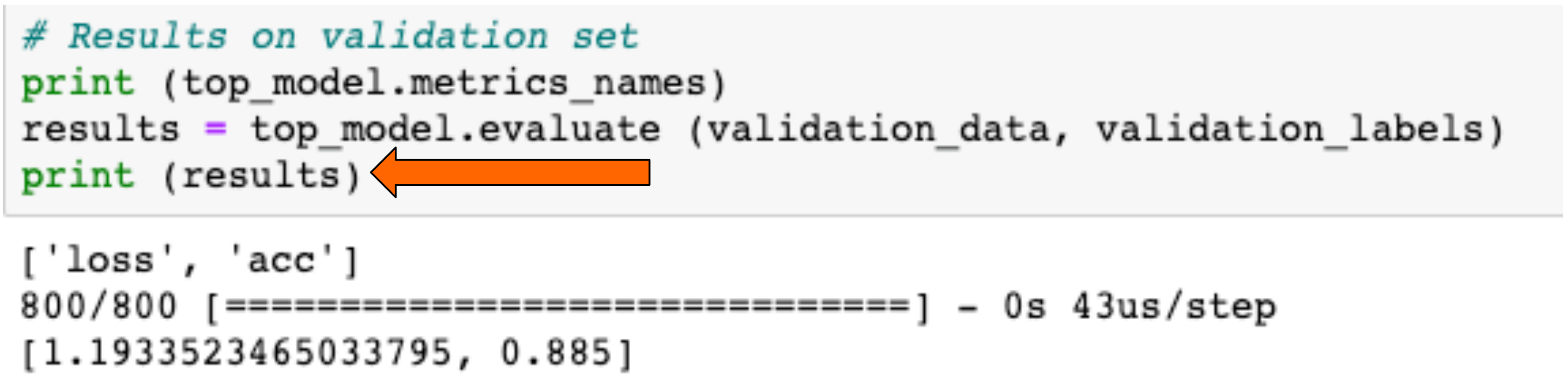


```
# Save model & weights to HDF5 file
top_model_file = 'features_model'
top_model.save(top_model_file + '.h5')

# Save model to JSON file & weights to HDF5 file
top_model_json = top_model.to_json()
with open(top_model_file + '.json','w') as json_file:
    json_file.write(top_model_json)
top_model.save_weights(top_model_file+'-wts.h5')
```

Test Model on Validation Data

- **Get prediction results on validation data**



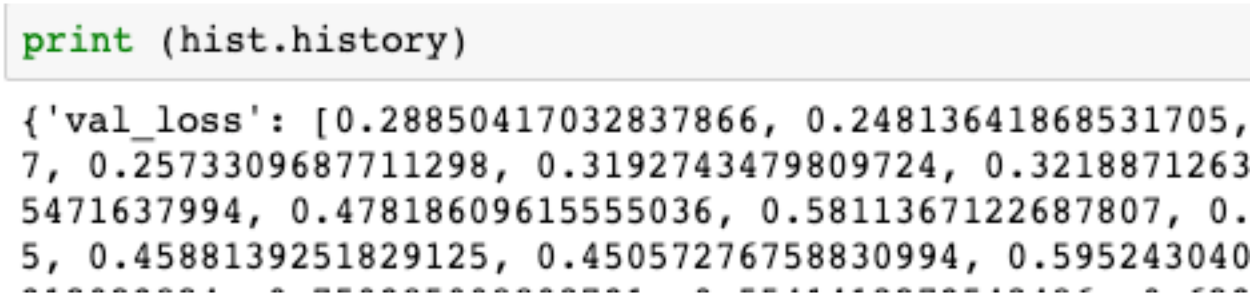
```
# Results on validation set
print (top_model.metrics_names)
results = top_model.evaluate (validation_data, validation_labels)
print (results)
```

```
['loss', 'acc']
800/800 [==============================] - 0s 43us/step
[1.1933523465033795, 0.885]
```

