Transfer Learning and Fine Tuning

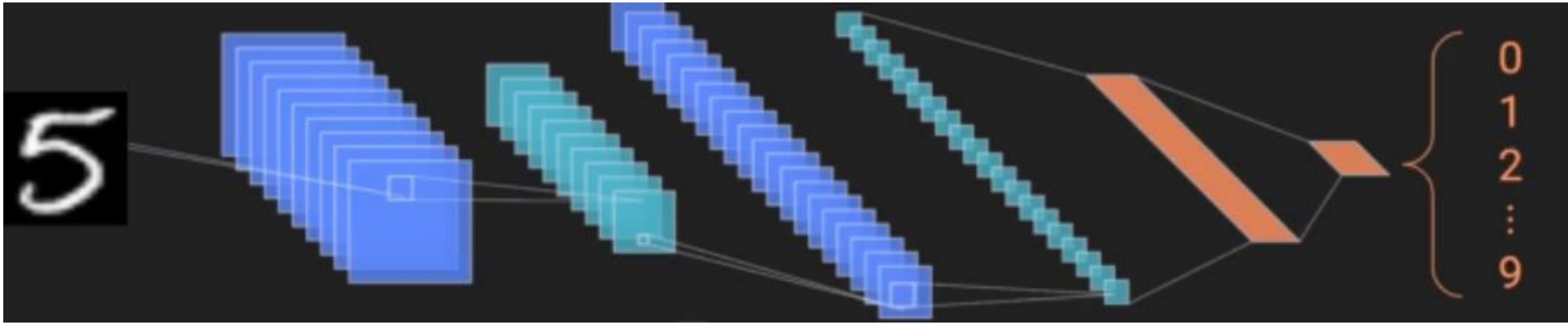
Dr. Mongkol Ekpanyapong

What is Transfer Learning?

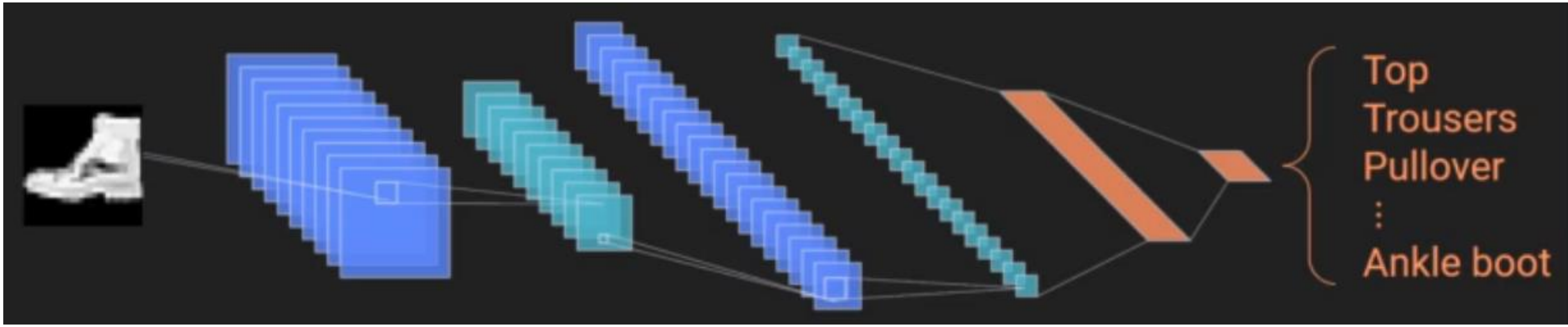
Transfer Learning

- In practice, very few people train an entire CNN from scratch
- The common practice is to use pretrain model and apply transfer learning
- Important low-level features are the same e.g., 4-legged animals such as dogs, cats, horses have some common features

