CNN TRANSFER LEARNING EXERCISE

- Data
  - Cats and dogs images from Kaggle

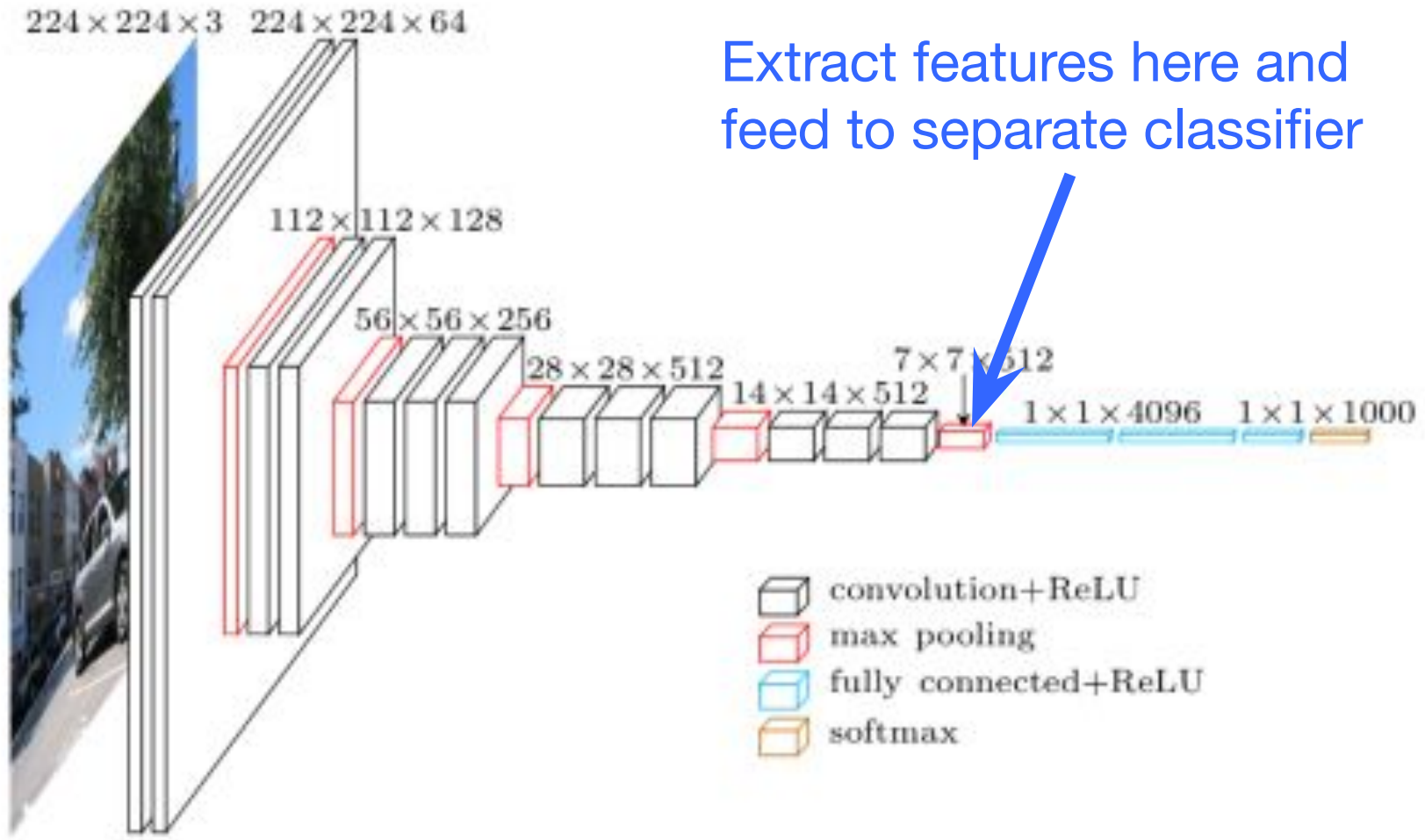
- Exercises
  - Feature extraction
    - Use pre-trained CNN to extract features from images
    - Train neural network to classify cats/dogs using extract features
  - Fine tune
    - Adjust weights of last few layers of pre-trained CNN through training

