SCHOOL OF
HUMAN SCIENCES
& TECHNOLOGY

Transfer Learning

Chapter Goals

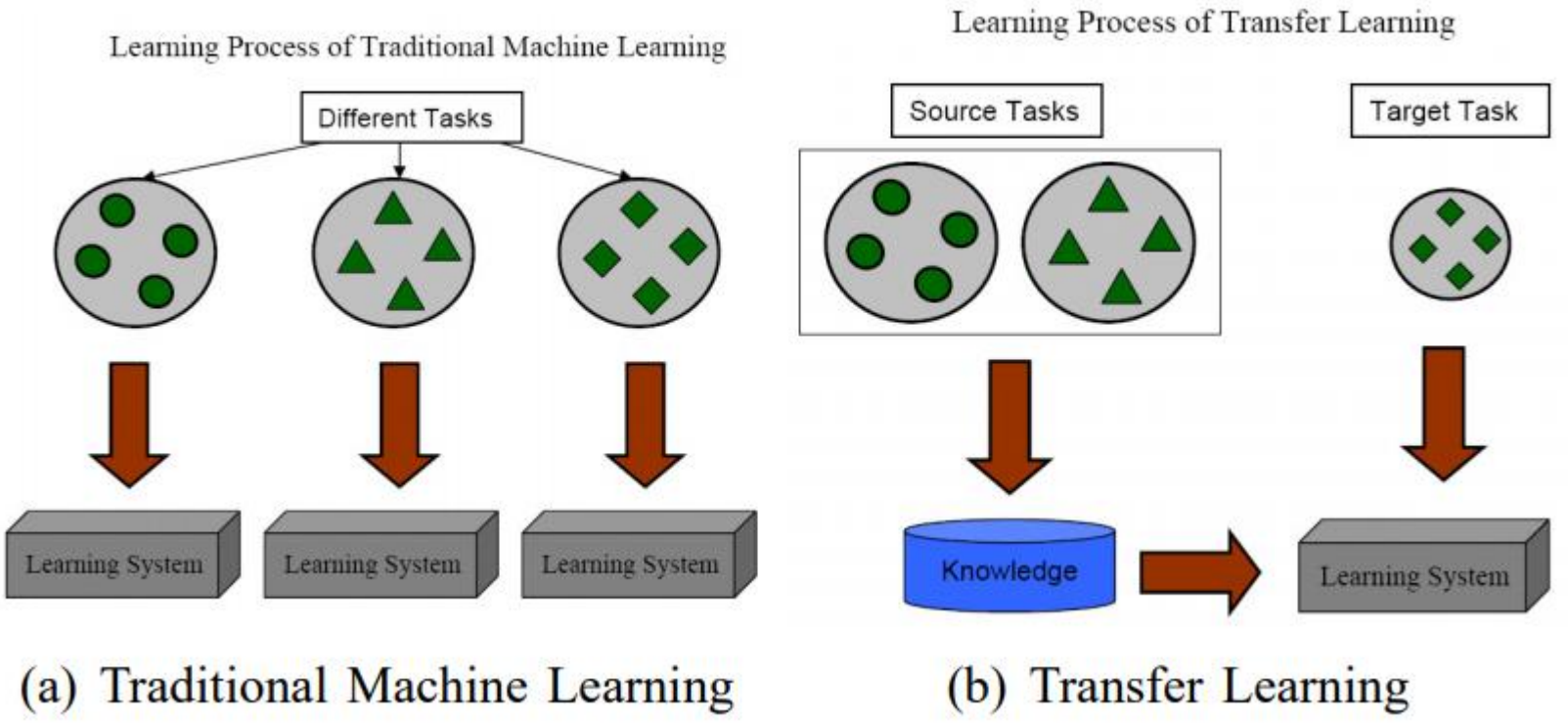
After completing this chapter, you should be able to understand :