- **Load model again and re-test**
  - Results should be the same
- **Validation accuracy on CNN trained from scratch**
  - ~80%

Print History & Plot Performance Measures

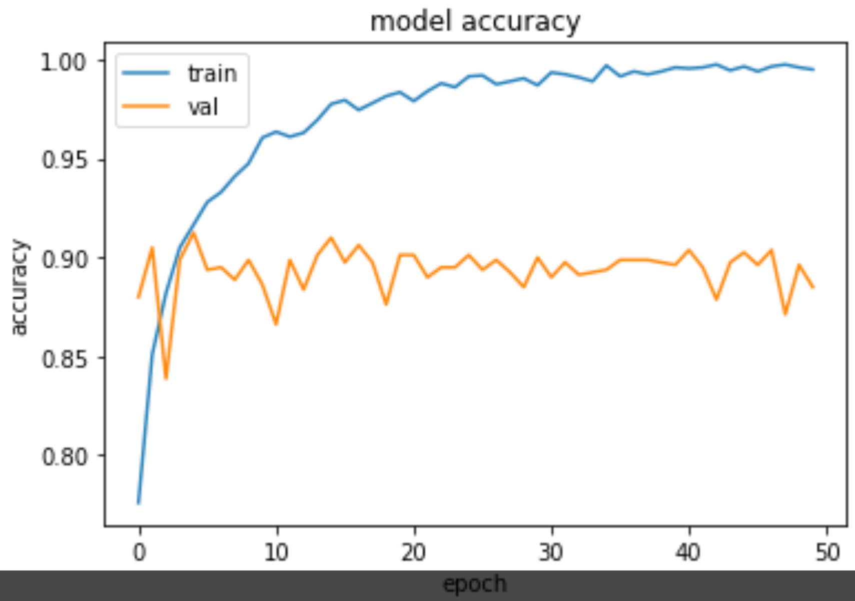
- **Print training history**



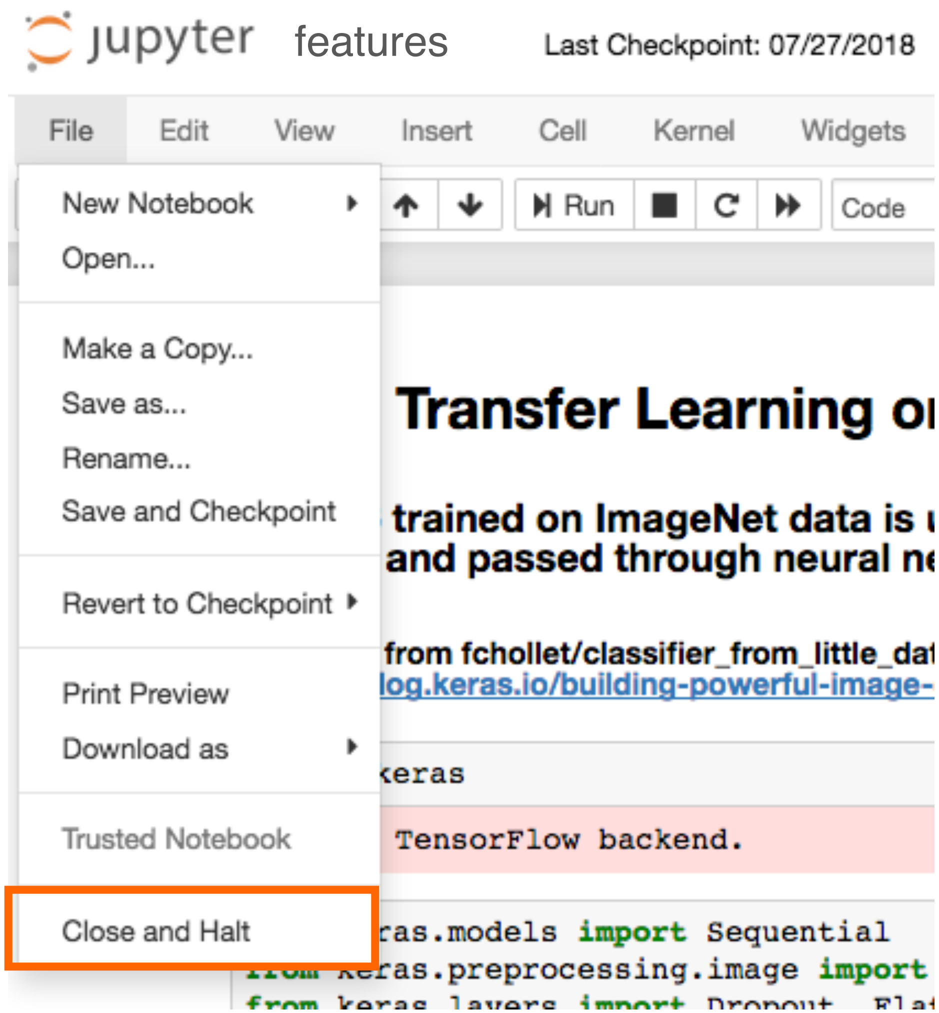
```
print (hist.history)
```

```
{'val_loss': [0.28850417032837866, 0.24813641868531705,
7, 0.2573309687711298, 0.3192743479809724, 0.3218871263
5471637994, 0.47818609615555036, 0.5811367122687807, 0.
5, 0.4588139251829125, 0.45057276758830994, 0.595243040
```

