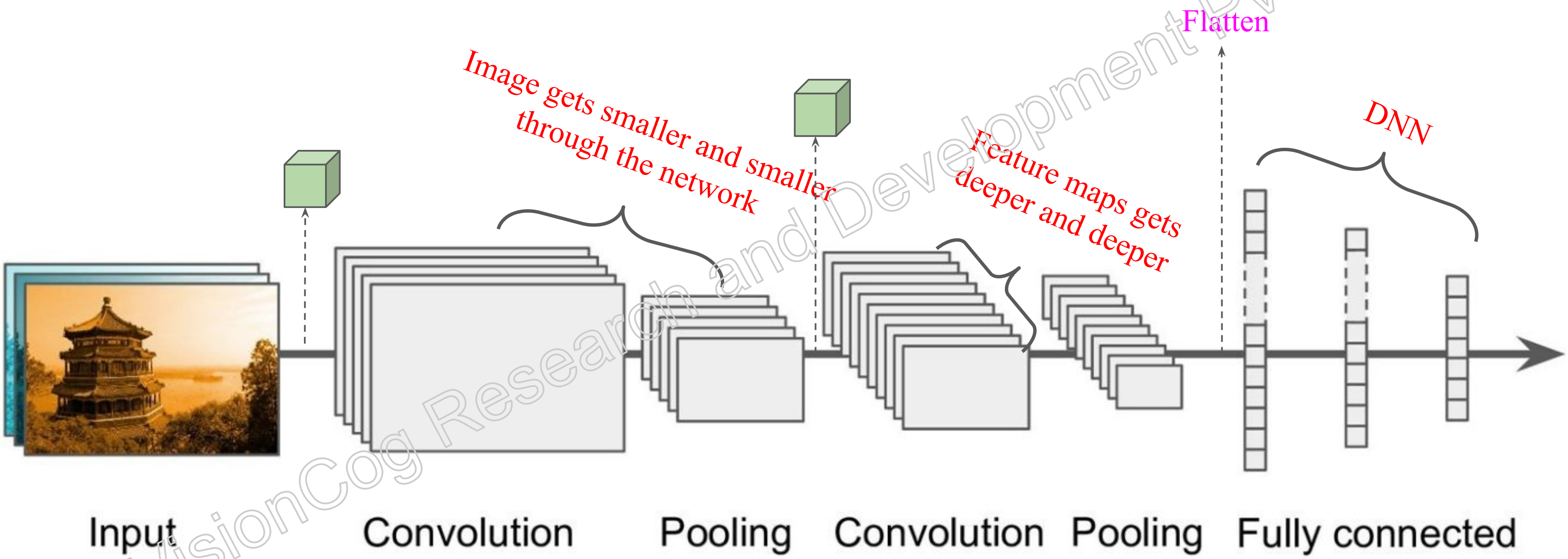
TRANSFER LEARNING

Dr. Ram Prasad K
VisionCog R&D

ram.krish@visioncog.com
https://www.visioncog.com

ConvNets

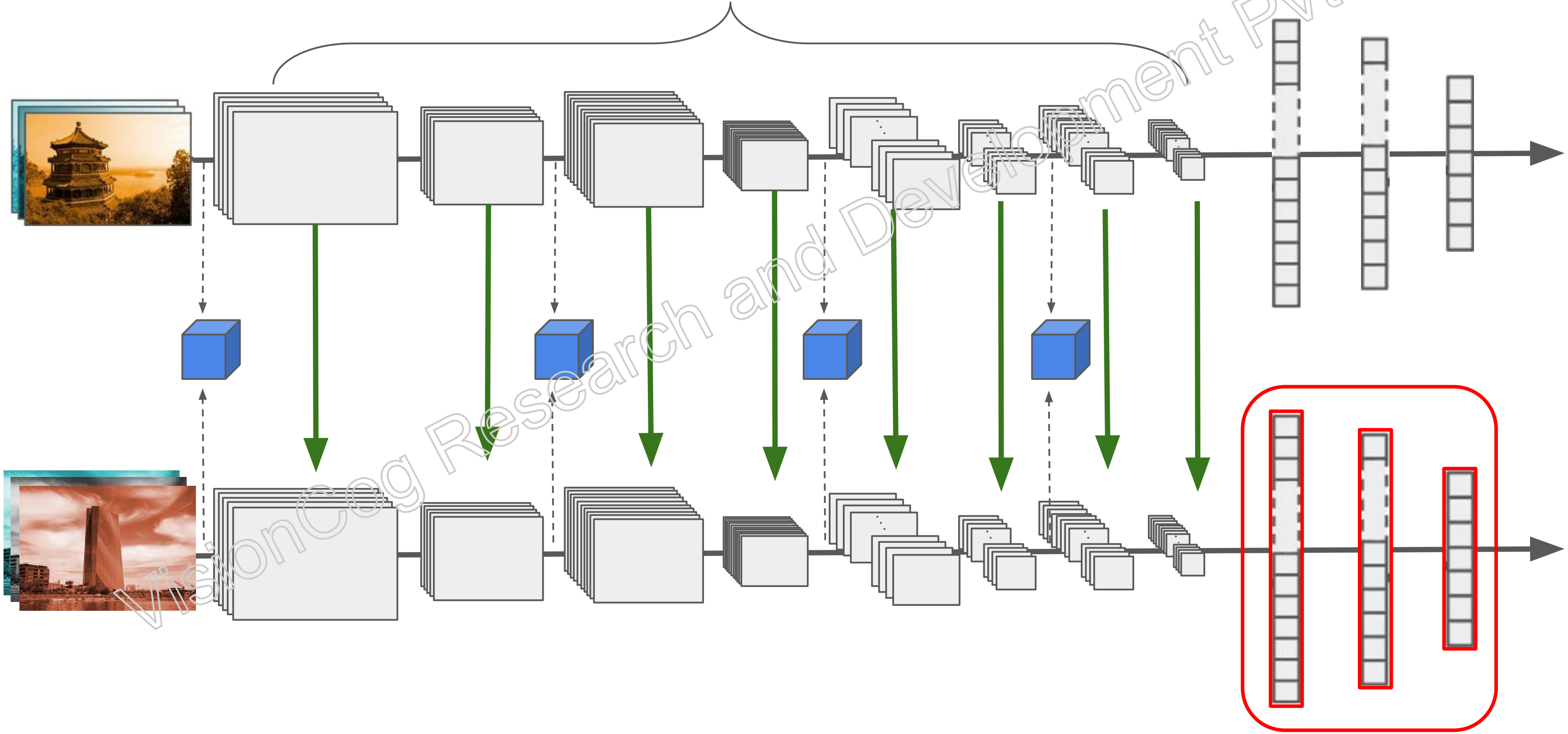
CNN architecture



Transfer Learning

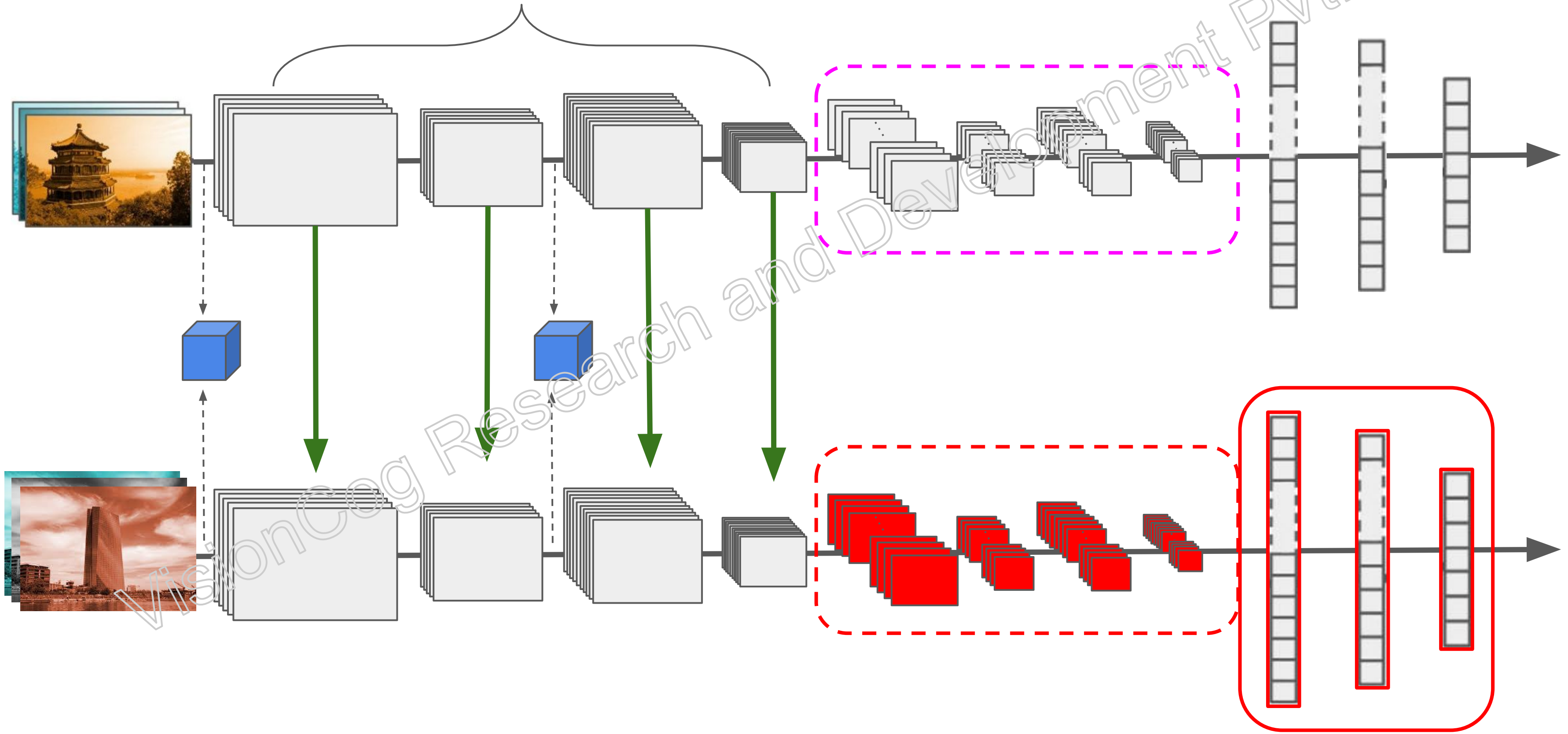
ConvNets

Transfer learning



ConvNets

Transfer learning and Fine-Tuning



Xception: Deep Learning with Depthwise Separable Convolutions

François Chollet
Google, Inc.
fchollet@google.com

Abstract

We present an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution). In this light, a depthwise separable convolution can be understood as an Inception module with a maximally large number of towers. This observation leads us to propose a novel deep convolutional neural network architecture inspired by Inception, where Inception modules have been replaced with depthwise separable convolutions. We show that this architecture, dubbed Xception, slightly outperforms Inception V3 on the ImageNet dataset (which Inception V3 was designed for), and significantly outperforms Inception V3 on a larger image classification dataset comprising 350 million images and 17,000 classes. Since the Xception architecture has the same number of parameters as Inception V3, the performance gains are not due to increased capacity but rather to a more efficient use of model parameters.