- Transfer learning
- Apply it using Keras framework

ie
SCHOOL OF
HUMAN SCIENCES
& TECHNOLOGY

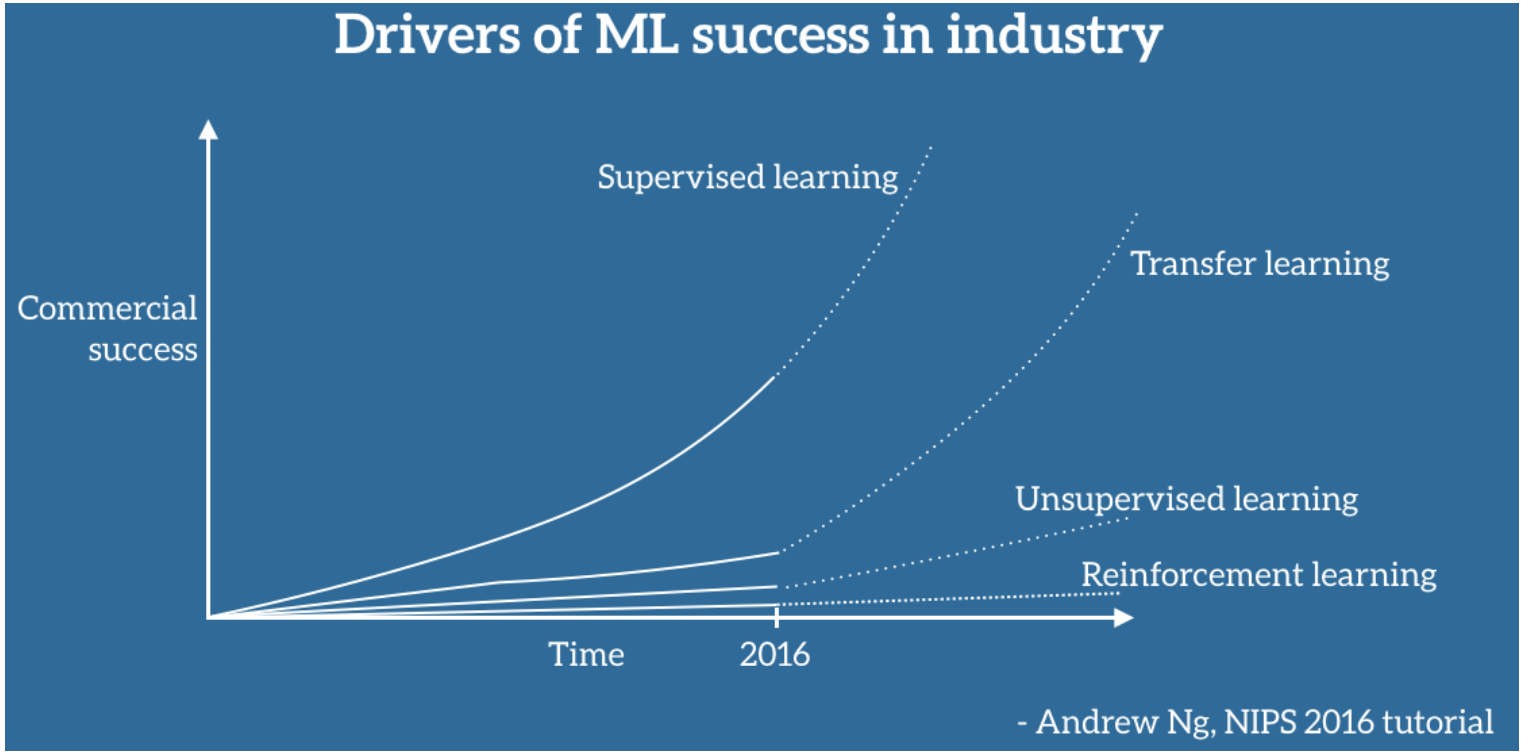
Transfer Learning

Traditional vs Transfer Learning



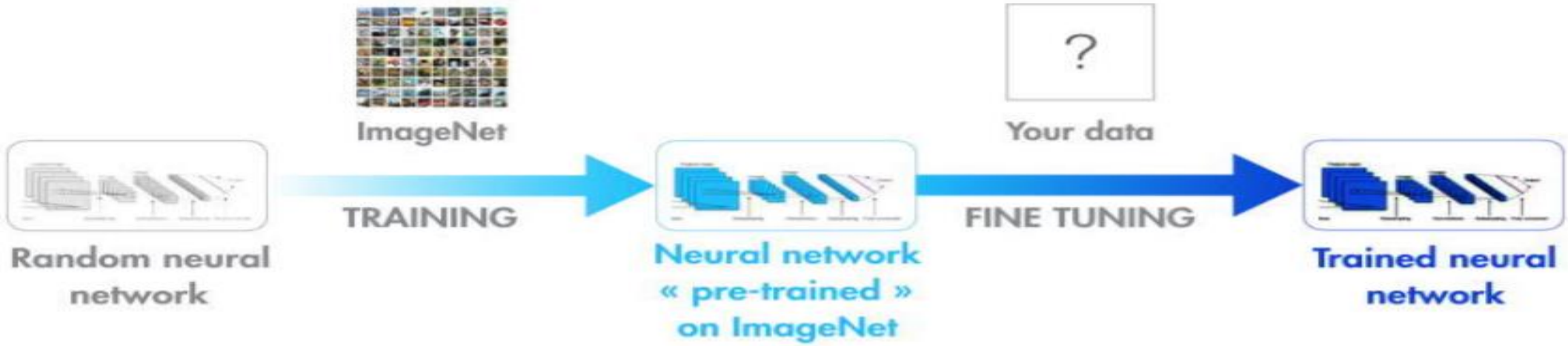
- Transfer learning is a method where a model developed for a task is reused as the starting point for a model on a second task.
- Transfer learning allows us to deal with these scenarios by leveraging the already existing labeled