- **Plot accuracy**



Exit Notebook



Transfer Learning o

trained on ImageNet data is
and passed through neural n

from fchollet/classifier_from_little_da
log.keras.io/building-powerful-image-

```
keras
```

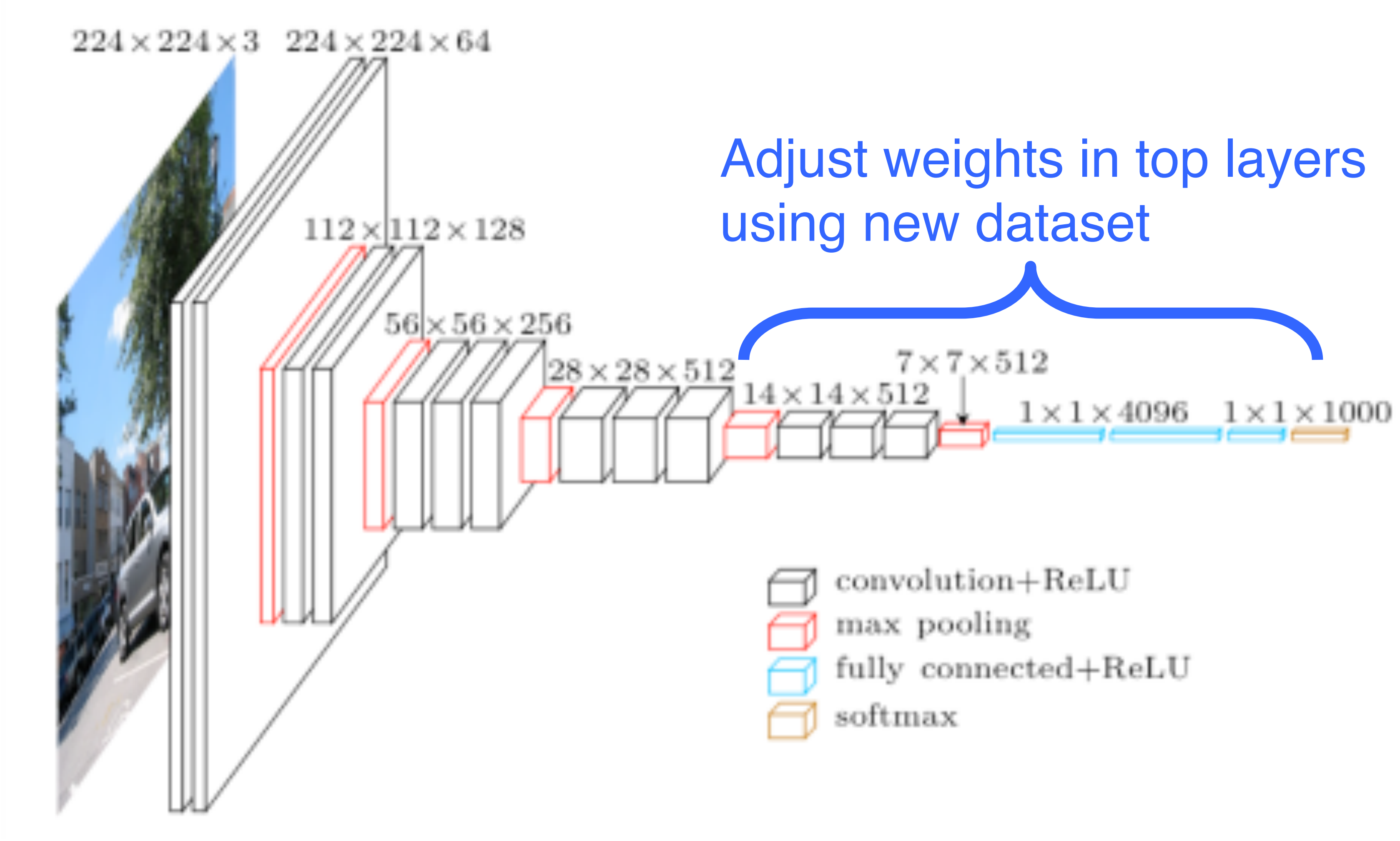
```
TensorFlow backend.
```

```
ras.models import Sequential
from keras.preprocessing.image import
from keras.layers import Dropout, Flat
```