Transfer Learning



Transfer Learning

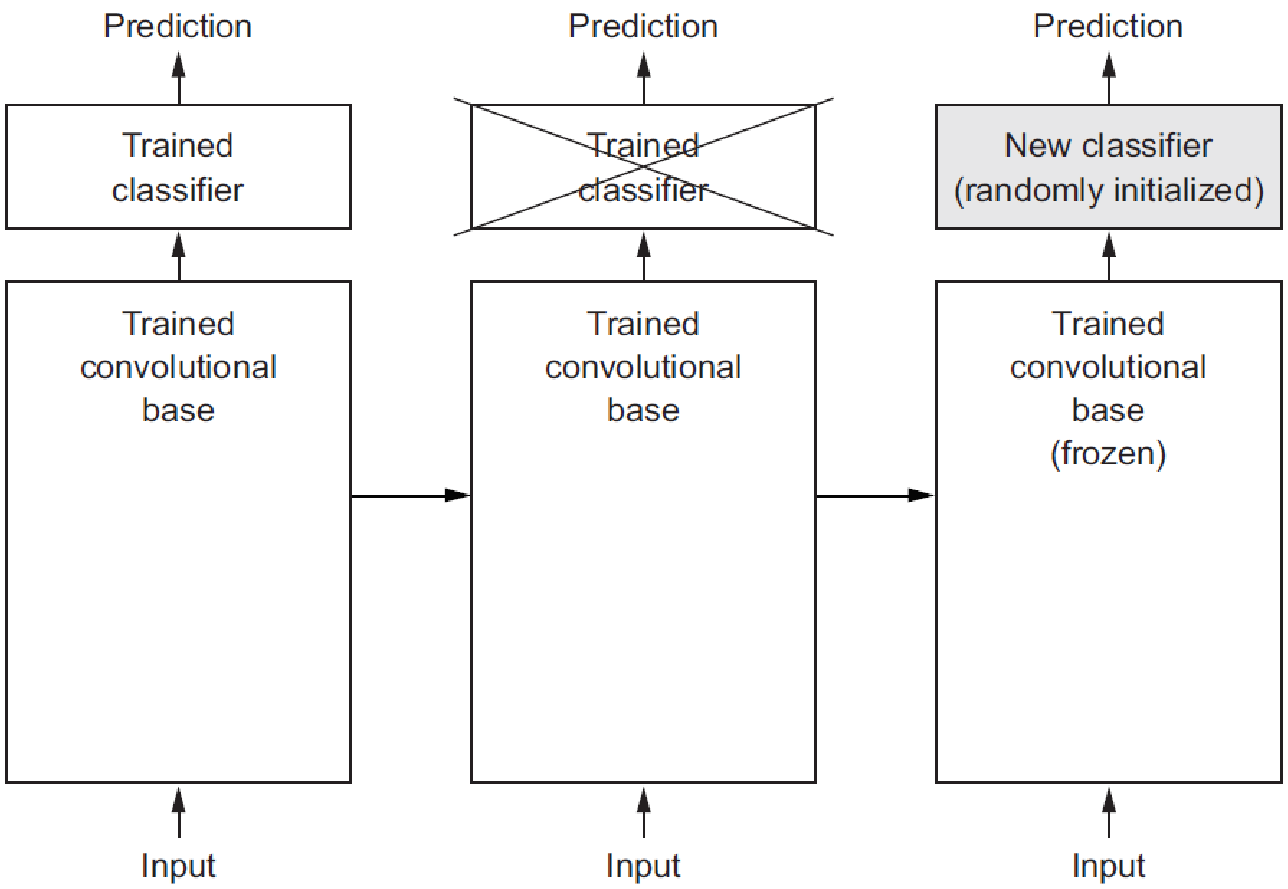


Transfer Learning

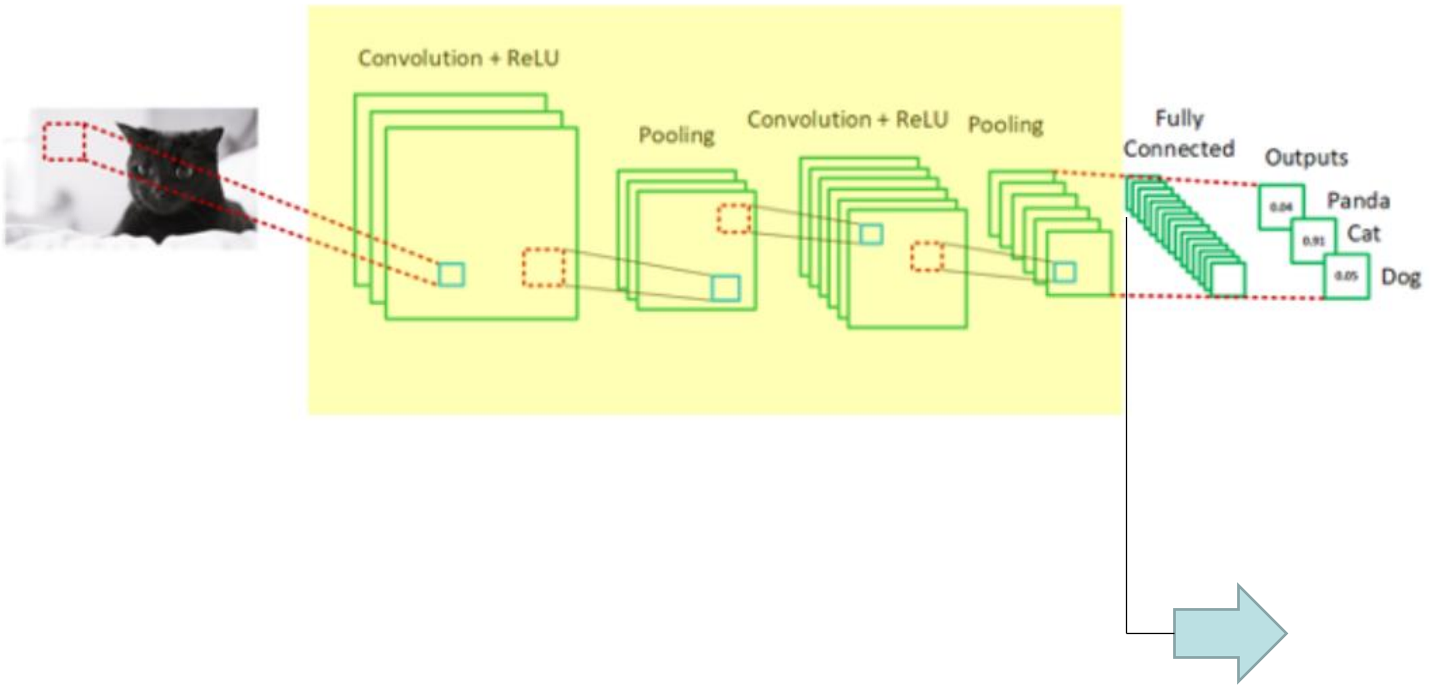
Effective approach on small image dataset