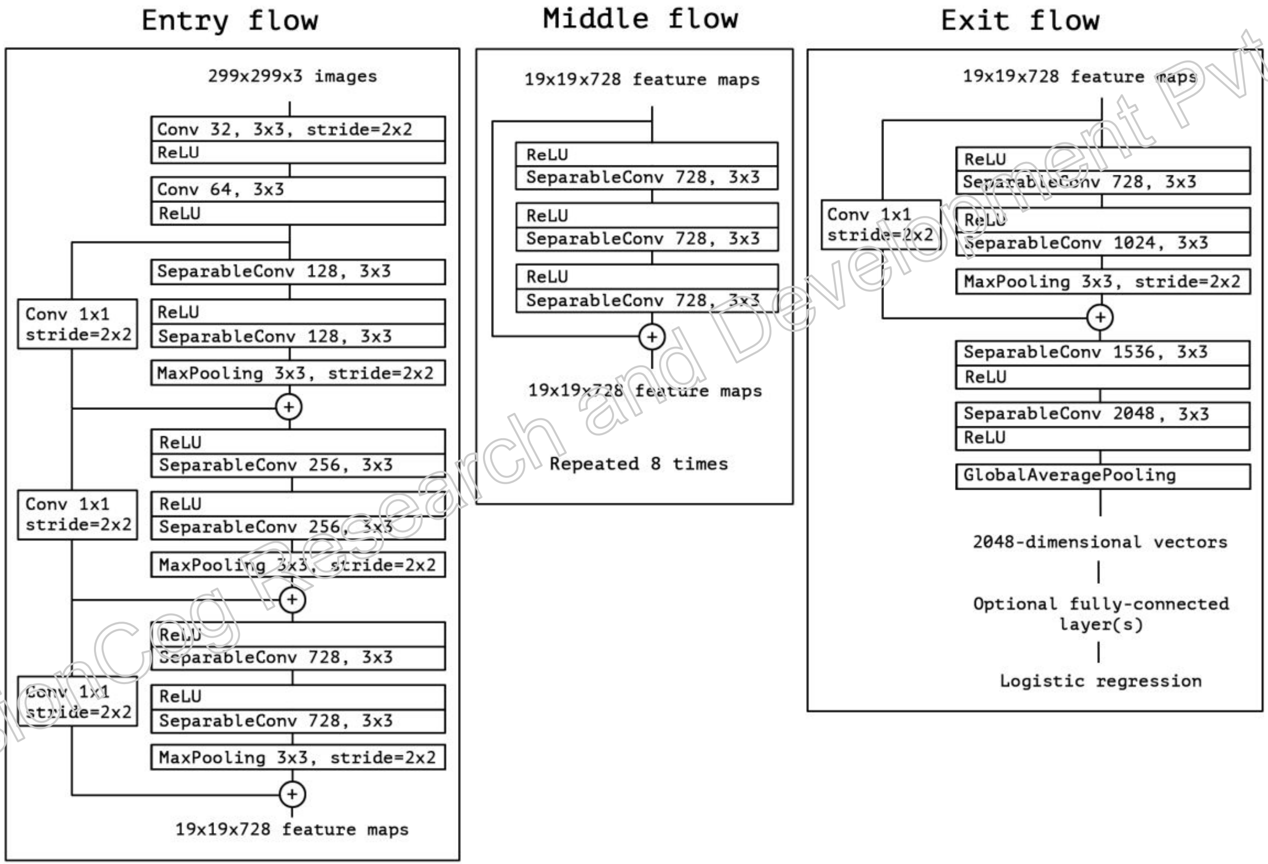
as GoogLeNet (Inception V1), later refined as Inception V2 [7], Inception V3 [21], and most recently Inception-ResNet [19]. Inception itself was inspired by the earlier Network-In-Network architecture [11]. Since its first introduction, Inception has been one of the best performing family of models on the ImageNet dataset [14], as well as internal datasets in use at Google, in particular JFT [5].

The fundamental building block of Inception-style models is the Inception module, of which several different versions exist. In figure 1 we show the canonical form of an Inception module, as found in the Inception V3 architecture. An Inception model can be understood as a stack of such modules. This is a departure from earlier VGG-style networks which were stacks of simple convolution layers.

While Inception modules are conceptually similar to convolutions (they are convolutional feature extractors), they empirically appear to be capable of learning richer representations with less parameters. How do they work, and how do they differ from regular convolutions? What design strategies come after Inception?

1.1. The Inception hypothesis

ConvNets



Entry flow
299x299x3 images
Conv 32, 3x3, stride=2x2
ReLU
Conv 64, 3x3
ReLU
SeparableConv 128, 3x3
Conv 1x1
stride=2x2
ReLU
SeparableConv 128, 3x3
MaxPooling 3x3, stride=2x2
ReLU
SeparableConv 256, 3x3
Conv 1x1
stride=2x2
ReLU
SeparableConv 256, 3x3
MaxPooling 3x3, stride=2x2
ReLU
SeparableConv 728, 3x3
Conv 1x1
stride=2x2
ReLU
SeparableConv 728, 3x3
MaxPooling 3x3, stride=2x2
19x19x728 feature maps
Middle flow
19x19x728 feature maps
ReLU
SeparableConv 728, 3x3
ReLU
SeparableConv 728, 3x3
ReLU
SeparableConv 728, 3x3
19x19x728 feature maps
Repeated 8 times
Exit flow
19x19x728 feature maps
ReLU
SeparableConv 728, 3x3
Conv 1x1
stride=2x2
ReLU
SeparableConv 1024, 3x3
MaxPooling 3x3, stride=2x2
SeparableConv 1536, 3x3
ReLU
SeparableConv 2048, 3x3
ReLU
GlobalAveragePooling
2048-dimensional vectors
Optional fully-connected
layer(s)
Logistic regression