data of some related task or domain. We try to store this knowledge gained in solving the source task in the source domain and apply it to our problem of interest

Motivation for Transfer Learning

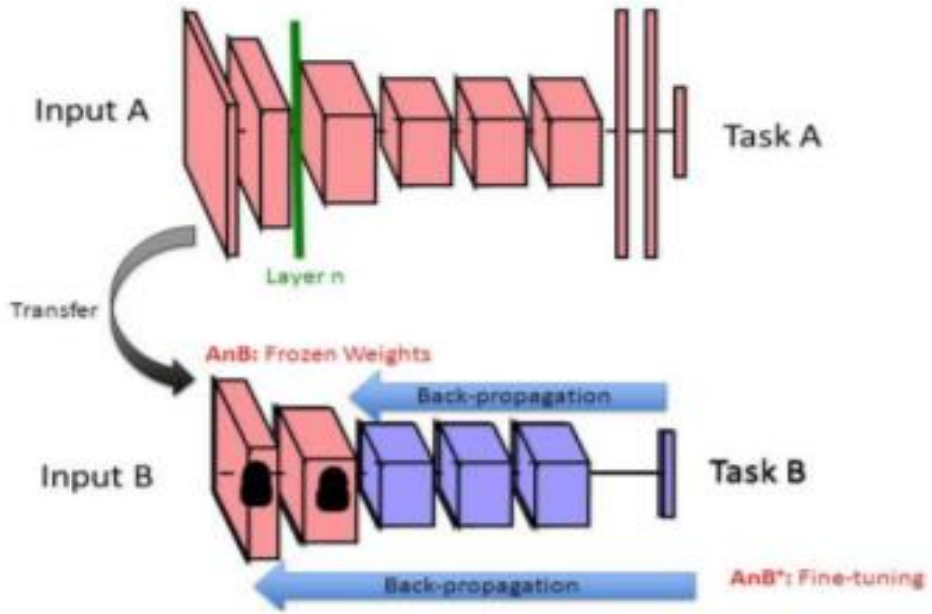


Andrew Ng, chief scientist at Baidu and professor at Stanford, said during his widely popular NIPS 2016 tutorial that transfer learning will be -- after supervised learning -- the next driver of ML commercial success.

