Transfer Learning with CNN

Mai H. Nguyen, Ph.D.

What is Transfer Learning?

- **To overcome challenges of training model from scratch:**
  - Insufficient data
  - Very long training time
- **Use pre-trained model**
  - Trained on another dataset
  - This serves as starting point for model
  - Then train model on current dataset for current task

Transfer Learning Approaches

- **Feature extraction**
  - Remove last fully connected layer from pre-trained model
  - Treat rest of network as feature extractor
  - Use features to train new classifier (“top model”)
- **Fine tuning**
  - Tune weights in some layers of original model (along with weights of top model)
  - Train model for current task using new dataset

CNNs for Transfer Learning

- **Popular architectures**
  - AlexNet
  - GoogLeNet
  - VGGNet
  - ResNet
- **All winners of ILSVRC**
  - ImageNet Large Scale Visual Recognition Challenge
  - Annual competition on vision tasks on ImageNet data

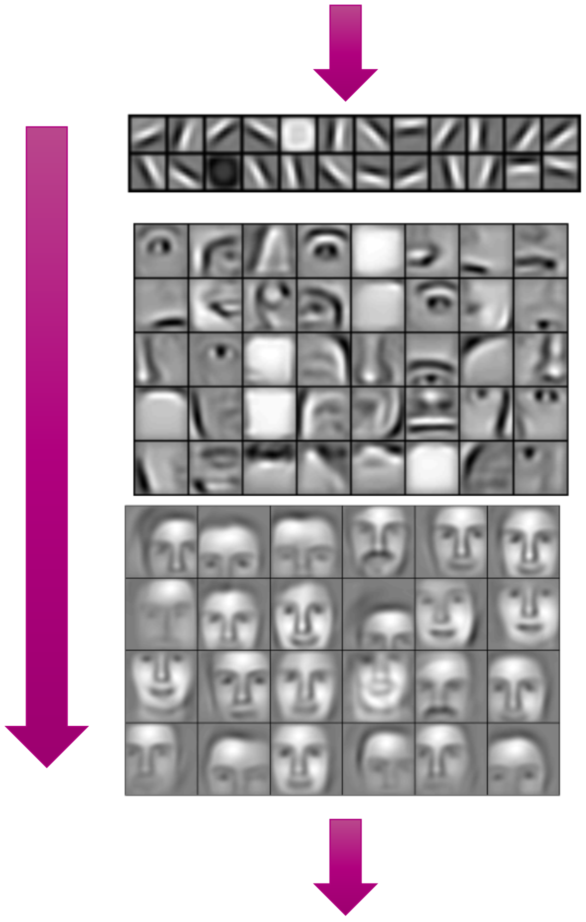
ImageNet

- **Database**
  - Developed for computer vision research
  - > 14,000,000 images hand-annotated
  - > 22,000 categories
- **ILSVRC History**
  - Started in 2010
  - Image classification task: 1,000 object categories
  - Image classification error rate
    - 2011: ~25%   (conventional image processing techniques)
    - 2012: 15.3% (AlexNet)
    - 2015:   3.57% (ResNet; better than human performance)
    - 2016:   2.99% (16.7% error reduction)
    - 2017:   2.25% (23.3% error reduction)

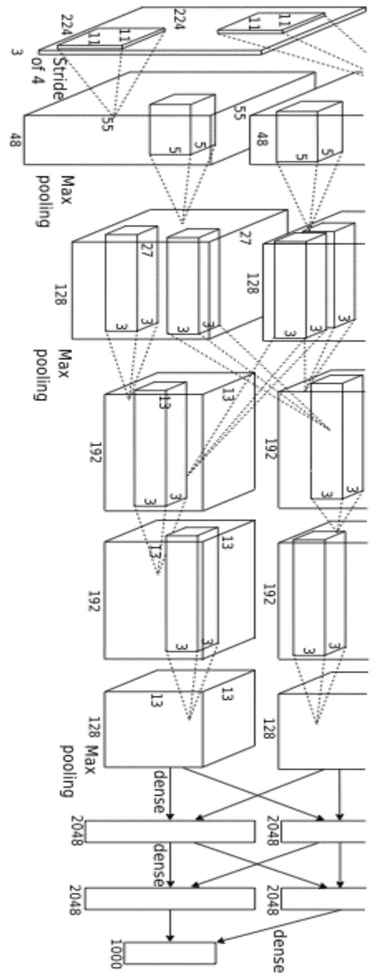
Why Does Transfer Learning Work?