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



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

AWS SETUP

- Create a SageMaker notebook instance
- Notebook instance name
- Notebook instance type(with GPU acceleration)
  - **ml.p2.xlarge**
- IAM Role - Default
- Create notebook instance
  - Click on ‘Create notebook instance’
  - Wait until notebook status changes to InService.
  - Click ‘Open JupyterLab’
- Upload existing notebook
  - Click on up arrow to upload notebook (features.ipynb and finetune.ipynb)
  - Select **conda_tensorflow2_p36** kernel

LOOK AT DATA

- In terminal window, do the following

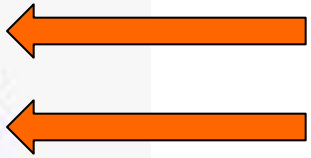
- Get counts of images
  - ls –l ~/train/cats/* | wc -l
  - ls –l ~/train/dogs/* | wc -l
  - ls –l ~/validation/cats/* | wc -l
  - ls –l ~/validation/dogs/* | wc -l

DATA DESCRIPTION

- Subset of Kaggle cats and dogs dataset
- Train
  - 1000 cats + 1000 dogs
- Validation
  - 200 cats + 200 dogs
- Test
  - 200 cats + 200 dogs

PRINT SOFTWARE VERSIONS

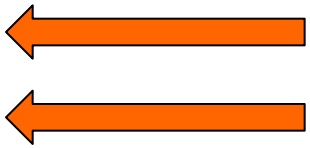
```
import tensorflow as tf
print (tf.__version__)
print (keras.__version__)
```



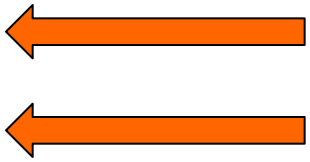
SET DATA PARAMETERS

- Set image dimensions
  - *img_width, img_height = 150, 150*

- Set data location
  - *train_data_dir = 'train'*
  - *validation_data_dir = 'validation'*



- Set number of images
  - *nb_train_samples = 2000*
  - *nb_validation_samples = 800*



(150, 150, 3)