- Feature Extraction
- Fine Tuning

Feature Extraction



Feature Extractor

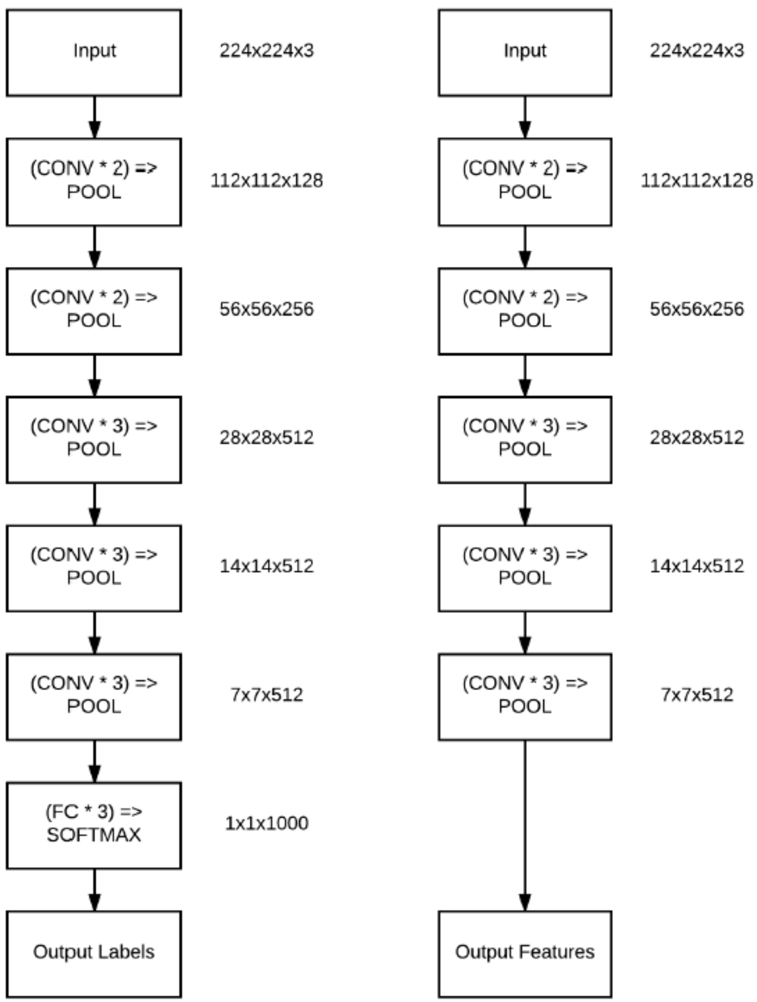


Networks as Feature Extractors

Up until this point, we treated CNN as end-to-end image classifiers:

- We input an image to the network
- The image forward propagates through the network
- We obtain the final classification probabilities from the end of the network

Can we use them to generate feature vectors

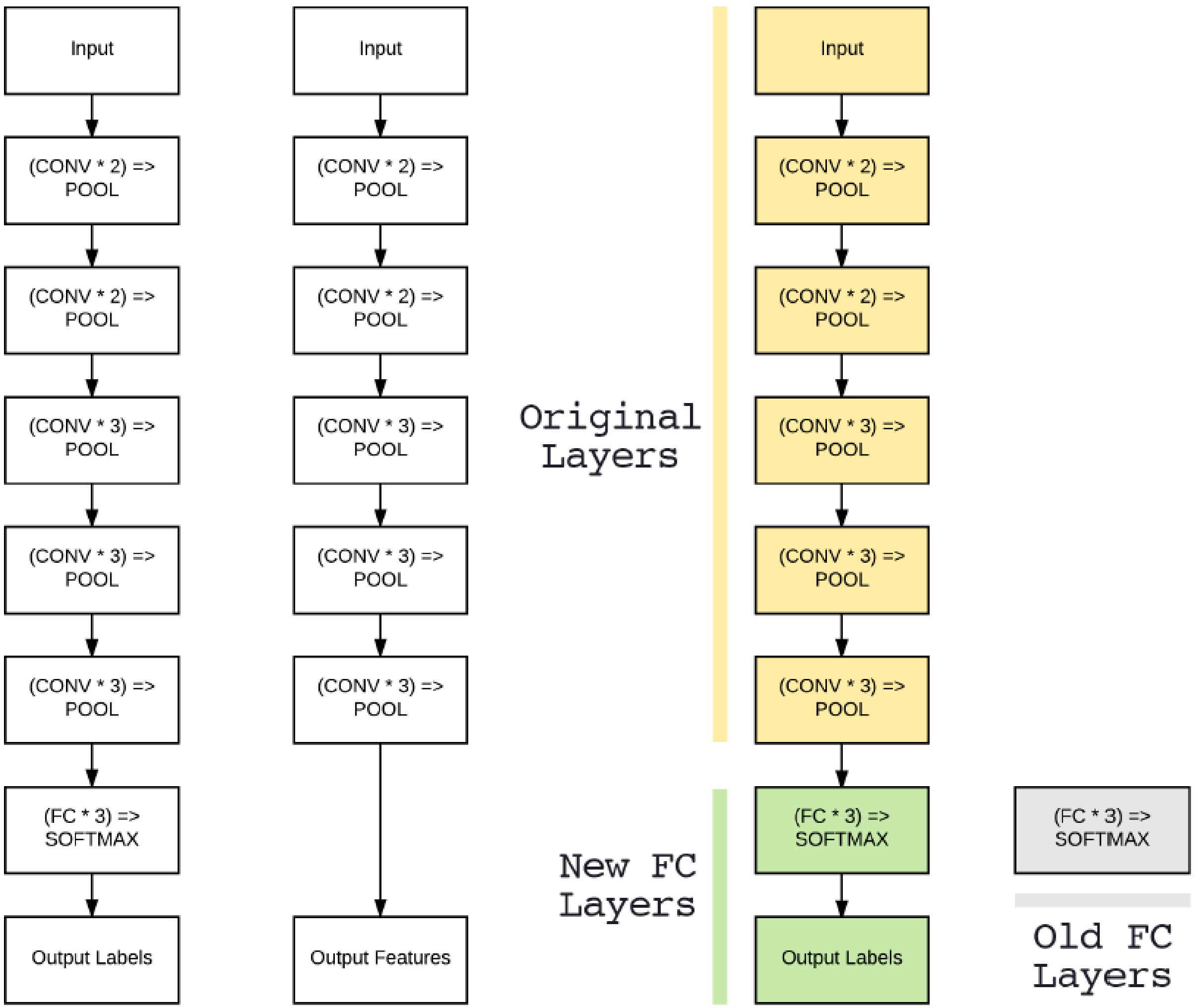


Instantiating VGG16

```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))
```

- Weights specifies the weight checkpoint
- Include_top refers to including (or not) the densely connected classifier
- input_shape is the shape of the input image that you'll feed to the network