Input

Learned hierarchy

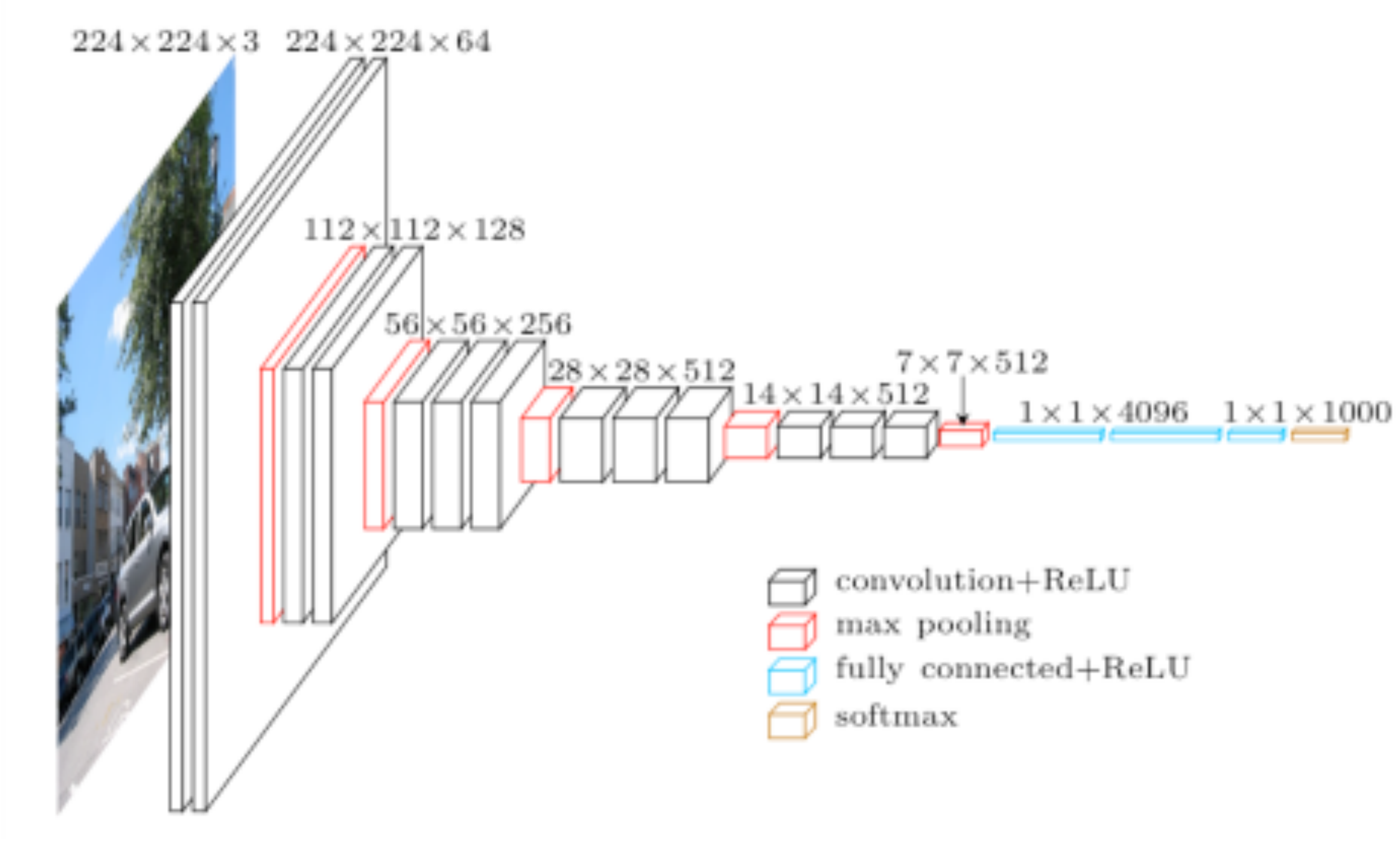


Output



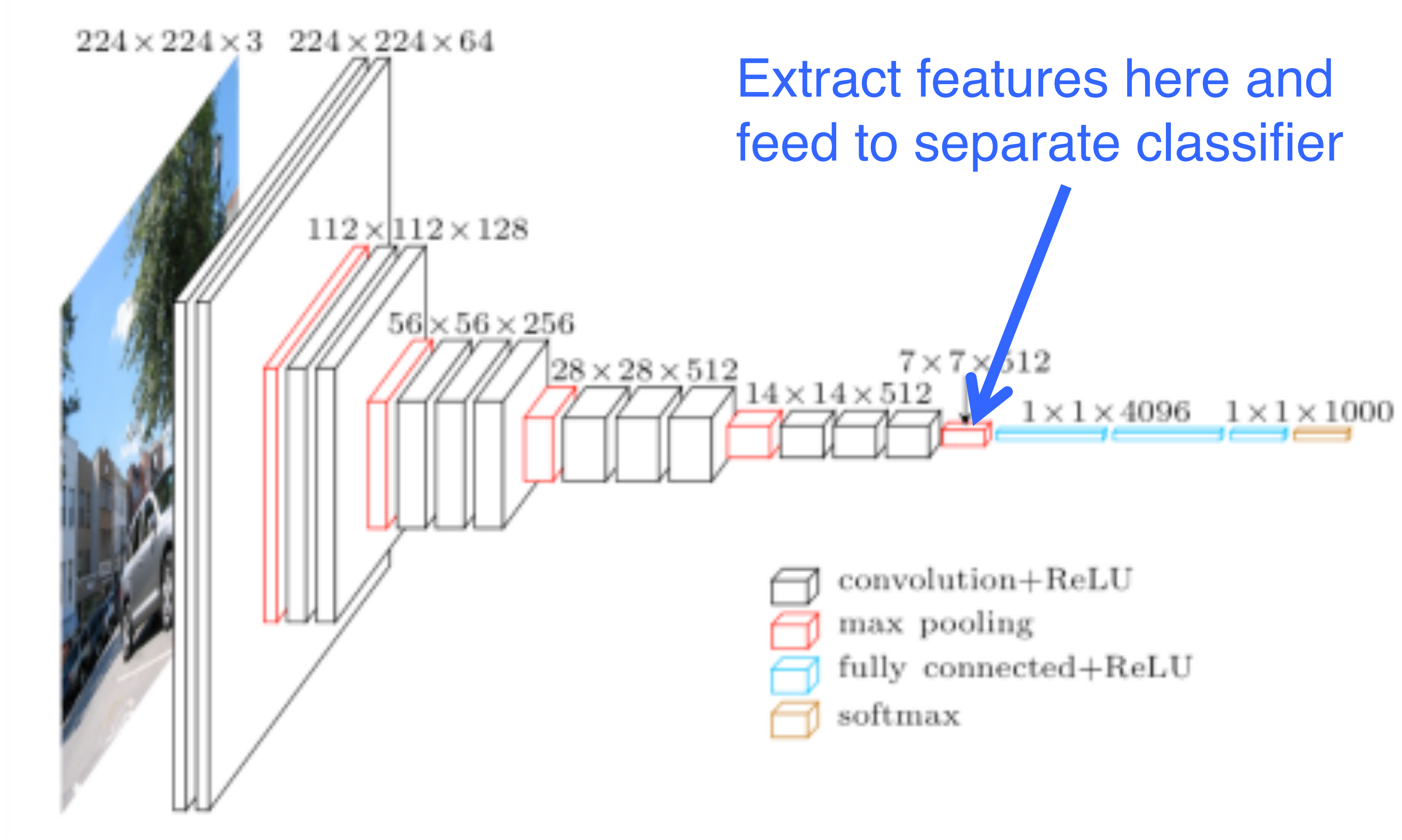
Lee et al. 'Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations' ICML 2009

VGG as Pre-Trained Network



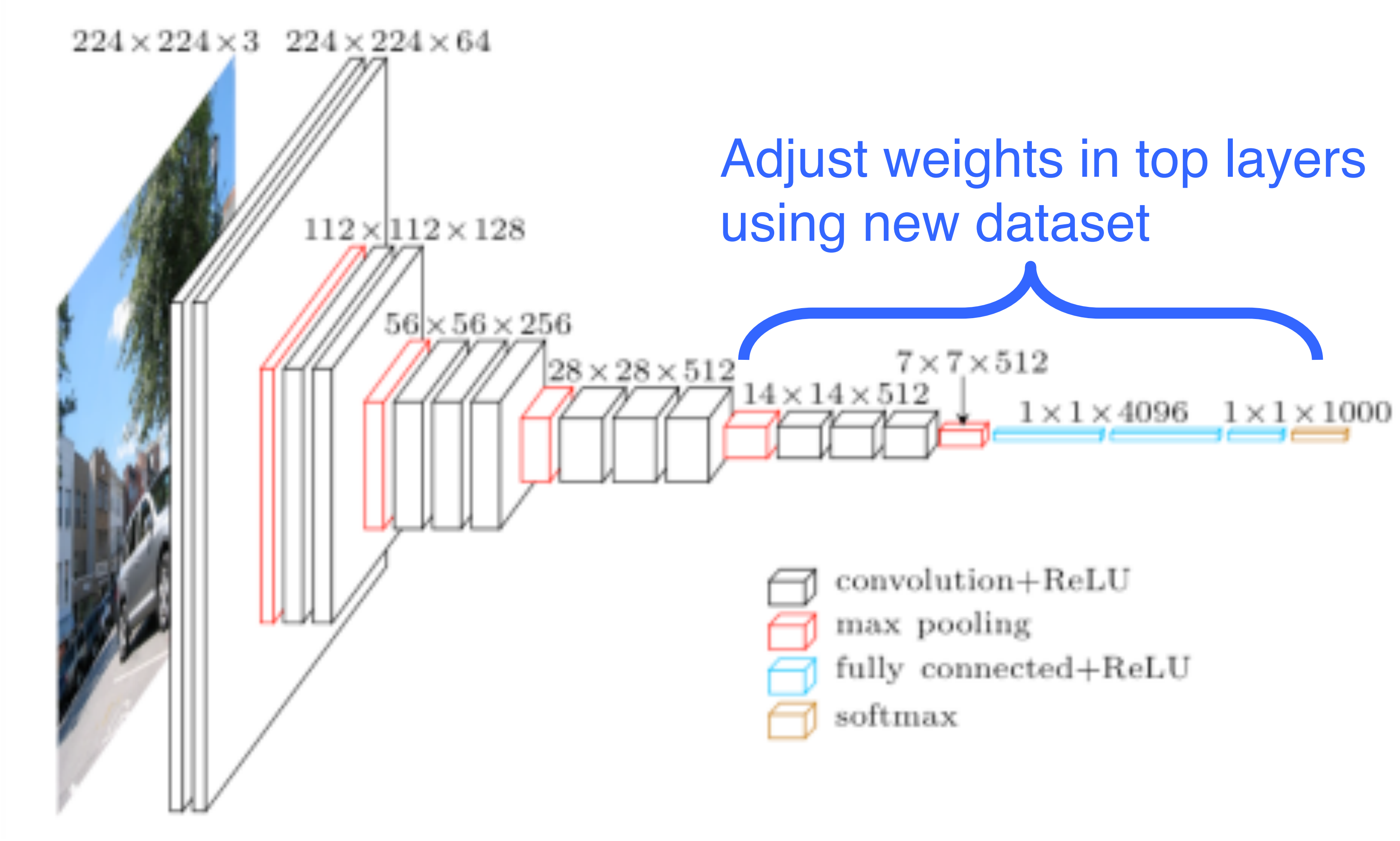
Source: https://www.cs.toronto.edu/~frossard/post/vgg16/