METHOD TO EXTRACT FEATURES FROM PRE-TRAINED MODEL

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

- 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 & 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)
```

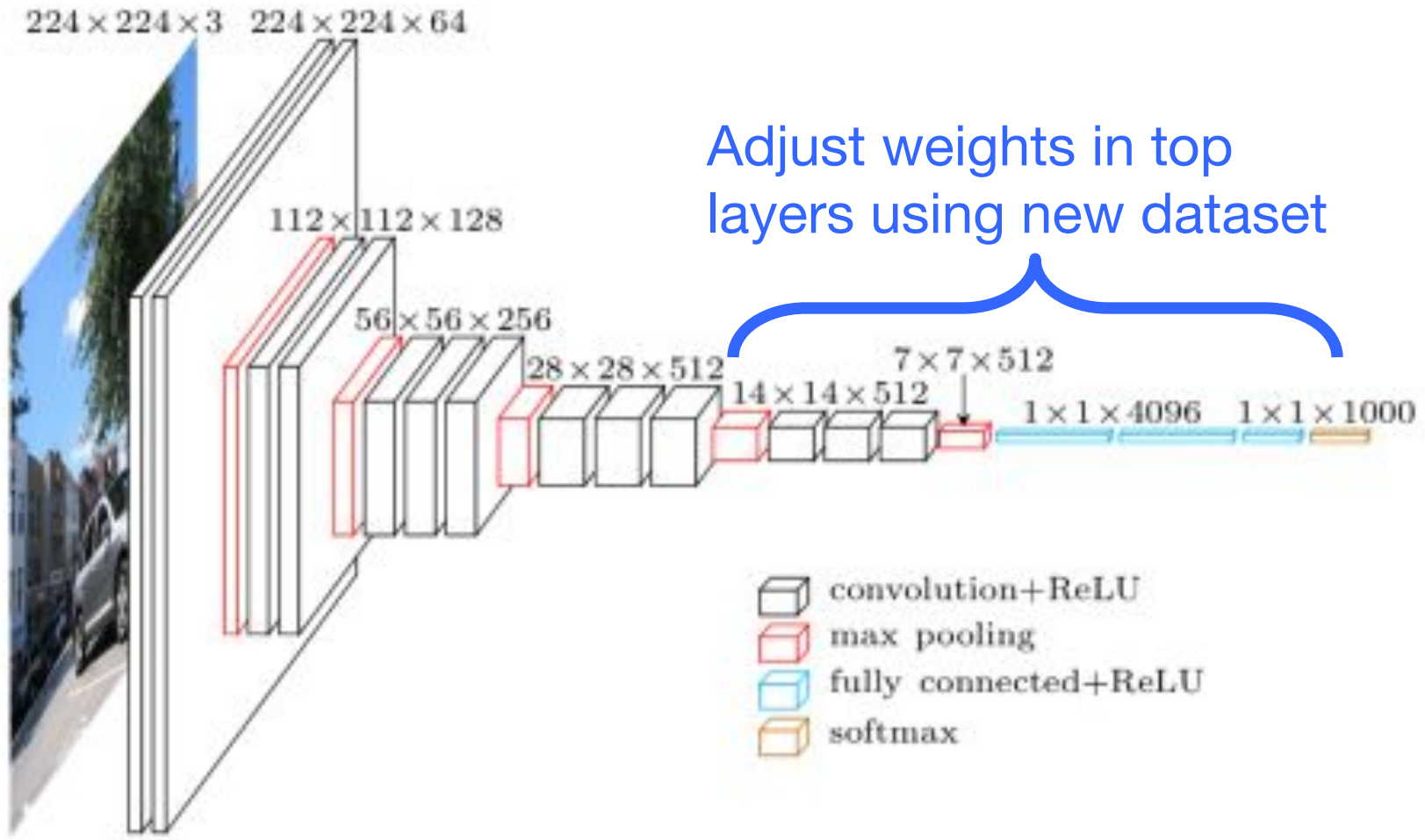
```
['loss', 'acc']
800/800 [==============================] - 0s 43us/step
[1.1933523465033795, 0.885]
```

  - Results should be the same
- Validation accuracy on CNN trained from scratch
  - ~87.5%

TRANSFER LEARNING - FINE TUNING

- 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

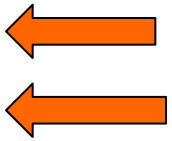


Source: https://www.cs.toronto.edu/~frossard/post/vgg16/

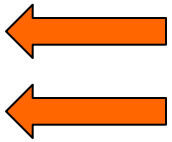
SET DATA PARAMETERS

- Set image dimensions
  - *img_width, img_height = 150, 150*

- Set data location
  - *train_data_dir = 'train'*