Transfer Learning

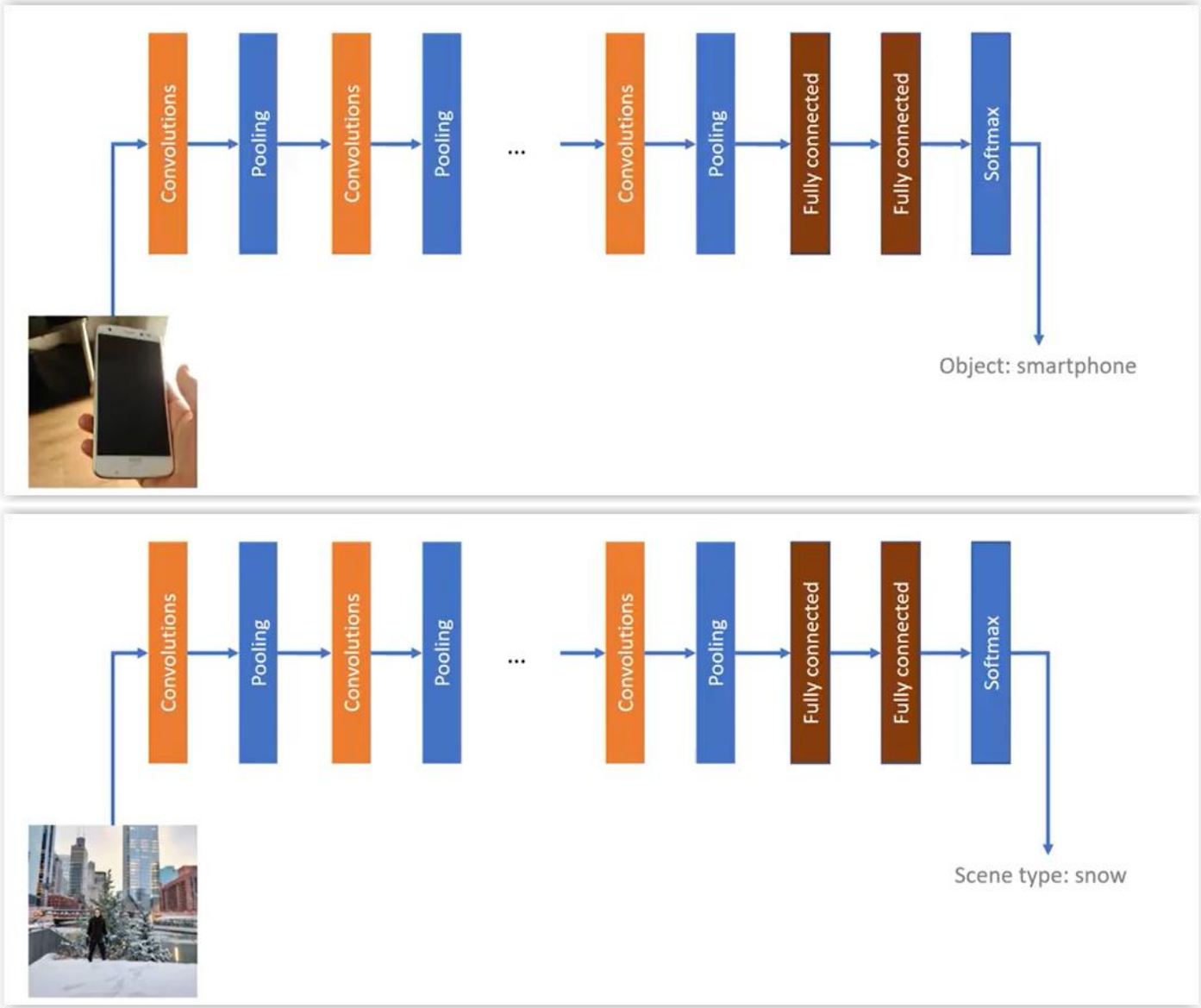


- Using the pre-trained model as a feature extractor, the idea is to train the model by replacing the last layer of the network with customized classifier. It is important to freeze (not change) the weights of all the other layers during gradient descent/optimization.
- If task specific dataset is quite different from the dataset used for the original model, then more high layers need to be trained and only a couple of the low layers will be frozen.