Transfer Learning – Feature Extraction



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

Transfer Learning – Fine Tuning



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

When & How to Fine Tune

- **New dataset is small & similar to original dataset**
  - Extract features from higher layer and feed to separate classifier
- **New dataset is large & similar to original dataset**
  - Fine tune top or all layers
- **New dataset is small & different from original dataset**
  - Extract features from lower layer and feed to separate classifier
- **New dataset is large & different from original dataset**
  - Fine tune top or all layers

Other Practical Tips

- **Learning rate**
  - Use very small learning rate for fine tuning. Don’t want to destroy what was already learned.
- **Start with properly trained weights**
  - Train top-level classifier first, then fine tune lower layers.
  - Top model with random weights may have negative effects on when fine tuning weights in pre-trained model
- **Data augmentation**
  - Simple ways to slightly alter images
    - Horizontal/vertical flips, random crops, translations, rotations, etc.
  - Use to artificially expand your dataset

Transfer Learning Hands-On

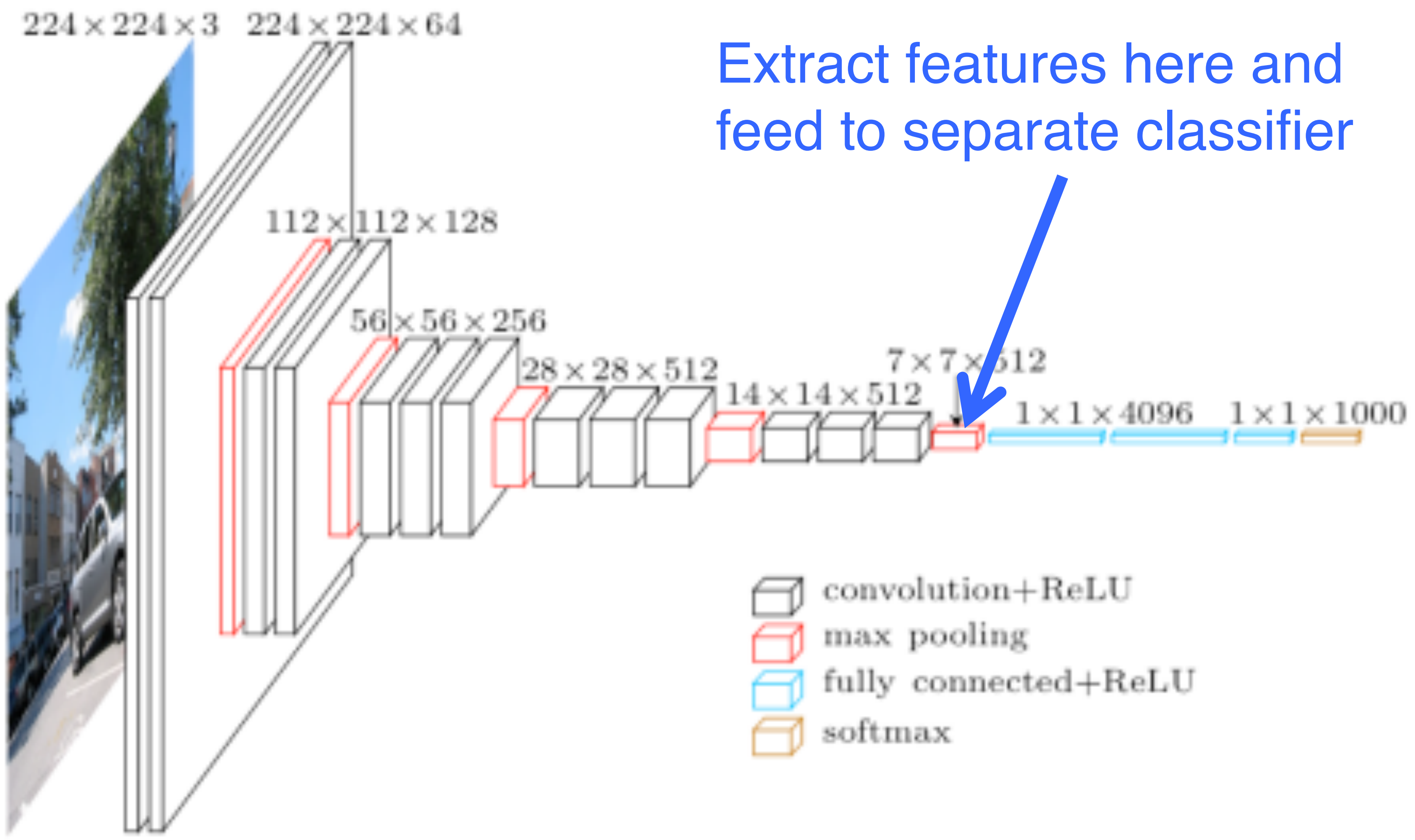
- **Data**
  - Cats and dogs images from Kaggle
- **Exercises**
  - Feature extraction
    - Use pre-trained CNN to extract features from images
    - Train neural network to classify cats/dogs using extract features
  - Fine tune
    - Adjust weights of last few layers of pre-trained CNN through training

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-2020

Server Setup

- **Set up server**
  - In terminal window: start_python_gpu
  - Should get something like this:

```
Your notebook is here:
        https://unkind-illicitly-mutt.comet-user-content.sdsc.edu?token=6615bbdb1a8e0fbe3ad948fb52678133
Submitted batch job 35032027
```

- **Connect to jupyter notebook**
  - In browser, paste URL of notebook from above step
- **Check queue**
  - *squeue –u $USER*

Data Setup

- **In terminal window, do the following:**

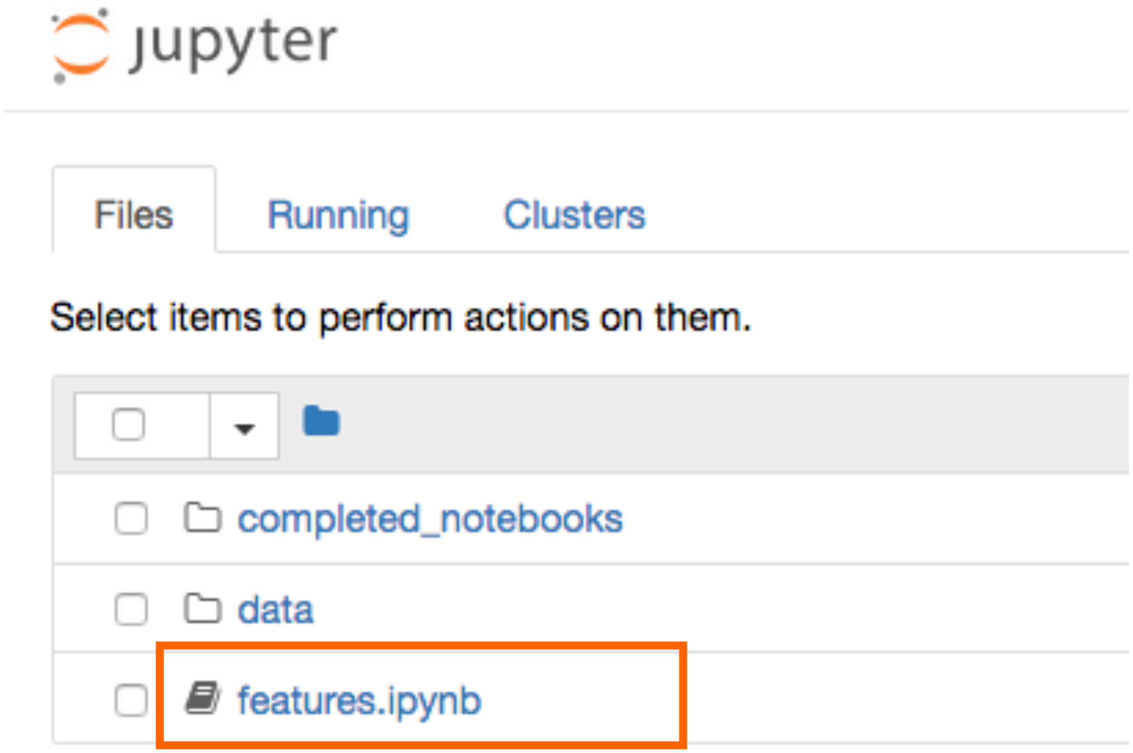
- **Create soft link to data**
  - ln –s ~/ML-data/data data