Flower classification

ConvNets

Flower Classification

daisy

ConvNets

Flower Classification

dandelion



ConvNets

Flower Classification

rose

ConvNets

Flower Classification

sunflower



ConvNets

Flower Classification

tulip

ConvNets

http://download.tensorflow.org/example_images/flower_photos.tgz (3,670)

Tiny version

https://www.visioncog.com/rpk/tiny_FR.zip (500)

Flower Classification
(100 each, size of image varies)

- daisy
- dandelion
- rose
- sunflower
- tulip

ConvNets

```
# Original dataset
# http://download.tensorflow.org/example_images/flower_photos.tgz

# Download tiny version of the dataset from VisionCog website
# After download and unzip, remember to comment the following two lines.
```

```
!wget https://www.visioncog.com/rpk/tiny_FR.zip
!unzip tiny_FR.zip
```

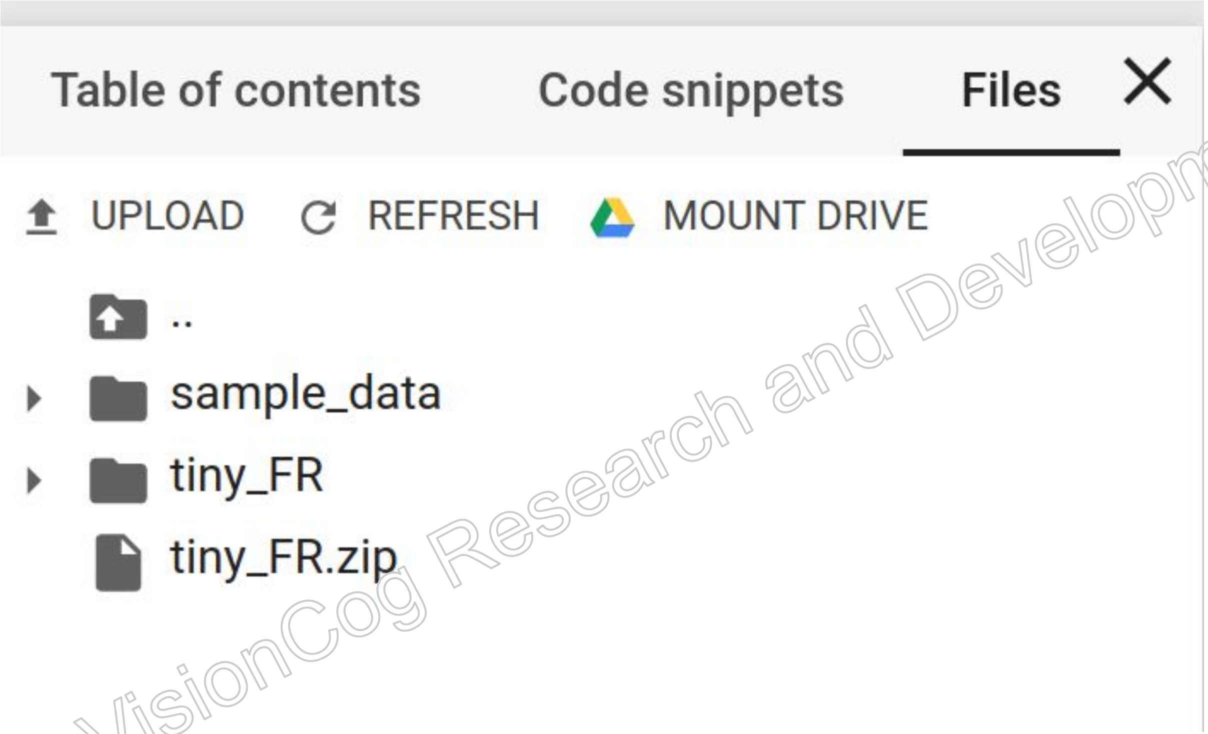


Table of contents
Code snippets
Files
UPLOAD
REFRESH
MOUNT DRIVE
..
sample_data
tiny_FR
tiny_FR.zip

ConvNets

```
# Original dataset
# http://download.tensorflow.org/example_images/flower_photos.tgz

# Download tiny version of the dataset from VisionCog website
# After download and unzip, remember to comment the following two lines.

#!wget https://www.visioncog.com/rpk/tiny_FR.zip
#!unzip tiny_FR.zip
```