Transfer Learning with Pretrained Models

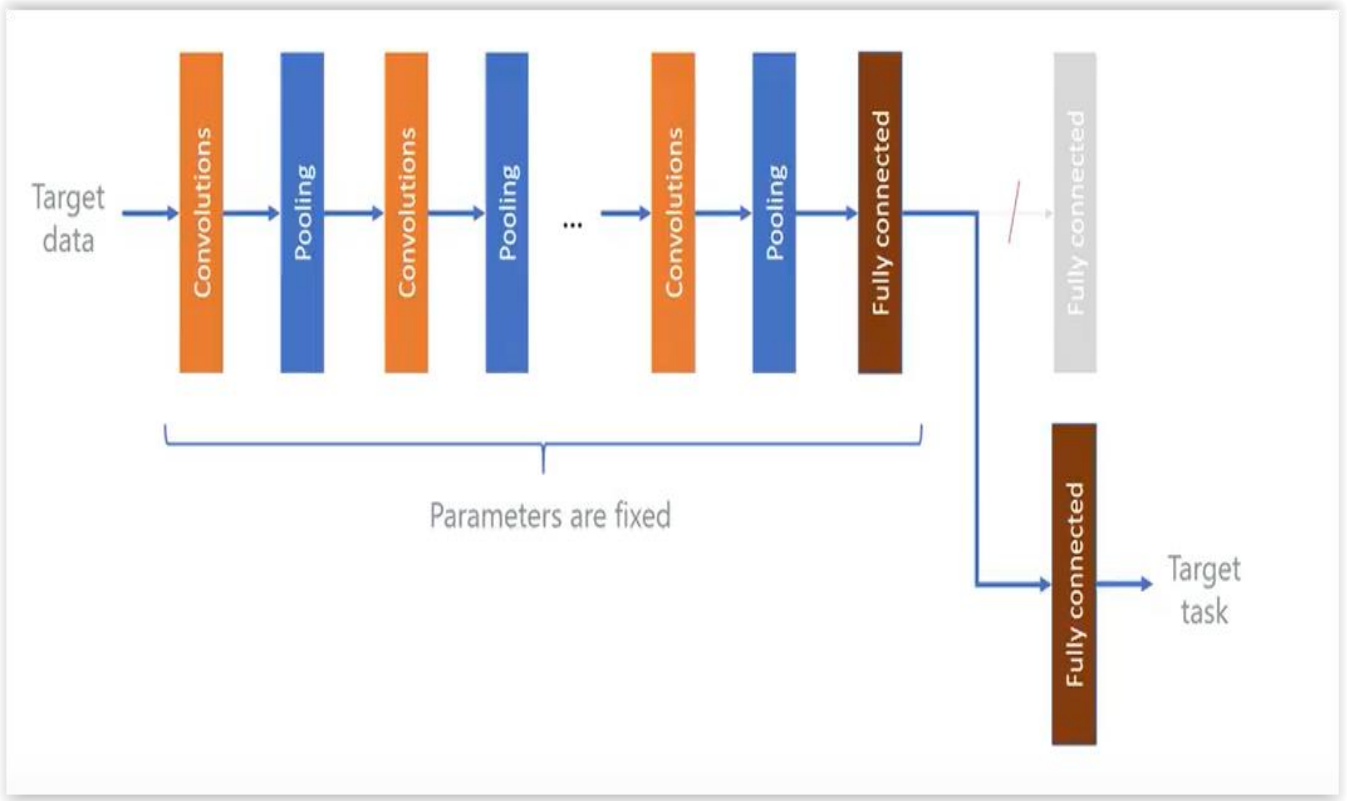
We can use a model trained for classifying images, to classify scenes