- **Get counts of images**
  - ls –l data/train/cats/* | wc -l
  - ls –l data/train/dogs/* | wc -l
  - ls –l data/validation/cats/* | wc -l
  - ls –l data/validation/dogs/* | wc -l

Data Description

- **Subset of Kaggle cats and dogs dataset**
- **Train**
  - 1000 cats + 1000 dogs
- **Validation**
  - 400 cats + 400 dogs

Open features.ipynb Notebook



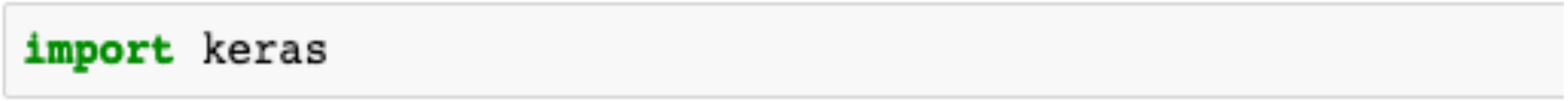
Select items to perform actions on them.

completed_notebooks

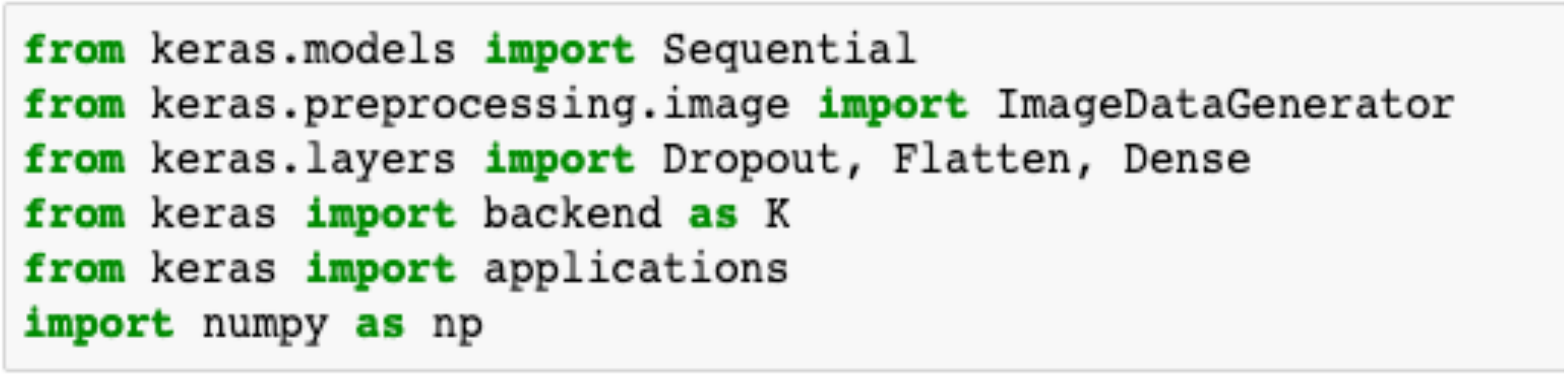
data

features.ipynb

Import Modules

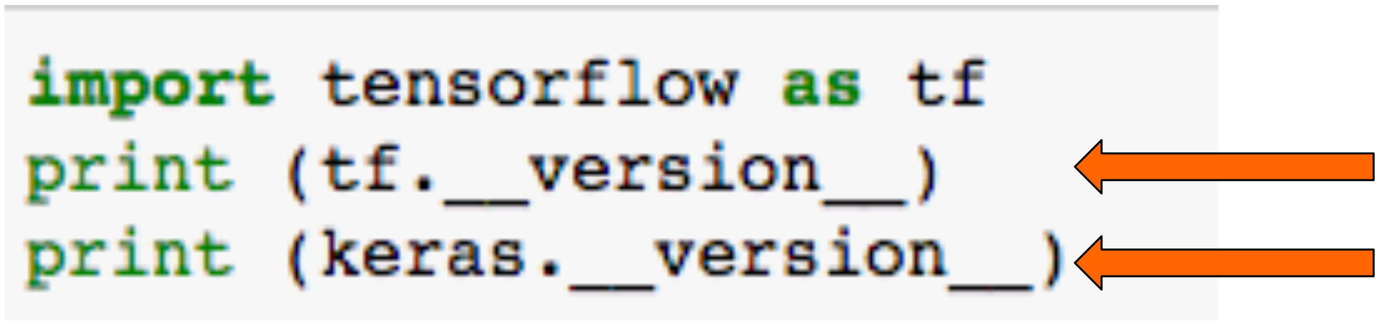


```python
import keras
```