VGG16 with feature extraction



```
>>> conv_base.summary()
Layer (type)                     Output Shape          Param #
================================================================
input_1 (InputLayer)             (None, 150, 150, 3)   0
________________________________________________________________
block1_conv1 (Convolution2D)     (None, 150, 150, 64)  1792
________________________________________________________________
block1_conv2 (Convolution2D)     (None, 150, 150, 64)  36928
________________________________________________________________
block1_pool (MaxPooling2D)       (None, 75, 75, 64)    0
________________________________________________________________
block2_conv1 (Convolution2D)     (None, 75, 75, 128)   73856
________________________________________________________________
block2_conv2 (Convolution2D)     (None, 75, 75, 128)   147584
________________________________________________________________
block2_pool (MaxPooling2D)       (None, 37, 37, 128)   0
________________________________________________________________
block3_conv1 (Convolution2D)     (None, 37, 37, 256)   295168
________________________________________________________________
block3_conv2 (Convolution2D)     (None, 37, 37, 256)   590080
________________________________________________________________
block3_conv3 (Convolution2D)     (None, 37, 37, 256)   590080
________________________________________________________________
block3_pool (MaxPooling2D)       (None, 18, 18, 256)   0
________________________________________________________________
block4_conv1 (Convolution2D)     (None, 18, 18, 512)   1180160
________________________________________________________________
block4_conv2 (Convolution2D)     (None, 18, 18, 512)   2359808
________________________________________________________________
block4_conv3 (Convolution2D)     (None, 18, 18, 512)   2359808
________________________________________________________________
block4_pool (MaxPooling2D)       (None, 9, 9, 512)     0
________________________________________________________________
block5_conv1 (Convolution2D)     (None, 9, 9, 512)     2359808
________________________________________________________________
block5_conv2 (Convolution2D)     (None, 9, 9, 512)     2359808
________________________________________________________________
block5_conv3 (Convolution2D)     (None, 9, 9, 512)     2359808
________________________________________________________________
block5_pool (MaxPooling2D)       (None, 4, 4, 512)     0
================================================================
Total params: 14,714,688
Trainable params: 14,714,688
```

Final feature map has (4,4,512)

Extracting features using the pretrained convolutional base

```
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator

base_dir = '/Users/fchollet/Downloads/cats_and_dogs_small'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20

def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i += 1
        if i * batch_size >= sample_count:
            break
    return features, labels

train_features, train_labels = extract_features(train_dir, 2000)
validation_features, validation_labels = extract_features(validation_dir, 1000)
test_features, test_labels = extract_features(test_dir, 1000)
```

Note that because generators yield data indefinitely in a loop, you must break after every image has been seen once.

Flatten the Data

- This code is to flatten the data before feeding them to another classifier

```
train_features = np.reshape(train_features, (2000, 4 * 4 * 512))
validation_features = np.reshape(validation_features, (1000, 4 * 4 * 512))
test_features = np.reshape(test_features, (1000, 4 * 4 * 512))
```

Training the densely connected classifier

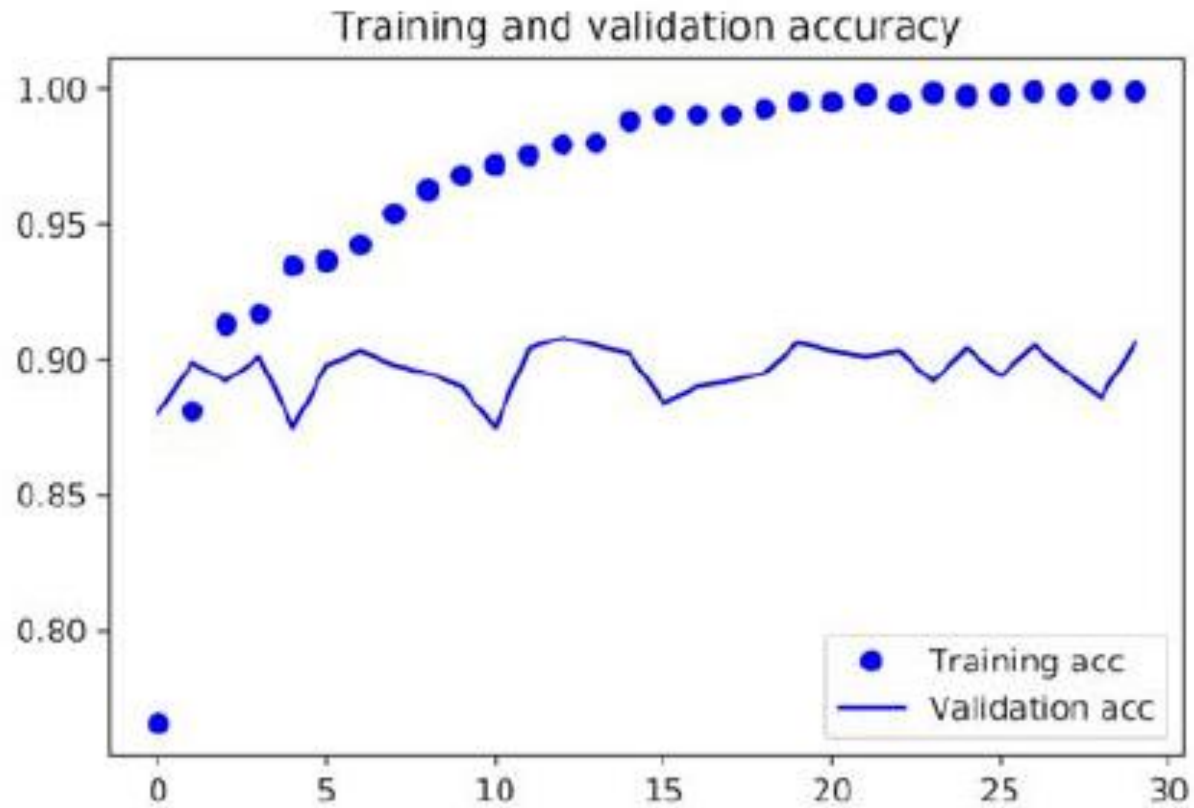
```
from keras import models
from keras import layers
from keras import optimizers

model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=20,
                    validation_data=(validation_features, validation_labels))
```