TL is particularly relevant in computer vision tasks

Transfer Learning with Pretrained Models

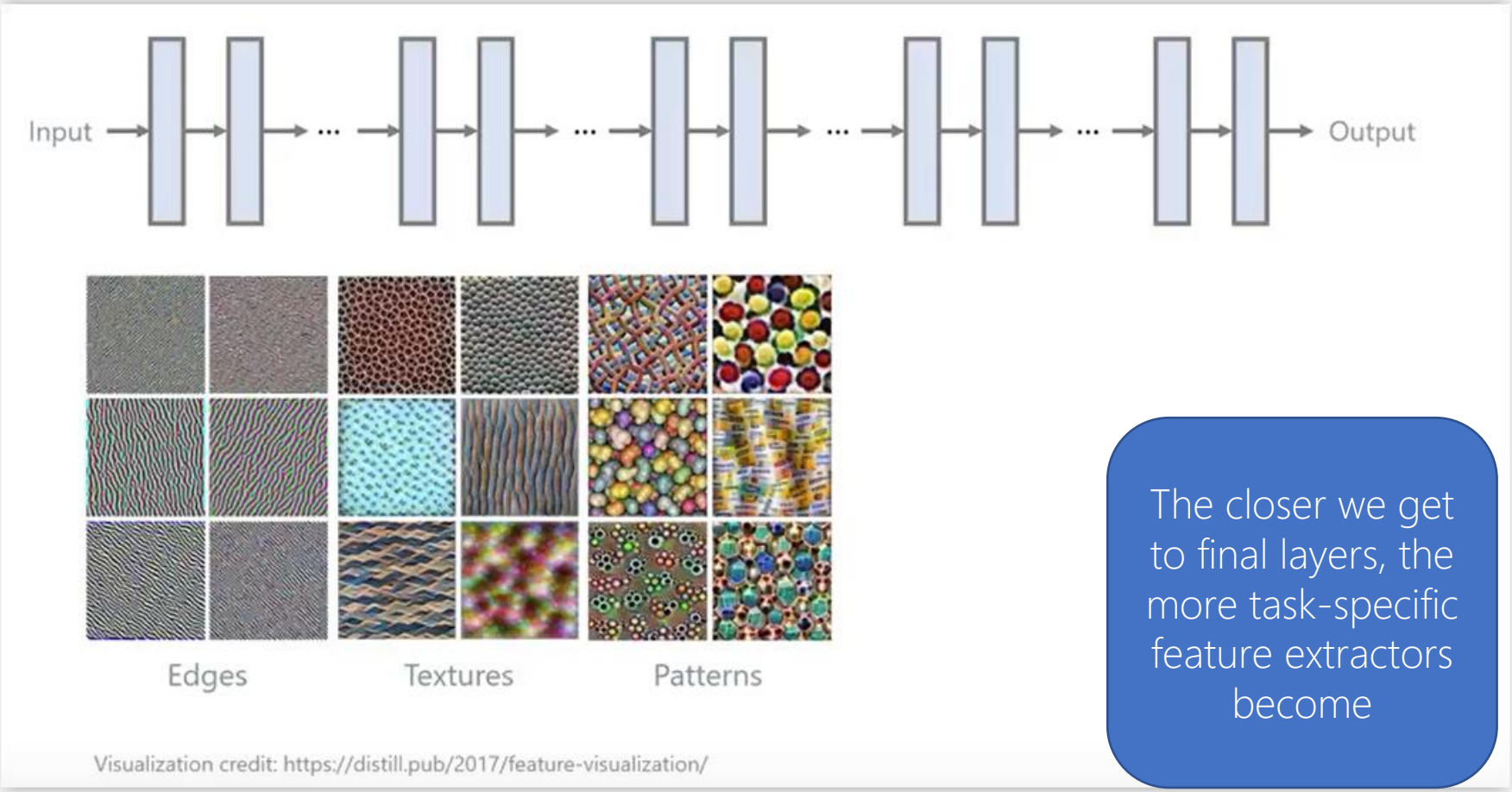
Remove top layer of already built model



Train parameters only on the top layer for the new task

Transfer Learning with Pretrained Models

Works because early layers capture similar features. Generic feature extractors that can be used in different settings



The closer we get to final layers, the more task-specific feature extractors become

Keras: Transfer Learning-Image Classification

Keras Applications - are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning. (https://keras.io/applications/)

- Weights are downloaded automatically when instantiating a model. They are stored at ~/.keras/models/. Models for image classification with weights trained on ImageNet:

- Xception
- VGG16
- VGG19
- ResNet50
- InceptionV3
- InceptionResNetV2
- MobileNet
- DenseNet
- NASNet

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88
DenseNet121	33 MB	0.745	0.918	8,062,504	121
DenseNet169	57 MB	0.759	0.928	14,307,880	169
DenseNet201	80 MB	0.770	0.933	20,242,984	201

Keras: Transfer Learning-Image Classification

First, we'll import the latest version of Keras and prepare to load our training data.

```
!pip install --upgrade keras

import keras
from keras import backend as K

print('Keras version:',keras.__version__)
```

Keras:Transfer Learning-Image Classification

Before we can train the model, we need to prepare the data

```
import os
from keras.preprocessing.image import ImageDataGenerator

# The images are in a folder named 'shapes/training'
training_folder_name = '../data/shapes/training'

# The folder contains a subfolder for each class of shape
classes = sorted(os.listdir(training_folder_name))
print(classes)

# Our source images are 128x128, but the base model we're going to use was trained with 224x224 images
pretrained_size = (224,224)
batch_size = 15

print("Getting Data...")
datagen = ImageDataGenerator(rescale=1./255, # normalize pixel values
                             validation_split=0.3) # hold back 30% of the images for validation

print("Preparing training dataset...")
train_generator = datagen.flow_from_directory(
    training_folder_name,
    target_size=pretrained_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training') # set as training data

print("Preparing validation dataset...")
validation_generator = datagen.flow_from_directory(
    training_folder_name,
    target_size=pretrained_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation') # set as validation data
```

```
['circle', 'square', 'triangle']
Getting Data...
Preparing training dataset...
Found 840 images belonging to 3 classes.
Preparing validation dataset...
Found 360 images belonging to 3 classes.
```