```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)

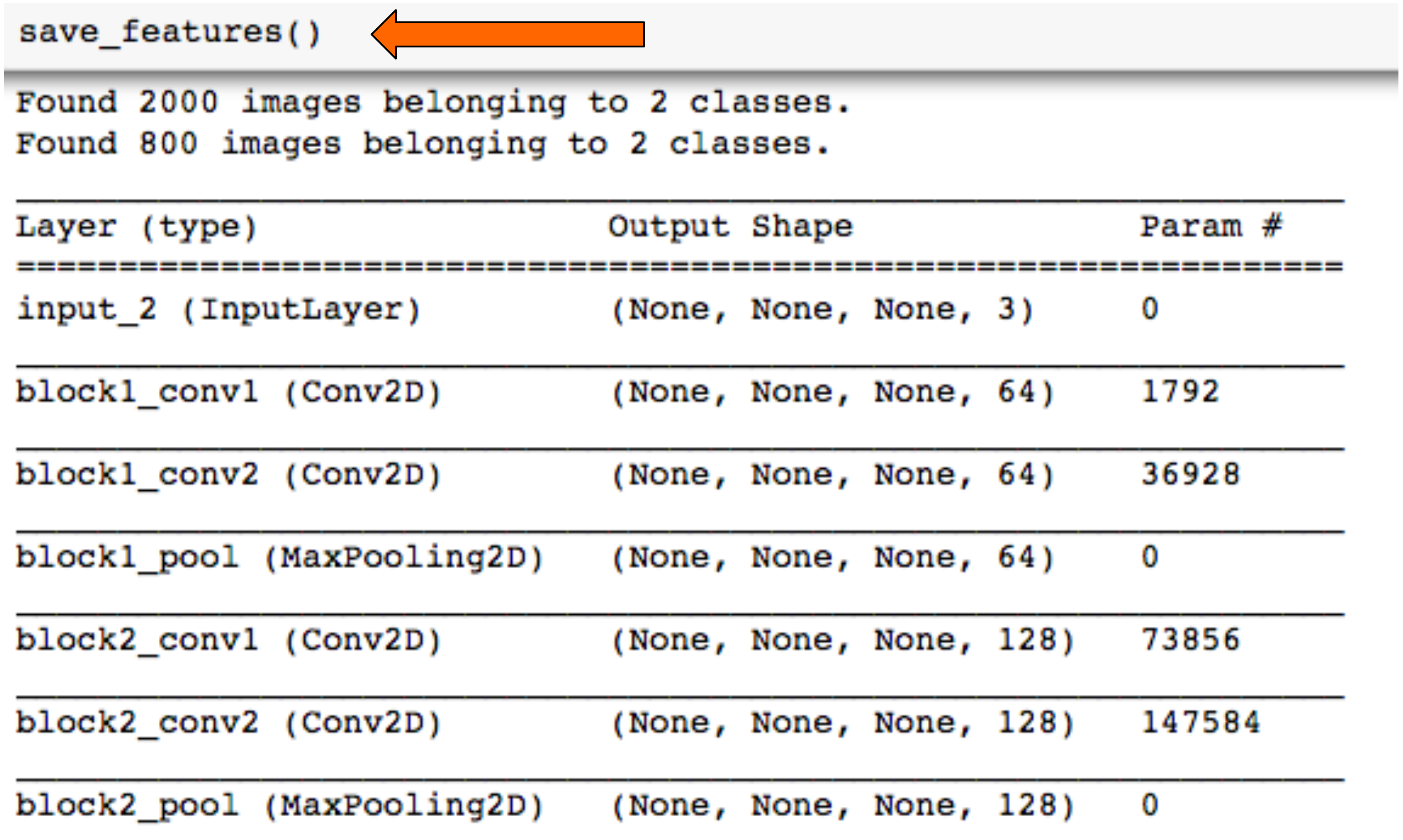
Method to Extract Features from Pre-Trained Network

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')

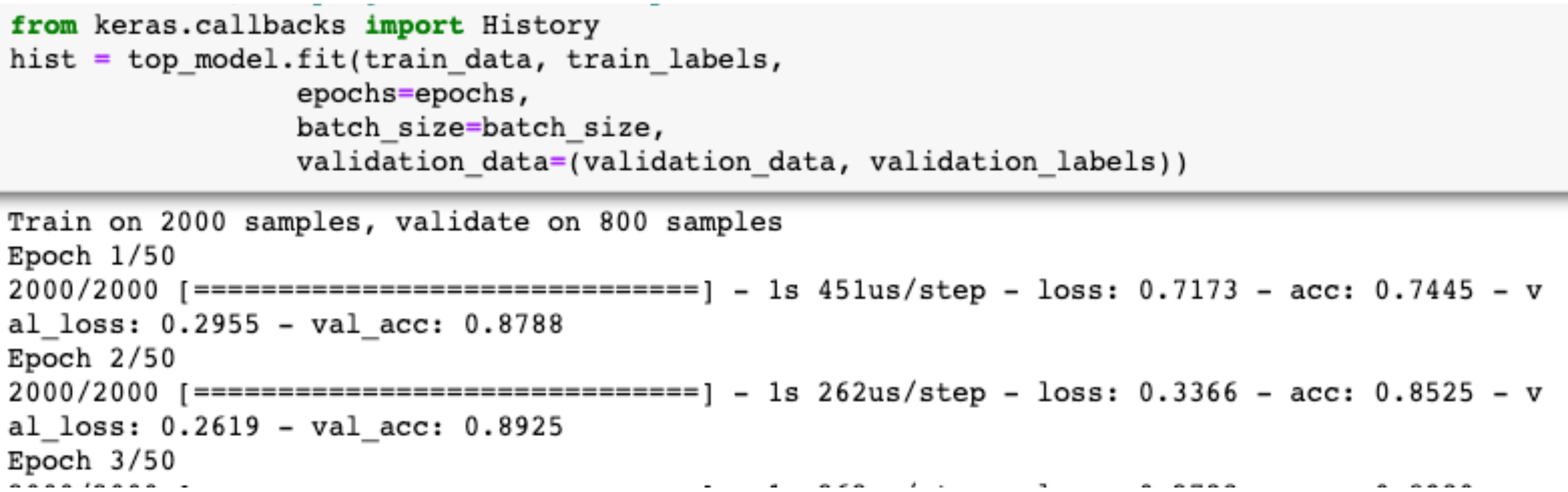
(2000,) (800,)

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**

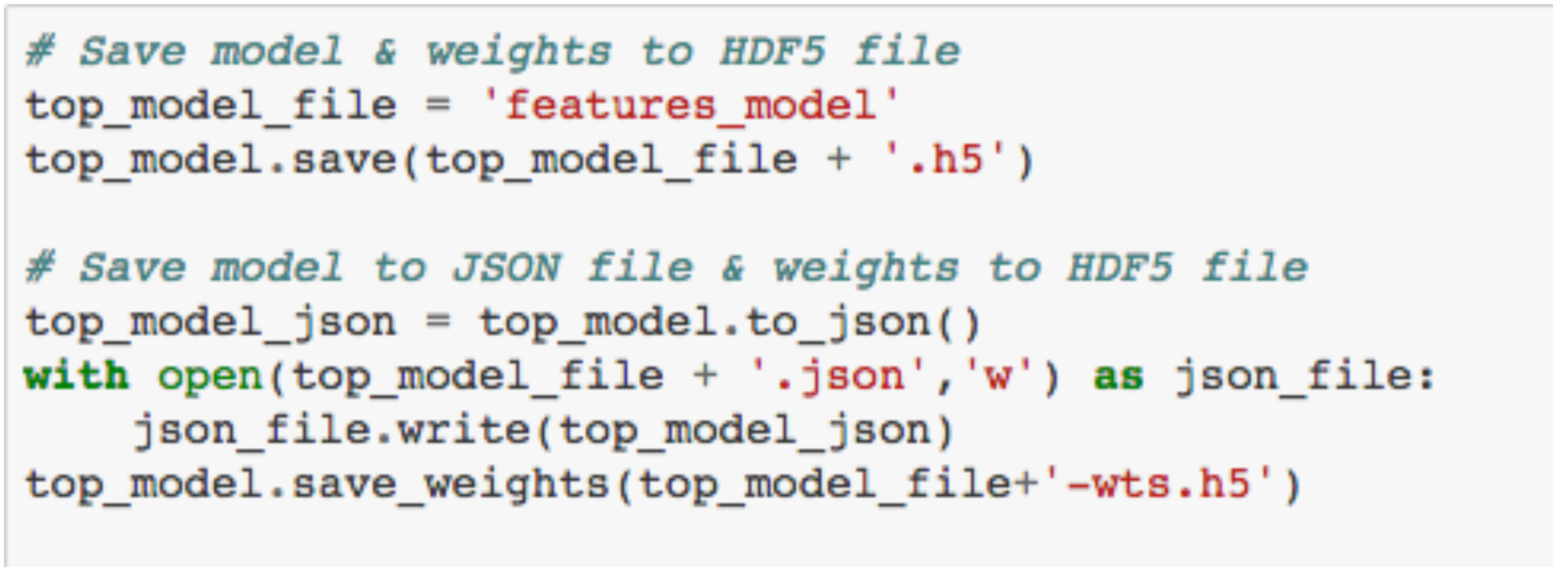


```
from keras.callbacks import History
hist = top_model.fit(train_data, train_labels,
                     epochs=epochs,
                     batch_size=batch_size,
                     validation_data=(validation_data, validation_labels))
```