Fine Tuning Hands-On

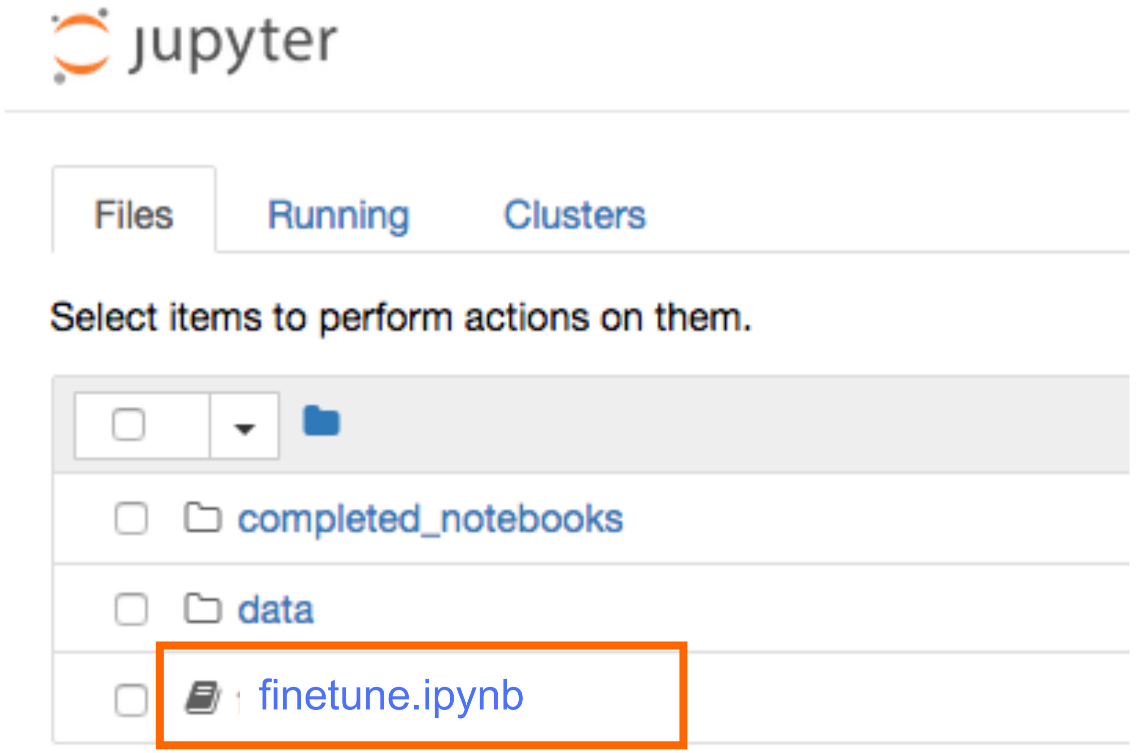
- **Data**
  - Cats and dogs images from Kaggle
- **Method**
  - Use VGG16 trained on ImageNet data as pre-trained model.
  - Replace last fully connected layer with neural network trained from Feature Extraction hands-on.
  - Fine tune last convolution block and fully connected layer.