Training is very fast since we deal with only two Dense layers

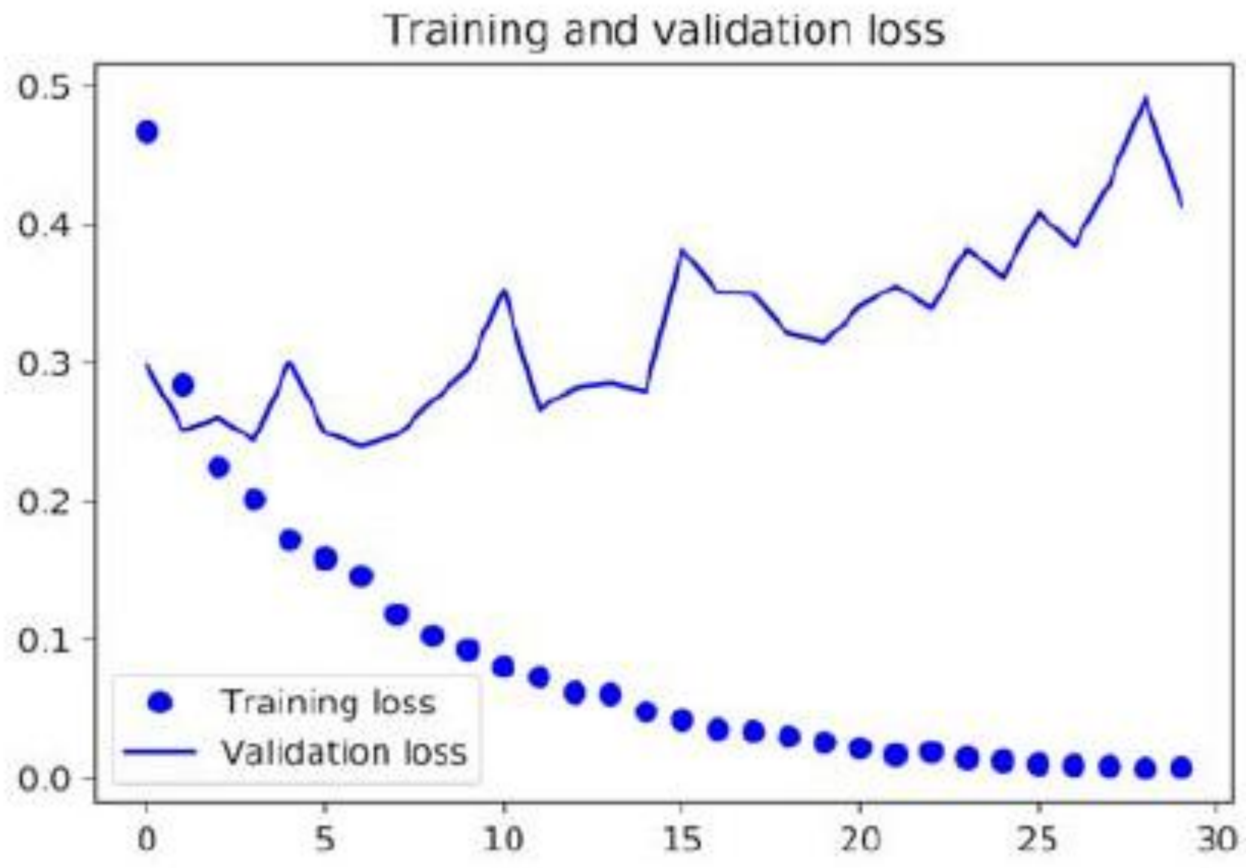
Plotting the Accuracy

```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Training and Validation accuracy



Training and Validation Loss



Complete codes

```
import keras
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))
conv_base.summary()

import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator

base_dir = './images/kaggle_dogs_vs_cats/'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20
```

```
def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i += 1
        if i * batch_size >= sample_count:
            # Note that since generators yield data indefinitely in a loop,
            # we must `break` after every image has been seen once.
            break
    return features, labels
```

```
train_features, train_labels = extract_features(train_dir, 2000)
validation_features, validation_labels = extract_features(validation_dir, 1000)
test_features, test_labels = extract_features(test_dir, 1000)

train_features = np.reshape(train_features, (2000, 4 * 4 * 512))
validation_features = np.reshape(validation_features, (1000, 4 * 4 * 512))
test_features = np.reshape(test_features, (1000, 4 * 4 * 512))
```

Train new classifier

```
from keras import models
from keras import layers
from keras import optimizers

model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=20,
                    validation_data=(validation_features, validation_labels))
```

Plot performance

```
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

Fine Tuning

We have studied feature extractor using deep learning, the fine tuning is another technique for transfer learning

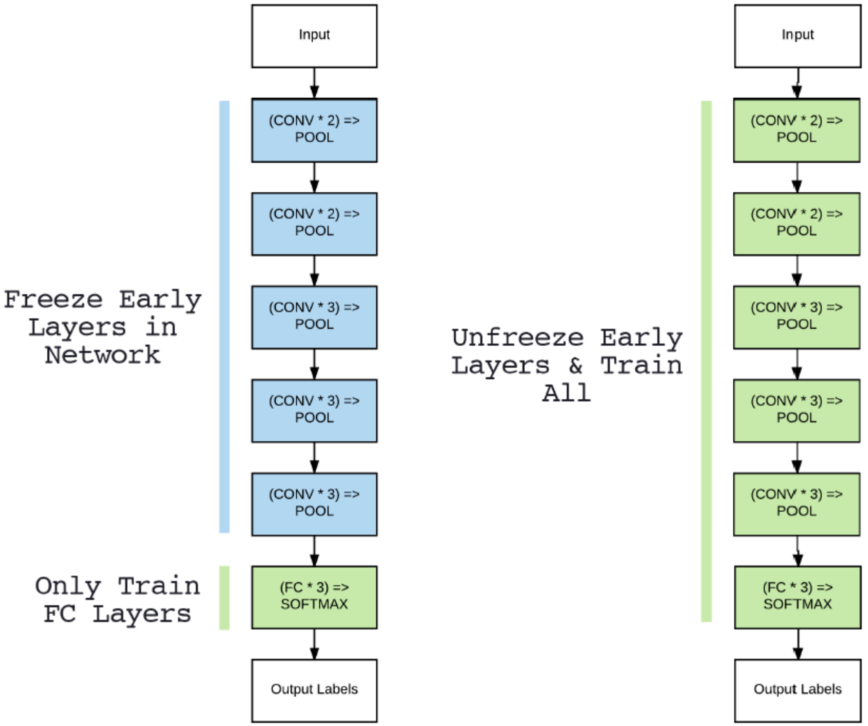
- We take fully-connected layers (head of the network) from a pre-trained CNN
- We replace the head with a new set of fully-connected layers with random weight
- All layers below the head are frozen, and we retrain the new head with a small learning rate.