```
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**

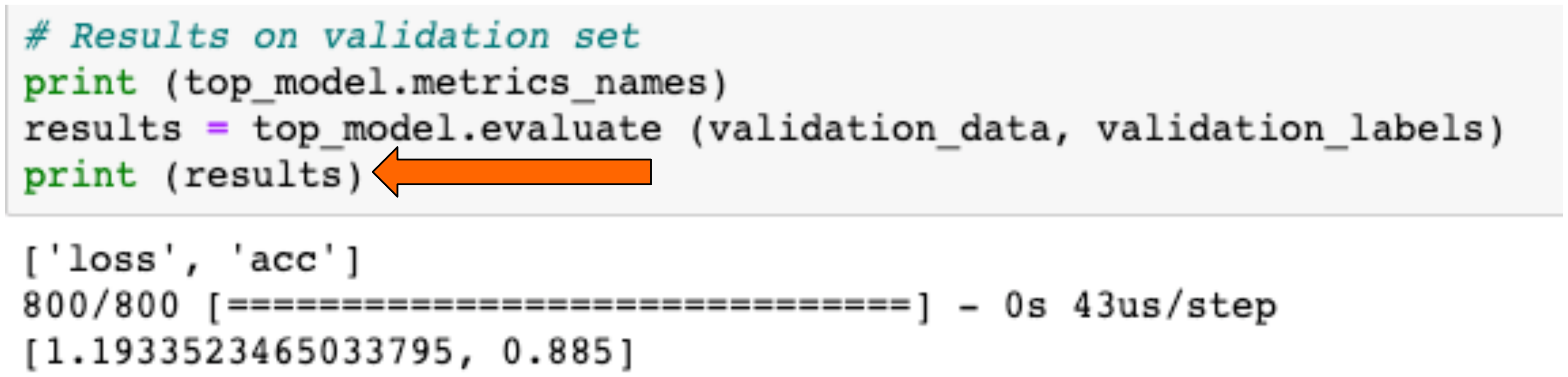


```
# Save model & weights to HDF5 file
top_model_file = 'features_model'
top_model.save(top_model_file + '.h5')

# 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**



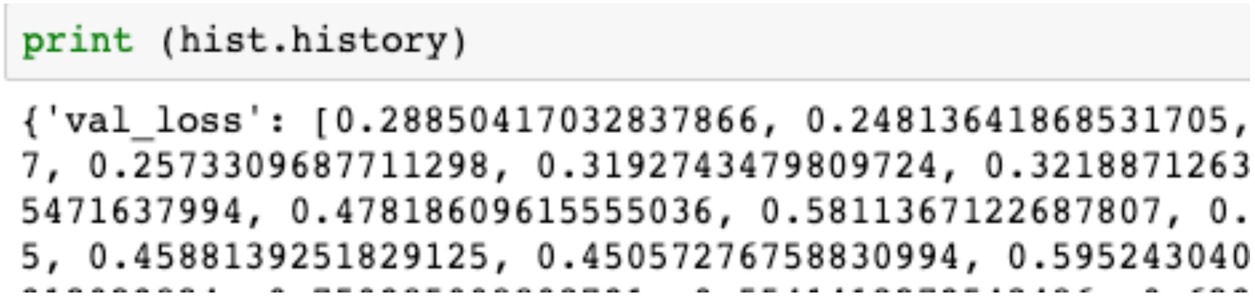
```
# Results on validation set
print (top_model.metrics_names)
results = top_model.evaluate (validation_data, validation_labels)
print (results)
```

```
['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

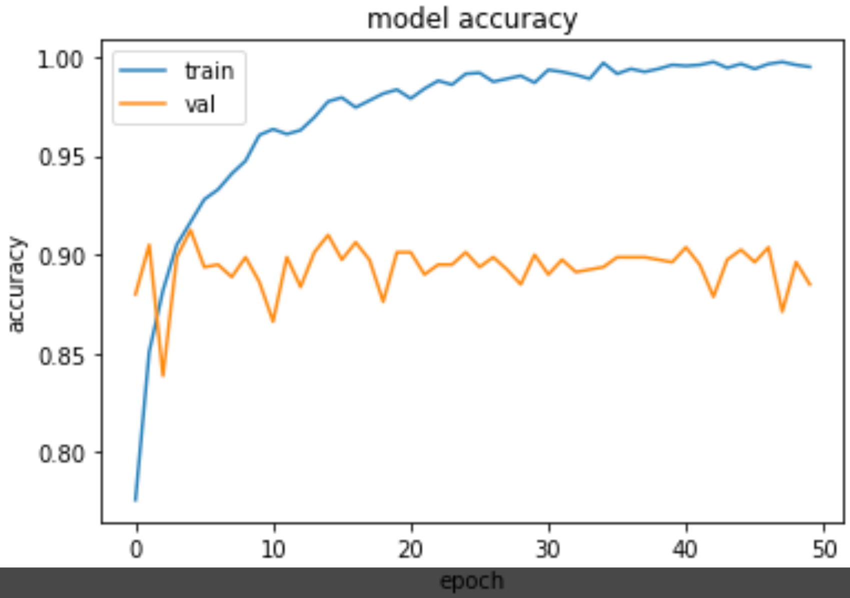
- Print training history



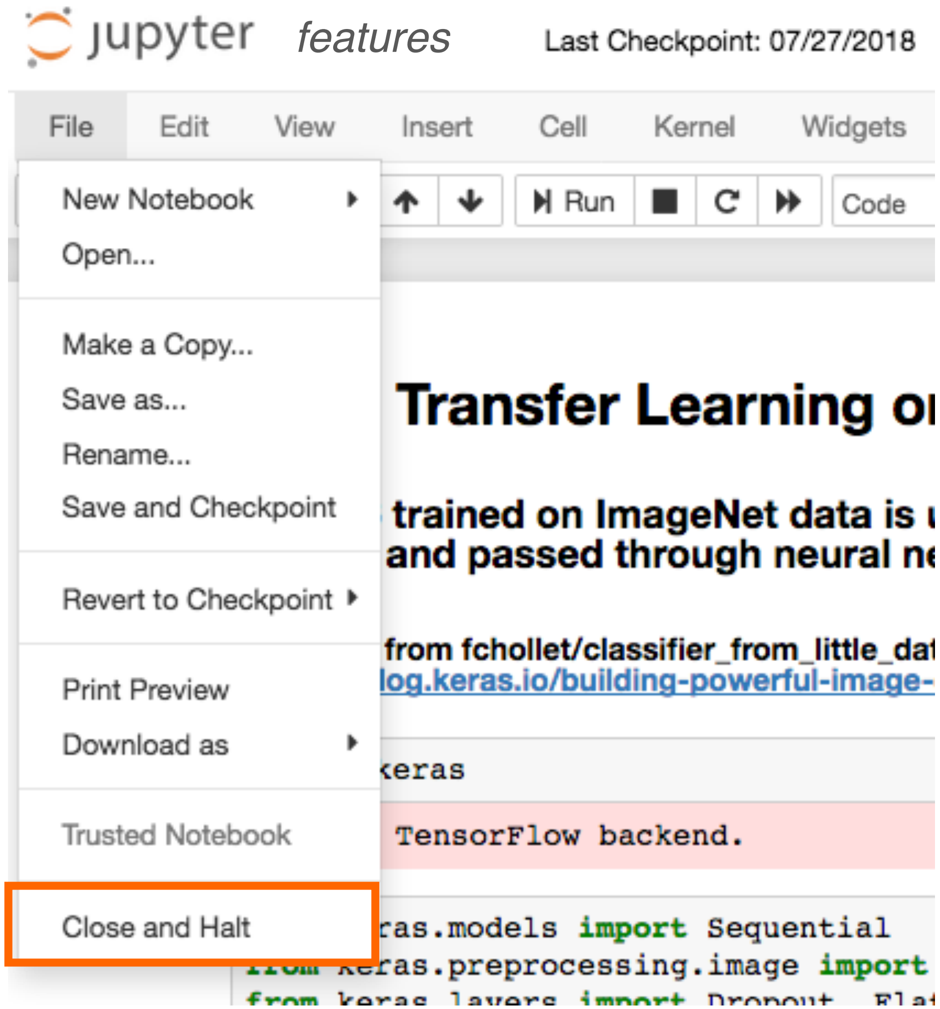
```
print (hist.history)
```

```
{'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 ne

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