ConvNets

```
# Install TensorFlow
try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass

import tensorflow as tf
print(tf.__version__)


# TensorFlow 2.x selected.
# 2.0.0-rc1
```

ConvNets

```
from tensorflow import keras
tf.random.set_seed(42)

import numpy as np
np.random.seed(42)

import matplotlib.pyplot as plt
%matplotlib inline

import glob
import PIL
from PIL import Image
```

ConvNets

```python
imgFiles = glob.glob("tiny_FR/*/*.jpg")

for items in imgFiles[:5]:
  print(items)

# tiny_FR/sunflower/1715303025_e7065327e2.jpg
# tiny_FR/sunflower/2442985637_8748180f69.jpg
# tiny_FR/sunflower/27466794_57e4fe5656.jpg
# tiny_FR/sunflower/40411019_526f3fc8d9_m.jpg
# tiny_FR/sunflower/253586685_ee5b5f5232.jpg
```

ConvNets

```
X = []
y = []

for fName in imgFiles:

  X_i = Image.open(fName) # tiny_FR/sunflower/1715303025_e7065327e2.jpg (500, 333)

  X_i = X_i.resize((299,299)) # To make them approriate to Xception model when using Transfer Learning

  X_i = np.array(X_i) / 255.0 # Normalize to range 0.0 to 1.0 (not stretching, only scaling)


  X.append(X_i)


  label = fName.split('/') # ['tiny_FR', 'sunflower', '1715303025_e7065327e2.jpg']

  y_i = label[1] # 'sunflower'


  y.append(y_i)
```

ConvNets

```
print(set(y))
# {'daisy', 'sunflower', 'dandelion', 'rose', 'tulip'}


from sklearn.preprocessing import LabelEncoder

lEncoder = LabelEncoder()
y = lEncoder.fit_transform(y)

print(set(y))
# {0, 1, 2, 3, 4}

print(lEncoder.classes_)
# ['daisy' 'dandelion' 'rose' 'sunflower' 'tulip']
```

ConvNets

```
X = np.array(X)
y = np.array(y)

print(X.shape)
# (500, 299, 299, 3)

print(y.shape)
# (500,)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    stratify=y, random_state=42)
```

ConvNets

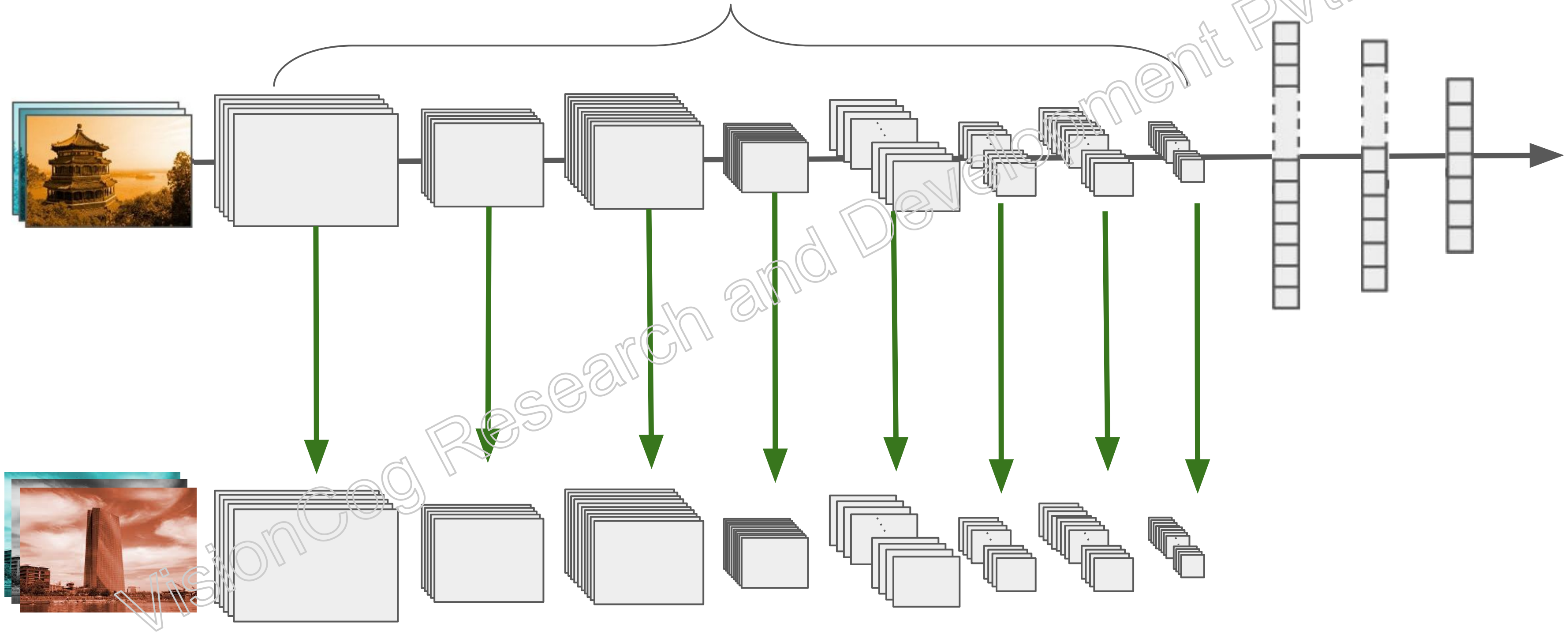
```
print("X_train_shape: {}".format(X_train.shape))
# X_train_shape: (400, 299, 299, 3)

print("X_test_shape: {}".format(X_test.shape))
# X_test_shape: (100, 299, 299, 3)


# Standard scaling
mu = X_train.mean()
std = X_train.std()

X_train_std = (X_train-mu)/std
X_test_std = (X_test-mu)/std
```