Fine Tuning

- Add your custom network on top of an already-trained base network
- Freeze the base network
- Train the part you added
- Unfreeze some layers in the base network
- Jointly train both these layers and the part you added

Freezing and Unfreezing

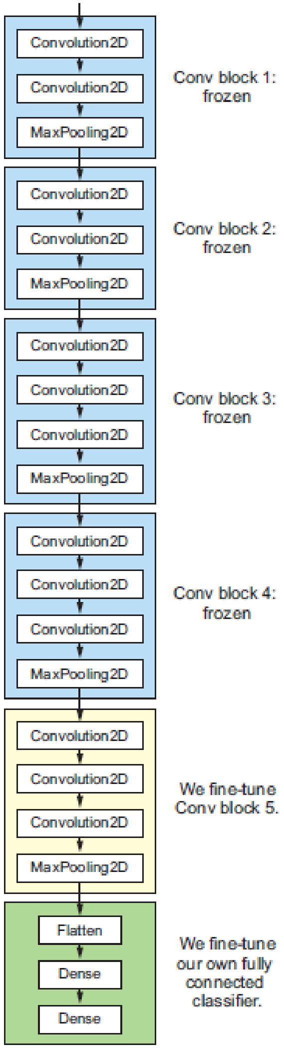
- Freezing is to keep weight unchanged
- Unfreeze layers should be trained with very small learning rate



Another Fine Tuning

- Freezing the code

```
conv_base.trainable = True

set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```



Model Summary

```
>>> conv_base.summary()
Layer (type)                     Output Shape          Param #
================================================================
input_1 (InputLayer)             (None, 150, 150, 3)   0
________________________________________________________________
block1_conv1 (Convolution2D)     (None, 150, 150, 64)  1792
________________________________________________________________
block1_conv2 (Convolution2D)     (None, 150, 150, 64)  36928
________________________________________________________________
block1_pool (MaxPooling2D)       (None, 75, 75, 64)    0
________________________________________________________________
block2_conv1 (Convolution2D)     (None, 75, 75, 128)   73856
________________________________________________________________
block2_conv2 (Convolution2D)     (None, 75, 75, 128)   147584
________________________________________________________________
block2_pool (MaxPooling2D)       (None, 37, 37, 128)   0
________________________________________________________________
block3_conv1 (Convolution2D)     (None, 37, 37, 256)   295168
________________________________________________________________
block3_conv2 (Convolution2D)     (None, 37, 37, 256)   590080
________________________________________________________________
block3_conv3 (Convolution2D)     (None, 37, 37, 256)   590080
________________________________________________________________
block3_pool (MaxPooling2D)       (None, 18, 18, 256)   0
________________________________________________________________
block4_conv1 (Convolution2D)     (None, 18, 18, 512)   1180160
________________________________________________________________
block4_conv2 (Convolution2D)     (None, 18, 18, 512)   2359808
________________________________________________________________
block4_conv3 (Convolution2D)     (None, 18, 18, 512)   2359808
________________________________________________________________
block4_pool (MaxPooling2D)       (None, 9, 9, 512)     0

________________________________________________________________
block5_conv1 (Convolution2D)     (None, 9, 9, 512)     2359808
________________________________________________________________
block5_conv2 (Convolution2D)     (None, 9, 9, 512)     2359808
________________________________________________________________
block5_conv3 (Convolution2D)     (None, 9, 9, 512)     2359808
________________________________________________________________
block5_pool (MaxPooling2D)       (None, 4, 4, 512)     0
================================================================
Total params: 14714688
```

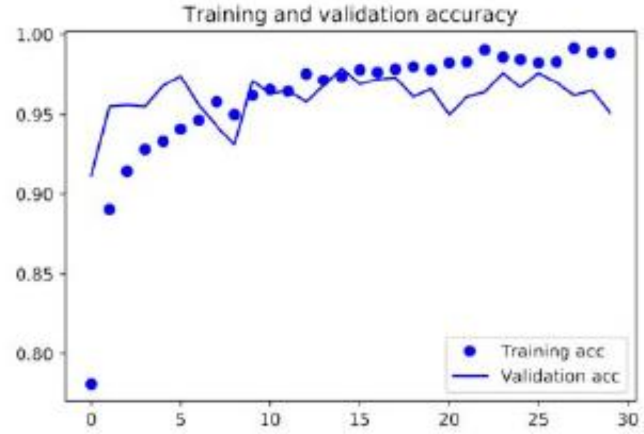
Freezing Layers

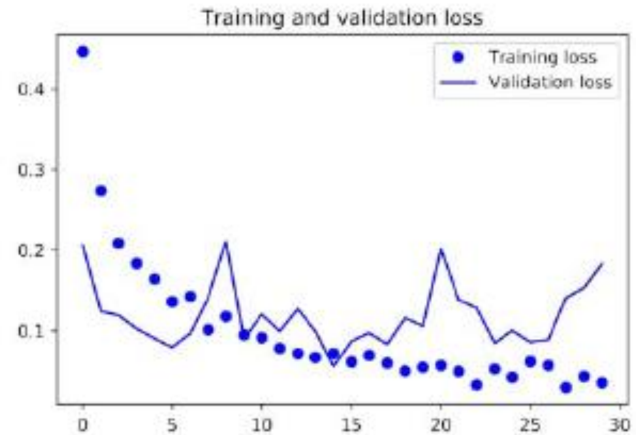
```
conv_base.trainable = True

set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

Fine Tuning

```
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5),
              metrics=['acc'])

history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50)
```