```
keras
```

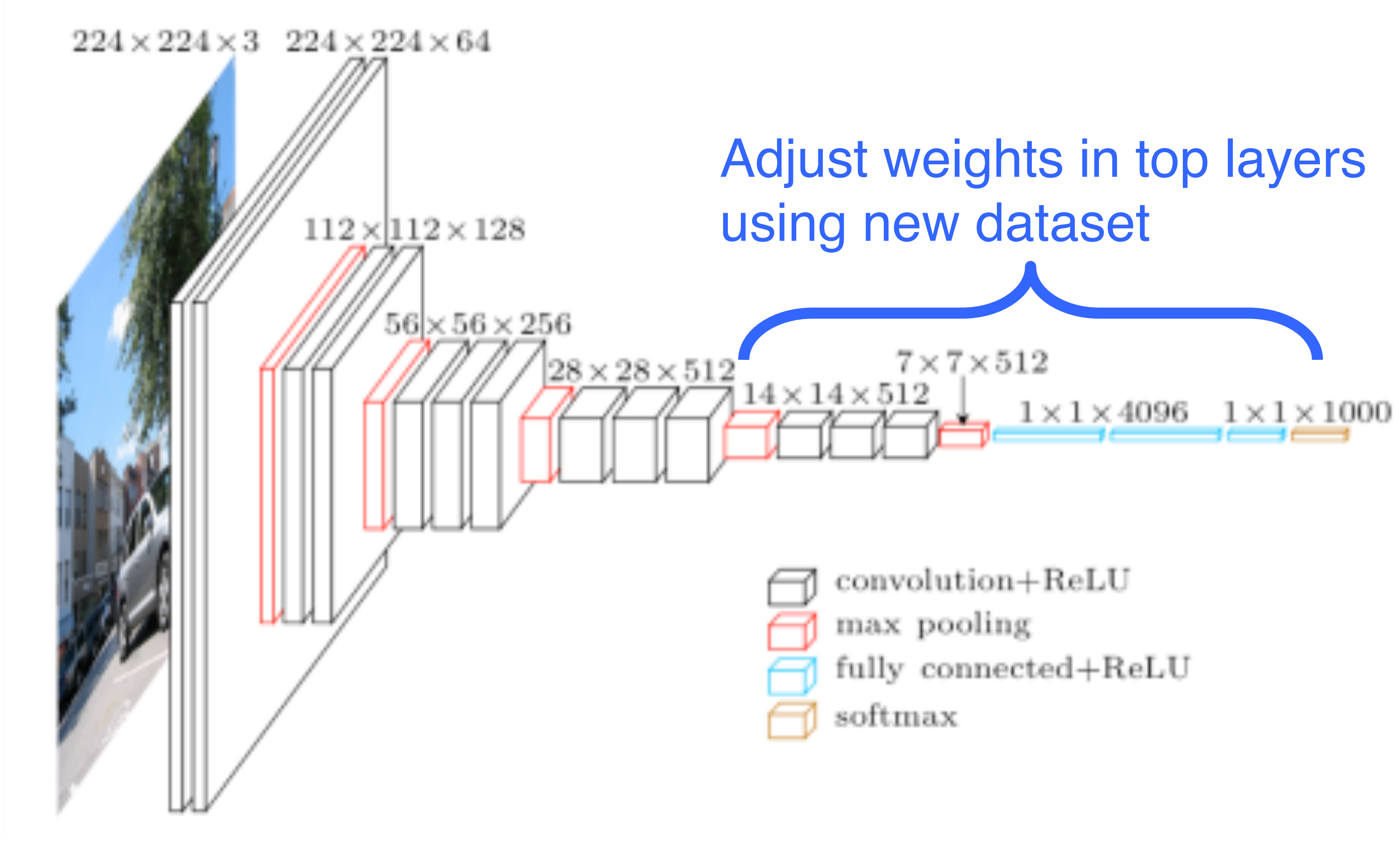
```
TensorFlow backend.
```

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

Fine Tuning Hands-On

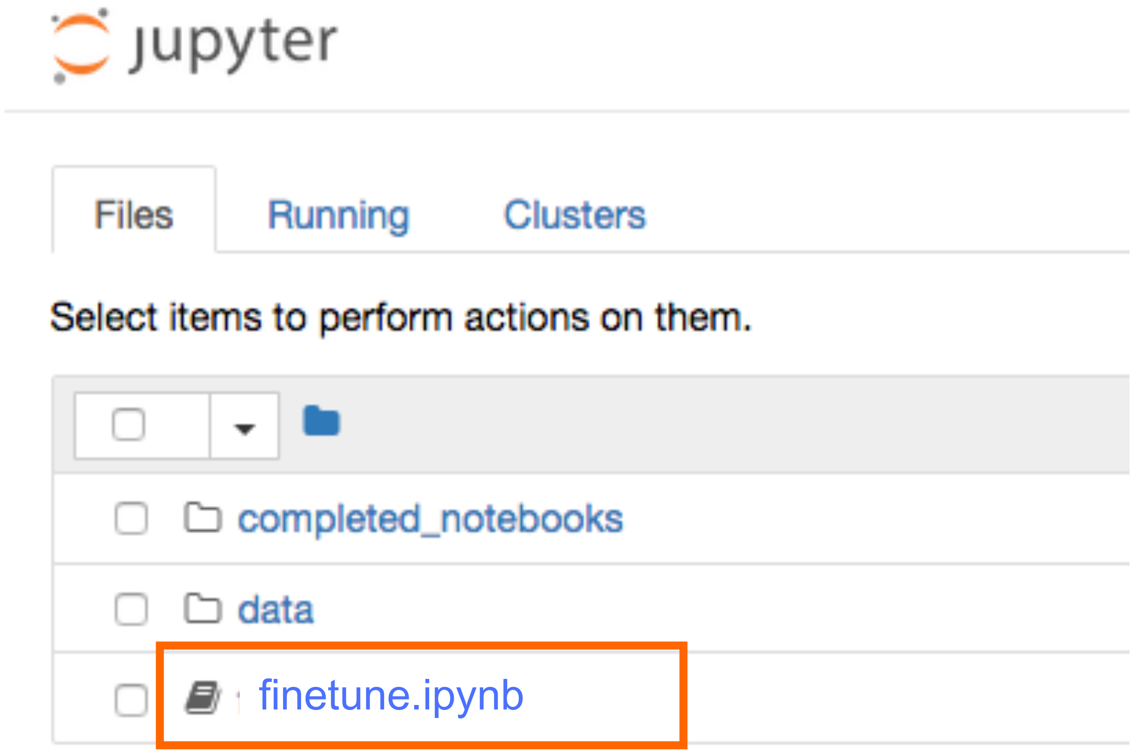
- **Data**
  - Cats and dogs images from Kaggle
- **Method**
  - Use VGG16 trained on ImageNet data as pre-trained model.
  - Replace last fully connected layer with neural network trained from Feature Extraction hands-on.
  - Fine tune last convolution block and fully connected layer.

Transfer Learning – Fine Tuning



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

Open fine-tune.ipynb Notebook



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)

Load Pre-Trained CNN

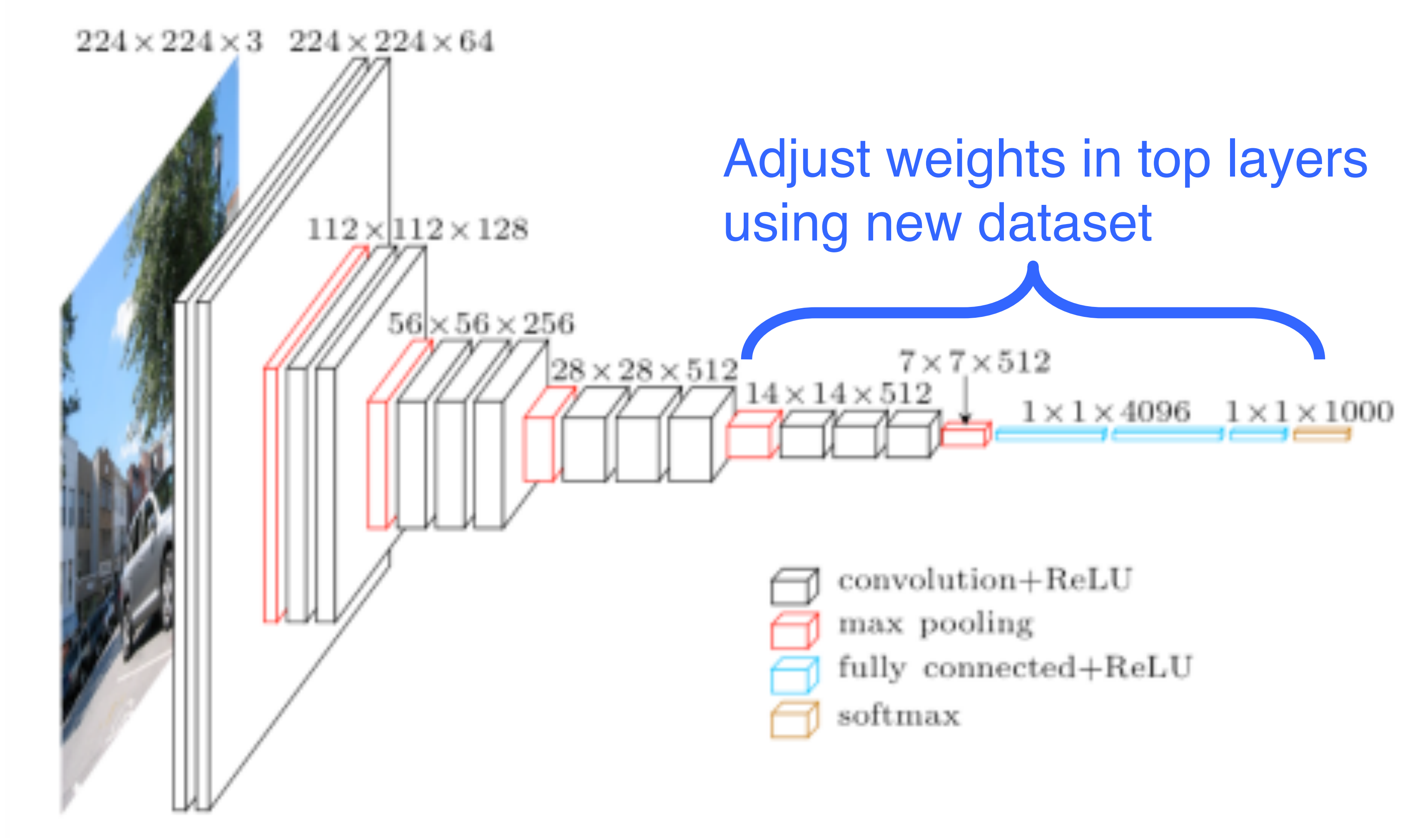
- Load pre-trained model without last fully connected layer