ConvNets

```
base_model = keras.applications.xception.Xception(weights='imagenet',
                                                   include_top=False)
```

ConvNets

Transfer learning

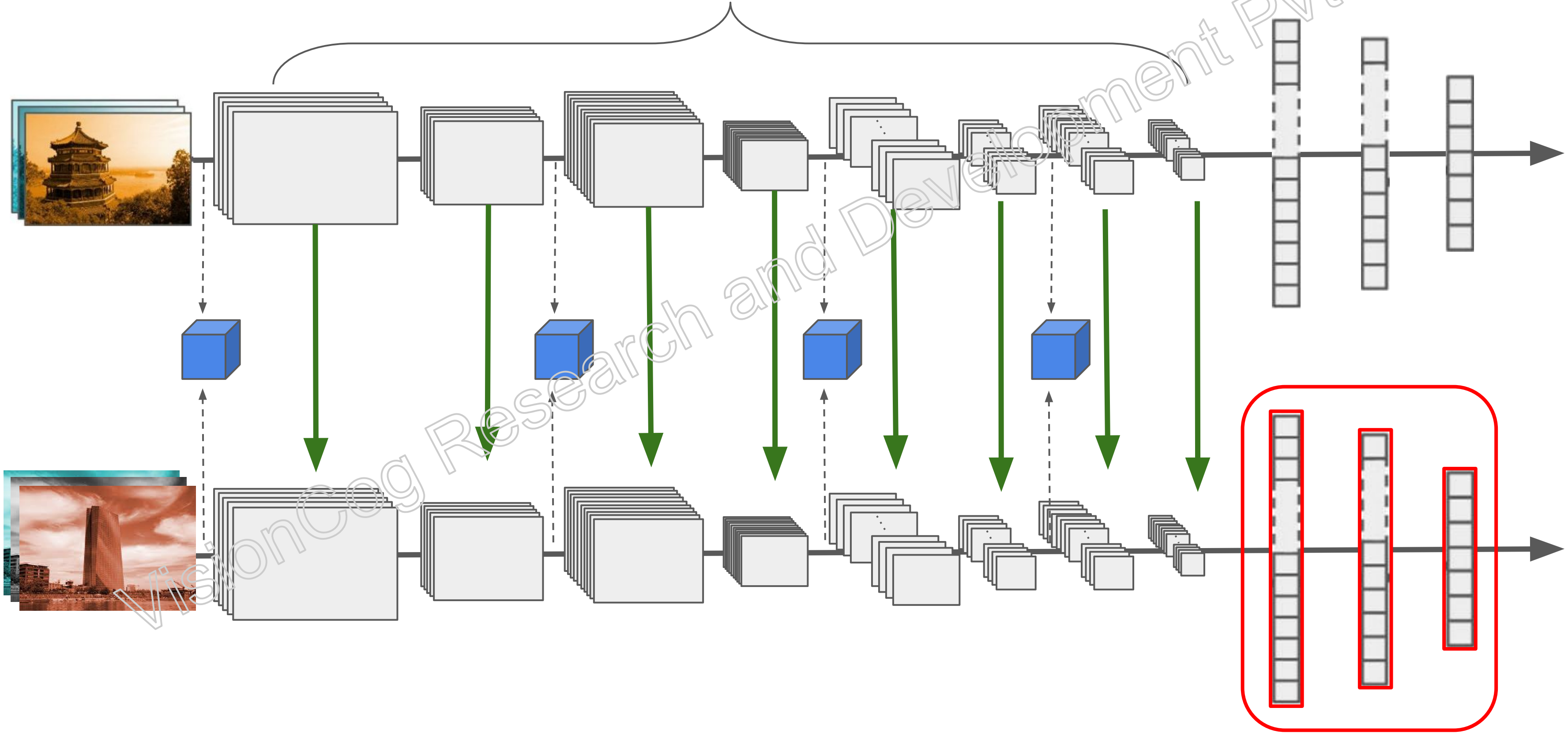


ConvNets

```
base_model = keras.applications.xception.Xception(weights='imagenet',
                                                  include_top=False)

for layer in base_model.layers:
  layer.trainabe = False

global_pool = keras.layers.GlobalAveragePooling2D()(base_model.output)
output_ = keras.layers.Dense(units=5, activation='softmax')(global_pool)

model_TL = keras.models.Model(inputs=[base_model.input], outputs=[output_])
```

ConvNets

Transfer learning



ConvNets

```
model_TL.compile(loss='sparse_categorical_crossentropy',
                 optimize='adam', metrics=['accuracy'])

history_TL = model_TL.fit(x = X_train_std, y = y_train, epochs=25,
                          validation_split=0.1, batch_size=16)
```

```
Train on 360 samples, validate on 40 samples

Epoch 1/25
360/360 [===]-27s 75ms/sample - loss: 0.9786 - accuracy: 0.6500 - val_loss: 7.0164 - val_accuracy: 0.5500
Epoch 2/25
360/360 [===]-15s 40ms/sample - loss: 0.4938 - accuracy: 0.8583 - val_loss: 10.1537 - val_accuracy: 0.3000
Epoch 3/25