Keras: Transfer Learning-Image Classification

Download a trained model to use as a base

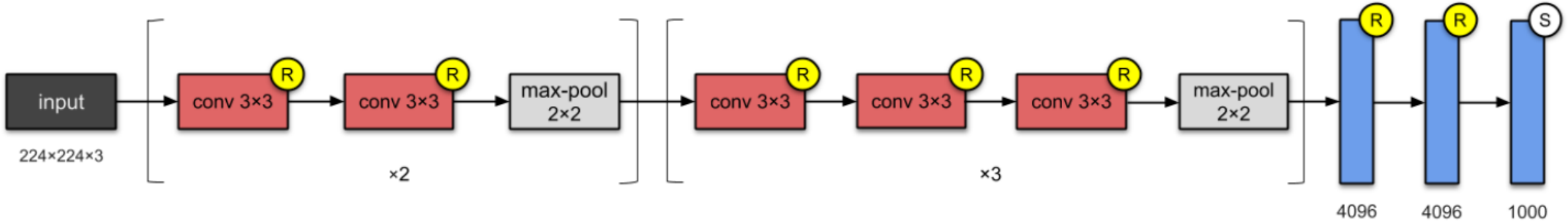
The VGG16 model is an image classifier that was trained on the ImageNet dataset - a huge dataset containing thousands of images of many kinds of object. We'll download the trained model, excluding its top layer, and set its input shape to match our image data.

*Note: The **keras.applications** namespace includes multiple base models, some which may perform better for your dataset than others. We can experiment with various models.*

```
from keras import applications
#Load the base model, not including its final connected layer, and set the input shape to match our images
base_model = keras.applications.vgg16.VGG16(weights='imagenet', include_top=False, input_shape=train_generator.image_shape)
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 12s 0us/step
```

3. VGG-16 (2014)



Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
VGG16	528 MB	0.713	0.901	138,357,544	23
InceptionV3	92 MB	0.779	0.937	23,851,784	159
ResNet50	98 MB	0.749	0.921	25,636,712	-
Xception	88 MB	0.790	0.945	22,910,480	126
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
ResNeXt50	96 MB	0.777	0.938	25,097,128	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Depth refers to the topological depth of the network. This includes activation layers, batch normalization layers etc.

Transfer Learning-Image Classification

Freeze the already trained layers and add a custom output layer for our classes

The existing feature extraction layers are already trained, so we just need to add a couple of layers so that the model output is the predictions for our classes.

```
from keras import Model
from keras.layers import Flatten, Dense
from keras import optimizers

# Freeze the already-trained layers in the base model
for layer in base_model.layers:
    layer.trainable = False

# Create layers for classification of our images
x = base_model.output
x = Flatten()(x)
prediction_layer = Dense(len(classes), activation='sigmoid')(x)
model = Model(inputs=base_model.input, outputs=prediction_layer)

# Compile the model
opt = optimizers.Adam(lr=0.001)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

# Now print the full model, which will include the layers of the base model plus the dense layer we added
print(model.summary())
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 75267
=================================================================
Total params: 14,789,955
Trainable params: 75,267
Non-trainable params: 14,714,688
_________________________________________________________________
None
```

Transfer Learning-Image Classification

Train the Model

With the layers of the CNN defined, we're ready to train the top layer using our image data. This will take a considerable amount of time on a CPU due to the complexity of the base model, so we'll train the model over only one epoch.

```
# Train the model over 1 epoch using 15-image batches and using the validation holdout dataset for validation
num_epochs = 1
history = model.fit_generator(
    train_generator,
    steps_per_epoch = train_generator.samples // batch_size,
    validation_data = validation_generator,
    validation_steps = validation_generator.samples // batch_size,
    epochs = num_epochs)
```

```
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Epoch 1/1
56/56 [==============================] - 1095s 20s/step - loss: 0.1171 - accuracy: 0.9571 - val_loss: 1.4179e-04 - val_accuracy: 1.0000
```

Transfer Learning-Image Classification

Using the Trained Model

Now that we've trained the model, we can use it to predict the class of an image.