  - *validation_data_dir = 'validation'*



- Set number of images
  - *nb_train_samples = 2000*
  - *nb_validation_samples = 800*



(150, 150, 3)

LOAD PRE-TRAINED MODEL

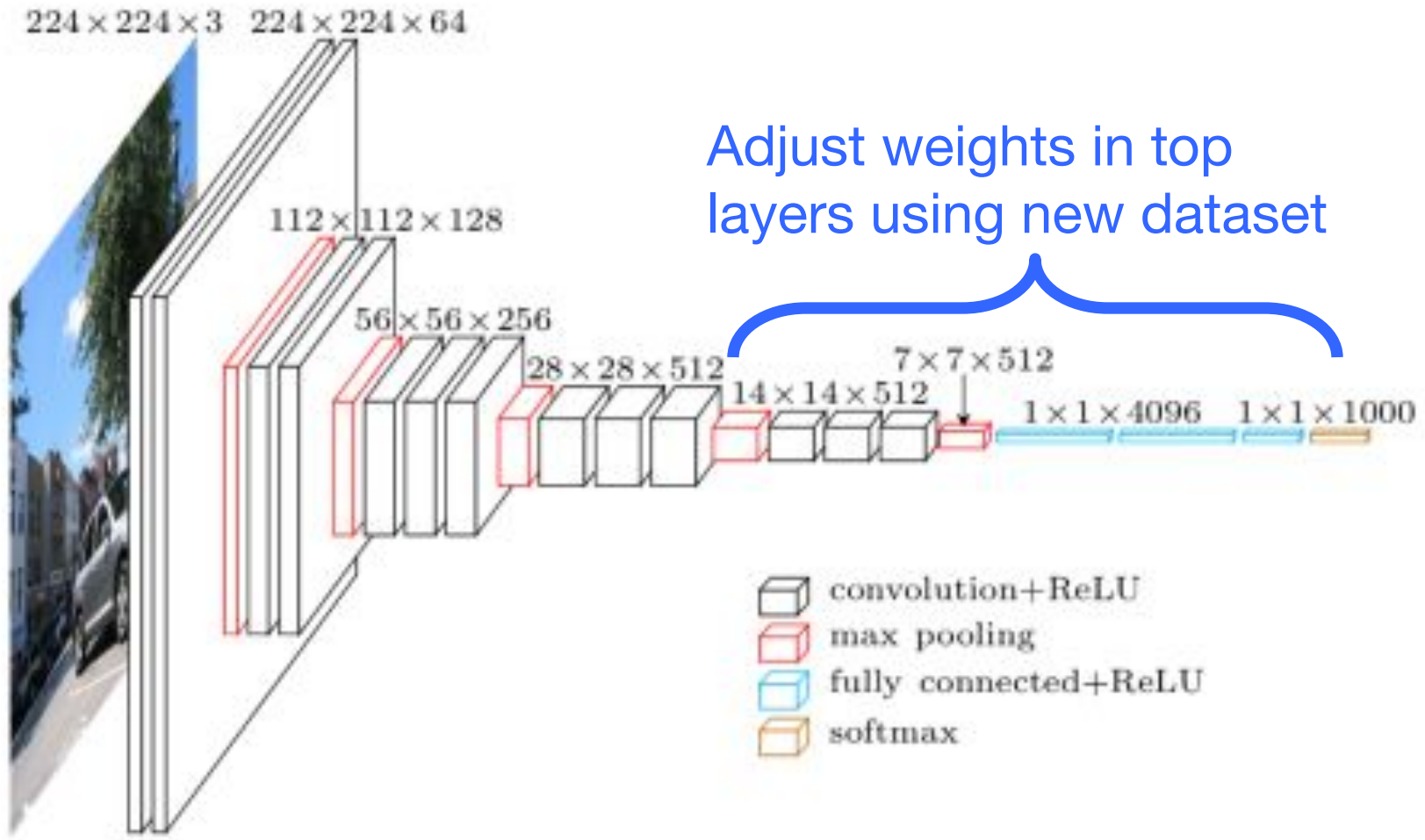
- Load pre-trained model without last fully connected layer

```
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
```

  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 TUNING

- 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 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 & 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)
```

OUTPUT TRAINING HISTORY

- Print history

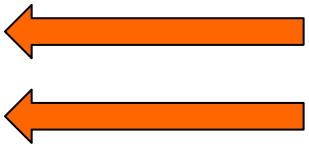
  print (hist.history)

INFERENCE

- Use model to predict class of image

  result = model.predict(x)

  print (“Prediction probability: “, result)



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]