```
base_model = applications.VGG16
                        (weights='imagenet’,
                         include_top=False,
                         input_shape=(img_width,img_height,3))
print (‘Model loaded’)
```

- Print out base model summary

```
base_model.summary()
```

Transfer Learning – Fine Tuning



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

Create Top Model

- Create top model
  - Create fully connected layer as top model and connect to pre-trained base model
- Load top model's weights
  - Weights are in 'features_model_wts.h5'
- Add top model to base CNN to create model
- Freeze weights

```
for layer in model.layers[:15]
    layer.trainable = False
```

- Compile model
- Print out model summary

```
model.summary()
```

Model

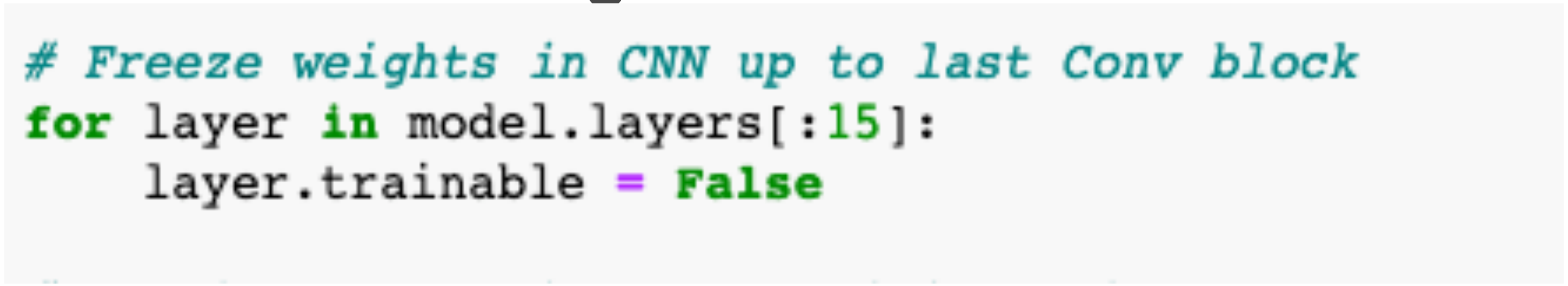
- **Original Model**

  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

Prepare Data

- Set batch size

```
batch_size = 16
```

- Set batch size for train_generator

```
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary',
    seed=seed)
```

Fine Tune Model

- Set number of training epochs

```
    epochs = 5
```

- Set batch size for train_generator

```
from keras.callbacks import History
hist = model.fit_generator(
    train_generator,
    steps_per_epoch = nb_train_samples // batch_size,
    epochs = epochs,
    validation_data = validation_generator,
    validation_steps = nb_validation_samples // batch_size,
    initial_epoch=0,
    verbose = 2)
```

Get Classification Results

- Get classification results after fine tuning

```
results = model.evaluate_generator(
            train_generator,
            steps=nb_train_samples // batch_size)
print (results)

results = model.evaluate_generator(
            validation_generator,
            steps=nb_validation_samples // batch_size)
print (results)
```

Save Model and Weights

- Save model & weights

```
    model_file = 'finetune'
```

- Get results on validation set

```
print (model.metrics_names)
results = model.evaluate_generator(
        validation_generator,
         steps = nb_validationsamaples // batch_size)
print (results)
```

Print Training History

- Print history

  print (hist.history)

Predict Class of Image

- Use model to predict class of image

  result = model.predict(x)

  print ("Prediction probability: ", result)

Clean Up

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

References

- **F. Chollet. The Keras Blog.**
  - https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
- **ImageNet**
  - http://www.image-net.org/
- **Transfer Learning**
  - http://cs231n.github.io/transfer-learning/
- **Satellite Image Analysis Use Case**
  - https://ieeexplore.ieee.org/abstract/document/8109118?casa_token=TCdQ0aSgBjgAAAAA:fQUwcByPhSuByj_8u2iTIl_kLh9BPKlSq6akqSK04SwBKKV1YprcVoezhcjpWcpIDxlXdtlF