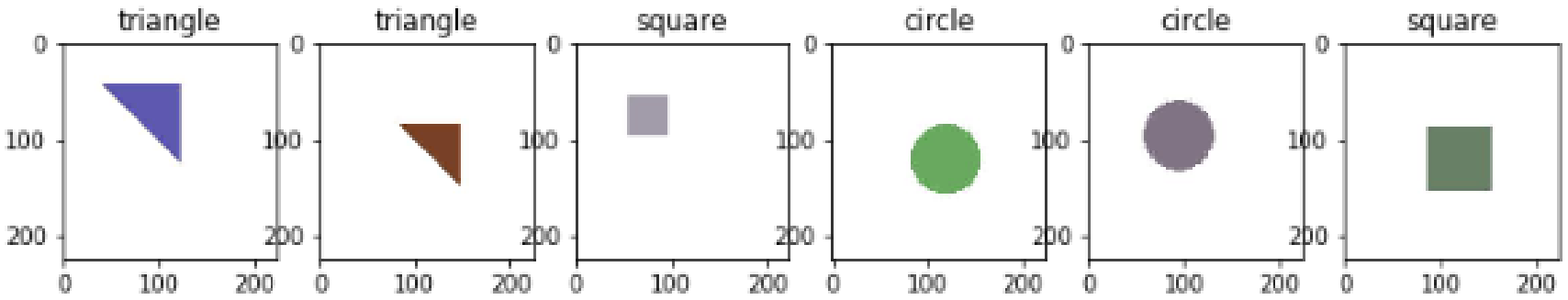
```
# Helper function to resize image
def resize_image(src_img, size=(128,128), bg_color="white"):
    from PIL import Image

    # rescale the image so the longest edge is the right size
    src_img.thumbnail(size, Image.ANTIALIAS)

    # Create a new image of the right shape
    new_image = Image.new("RGB", size, bg_color)

    # Paste the rescaled image onto the new background
    new_image.paste(src_img, (int((size[0] - src_img.size[0]) / 2), int((size[1] - src_img.size[1]) / 2)))

    # return the resized image
    return new_image

# Function to predict the class of an image
def predict_image(classifier, image_array):
    import numpy as np

    # We need to format the input to match the training data
    # The data generator loaded the values as floating point numbers
    # and normalized the pixel values, so...
    img_features = image_array.astype('float32')
    img_features /= 255

    # These are the classes our model can predict
    classnames = ['circle', 'square', 'triangle']

    # Predict the class of each input image
    predictions = classifier.predict(img_features)

    predicted_classes = []
    for prediction in predictions:
        # The prediction for each image is the probability for each class, e.g. [0.8, 0.1, 0.2]
        # So get the index of the highest probability
        class_idx = np.argmax(prediction)
        # And append the corresponding class name to the results
        predicted_classes.append(classnames[int(class_idx)])
    # Return the predictions
    return predicted_classes

print("Functions created - ready to use model for inference.")
```

```
Functions created - ready to use model for inference.
```

```
import os
from random import randint
import numpy as np
from PIL import Image
from keras.models import load_model
from matplotlib import pyplot as plt
%matplotlib inline

#get the list of test image files
test_folder = '../data/shapes/test'
test_image_files = os.listdir(test_folder)

# Empty array on which to store the images
image_arrays = []

size = (224,224)
background_color="white"

fig = plt.figure(figsize=(12, 8))

# Get the images and show the predicted classes
for file_idx in range(len(test_image_files)):
    img = Image.open(os.path.join(test_folder, test_image_files[file_idx]))

    # resize the image so it matches the training set - it  must be the same size as the images on which the model was trained
    resized_img = np.array(resize_image(img, size, background_color))

    # Add the image to the array of images
    image_arrays.append(resized_img)

# Get predictions from the array of image arrays
# Note that the model expects an array of 1 or more images - just like the batches on which it was trained
predictions = predict_image(model, np.array(image_arrays))

# plot easch image with its corresponding prediction
for idx in range(len(predictions)):
    a=fig.add_subplot(1,len(predictions),idx+1)
    imgplot = plt.imshow(image_arrays[idx])
    a.set_title(predictions[idx])
```

Transfer Learning-Image Classification

Using the Trained Model



ie
SCHOOL OF
HUMAN SCIENCES
& TECHNOLOGY

Chapter Summary

- We saw Transfer learning
- We explored how to apply it using Keras framework