360/360 [===]-14s 40ms/sample - loss: 0.3327 - accuracy: 0.8889 - val_loss: 6.9043 - val_accuracy: 0.4000

...

Epoch 24/25
360/360 [===]-14s 39ms/sample - loss: 0.0482 - accuracy: 0.9806 - val_loss: 1.0079 - val_accuracy: 0.7500
Epoch 25/25
360/360 [===]-14s 39ms/sample - loss: 0.0650 - accuracy: 0.9750 - val_loss: 7.1623 - val_accuracy: 0.6250
```

ConvNets

```
Train on 360 samples, validate on 40 samples

Epoch 1/25
360/360 [===]-27s 75ms/sample - loss: 0.9786 - accuracy: 0.6500 - val_loss: 7.0164 - val_accuracy: 0.5500
Epoch 2/25
360/360 [===]-15s 40ms/sample - loss: 0.4938 - accuracy: 0.8583 - val_loss: 10.1537 - val_accuracy: 0.3000
Epoch 3/25
360/360 [===]-14s 40ms/sample - loss: 0.3327 - accuracy: 0.8889 - val_loss: 6.9043 - val_accuracy: 0.4000

...

Epoch 24/25
360/360 [===]-14s 39ms/sample - loss: 0.0482 - accuracy: 0.9806 - val_loss: 1.0079 - val_accuracy: 0.7500
Epoch 25/25
360/360 [===]-14s 39ms/sample - loss: 0.0650 - accuracy: 0.9750 - val_loss: 7.1623 - val_accuracy: 0.6250
```

ConvNets

```
testLoss_TL, testAccuracy_TL = model_TL.evaluate(x = X_test_std, y = y_test)
# 100/1 [===] - 3s 26ms/sample - loss: 2.6403 - accuracy: 0.7200

print("Test-loss: %f, Test-accuracy: %f" % (testLoss_TL, testAccuracy_TL))
# Test-loss: 3.964784, Test-accuracy: 0.720000
```