Training/Validation Accuracy and Loss





Testing Accuracy

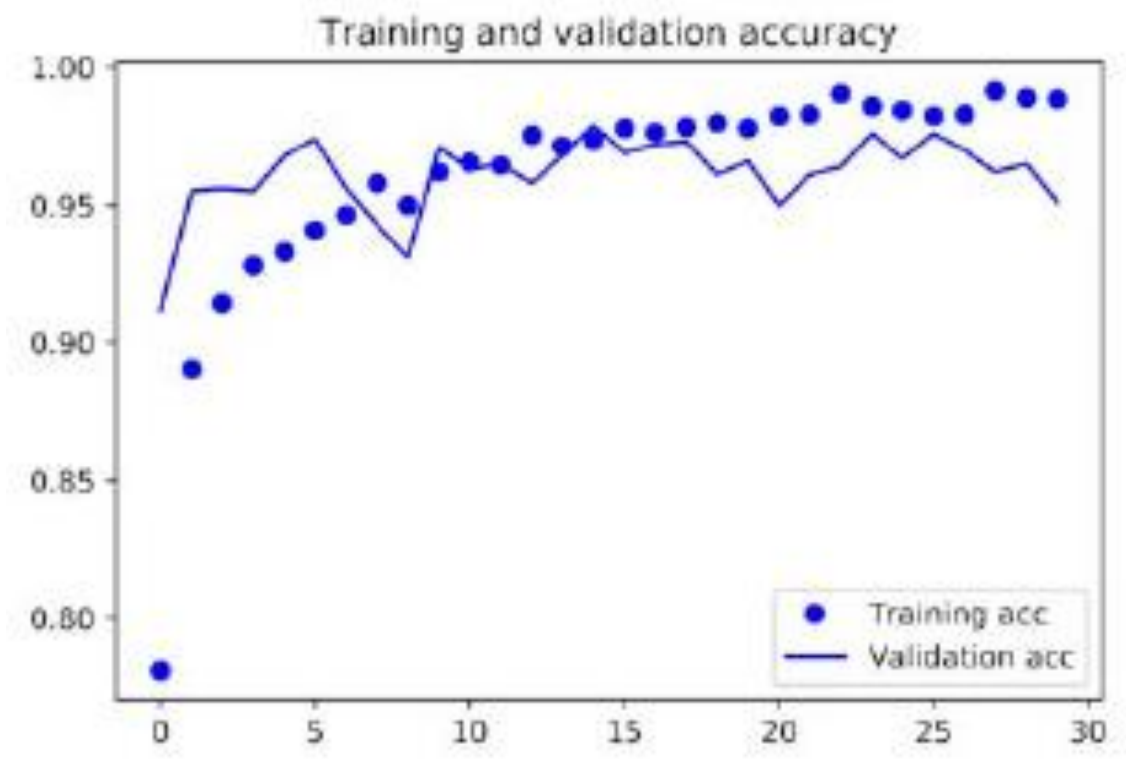
```
test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

test_loss, test_acc = model.evaluate_generator(test_generator, steps=50)
print('test acc:', test_acc)
```

Fine-tuning the model

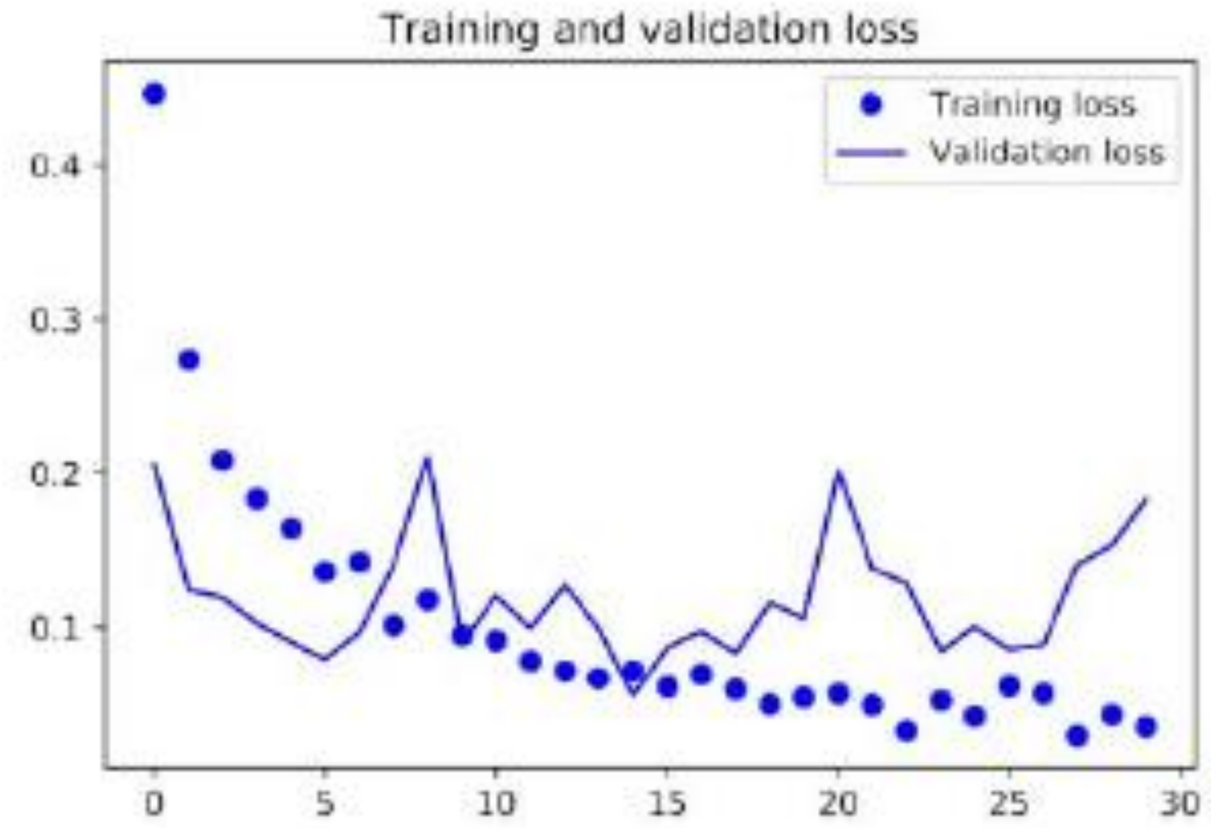
```
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5),
              metrics=['acc'])

history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50)
```