Transfer Learning – Fine Tuning



Source: https://www.cs.toronto.edu/~frossard/post/vgg16/

Open fine-tune.ipynb Notebook



Select items to perform actions on them.

completed_notebooks
data
finetune.ipynb

Fine Tuning

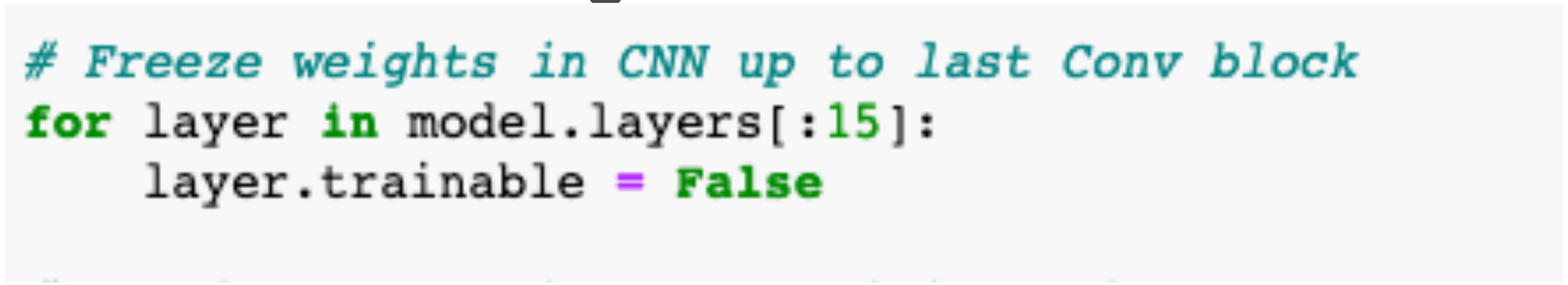
- **Original Model**

  Total params: 14,714,688

  Trainable params: 14,714,688

  Non-trainable params: 0

- **Freeze some weights**



```
# Freeze weights in CNN up to last Conv block
for layer in model.layers[:15]:
    layer.trainable = False
```

  Total params: 16,812,353

  Trainable params: 9,177,089

  Non-trainable params: 7,635,264

Fine Tuning Results

- **Before fine tuning**
  - [loss, accuracy]:
    [0.641176361694085, 0.925]
    [1.1933523442077918, 0.885]
- **After fine tuning**
  - Train adjustable parameters for 5 epochs
  - [loss, accuracy]:
    [0.07257955298712478, 0.978]
    [0.29664344725897535, 0.9125]

Clean Up

- **Exit notebook**
  - File -> Close and Halt
- **Exit Jupyter Notebook**
  - Click on ‘Logout’

References

- **The Keras Blog**
  - https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
- **Code**
  - Feature extraction
    - https://gist.github.com/fchollet/f35fbc80e066a49d65f1688a7e99f069
  - Fine tuning
    - https://gist.github.com/fchollet/7eb39b44eb9e16e59632d25fb3119975

Questions?