- We should set very low learning rate. The reason is that you want to limit the magnitude of the modifications

Training and Validation Accuracy



Training and Validation Loss



Completed code

```
import keras
keras.__version__
from keras.applications import VGG16
from keras import optimizers
from keras.preprocessing.image import ImageDataGenerator
import os
import numpy as np
from keras import models
from keras import layers

model = models.Sequential()
base_dir = './images/kaggle_dogs_vs_cats/'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')
```

```
train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150),
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')
validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

```python
conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))
conv_base.summary()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
conv_base.trainable = True

set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5),
              metrics=['acc'])
```

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50)
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))
import matplotlib.pyplot as plt
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

Fine Tuning Advantages

- Fine tuning is super powerful method to obtain good accuracy from already trained network
- Can work with even small datasets

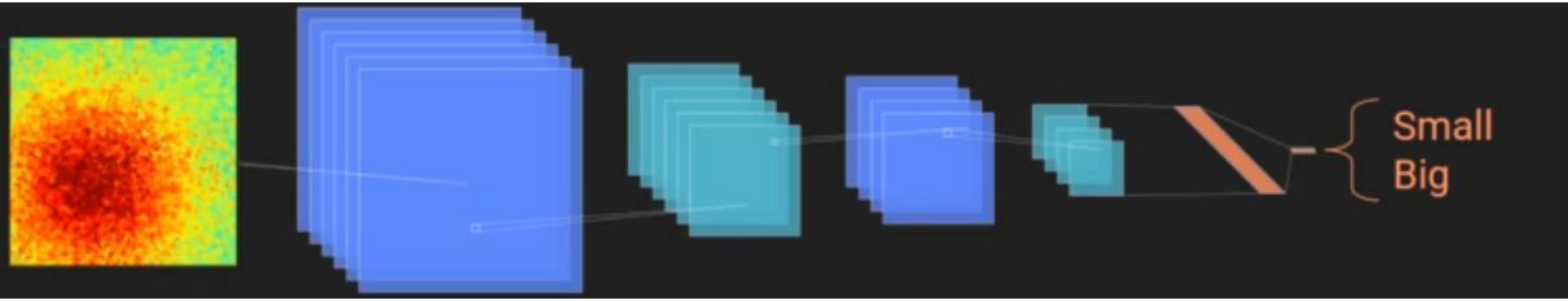
Transfer Learning or Train from Scatch

	Similar Dataset	Different Dataset
Small Dataset	Feature extraction using FC layers + classifier	Feature extraction using lower level CONV layers + classifier
Large Dataset	Fine-tuning likely to work, but might have to train from scratch	Fine-tuning worth trying, but will likely not work; likely have to train from scratch

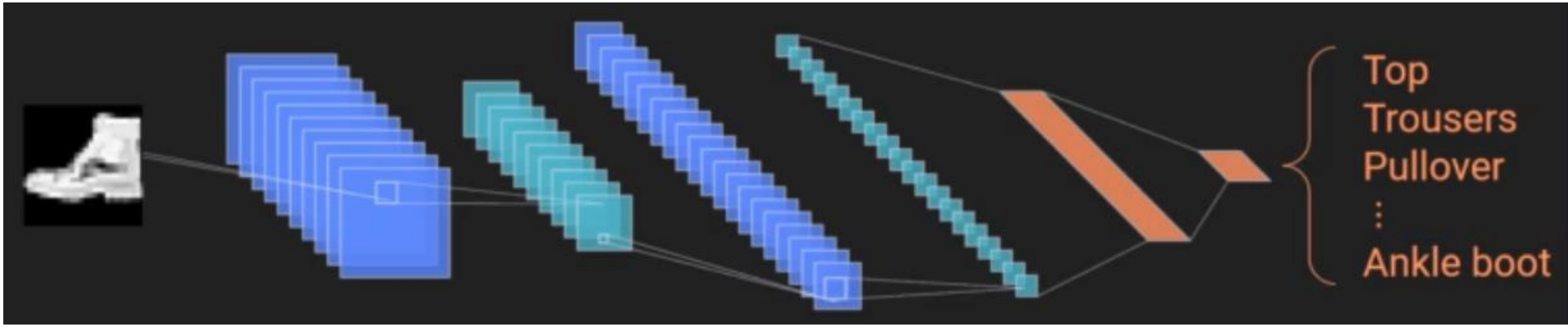
When is transfer learning useful?

- Use model that is trained from Huge datasets to solve similar problems
- Fine-tune existing trained model to your unique data
- Deploy model quickly with limited computing and data resources
- The model is deep

Is this good Transfer Learning?



Transfer Learning



Is transfer learning appropriate here?

- Fine tune AlexNet to distinguish different models of BMWs

- Fine tune AlexNet to predict credit card fraud

- Train a model on using US housing sales data and fine-tune to Bangkok housing sales data

Homework

- Implement transfer learning from mnist to mnisth cloth on your CNN network

Questions?