Convolutional Neural Networks (CNN)

Dr. Ram Prasad K
VisionCog R&D

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

ConvNets

- DNN can learn only global features.
- They are not robust enough to capture large variations.
- To overcome this issue, we can increase the training set with several variations.
- DNN works reasonably well for small images only.

ConvNets

- In computer vision applications, we usually have to deal with complex variations and large images.

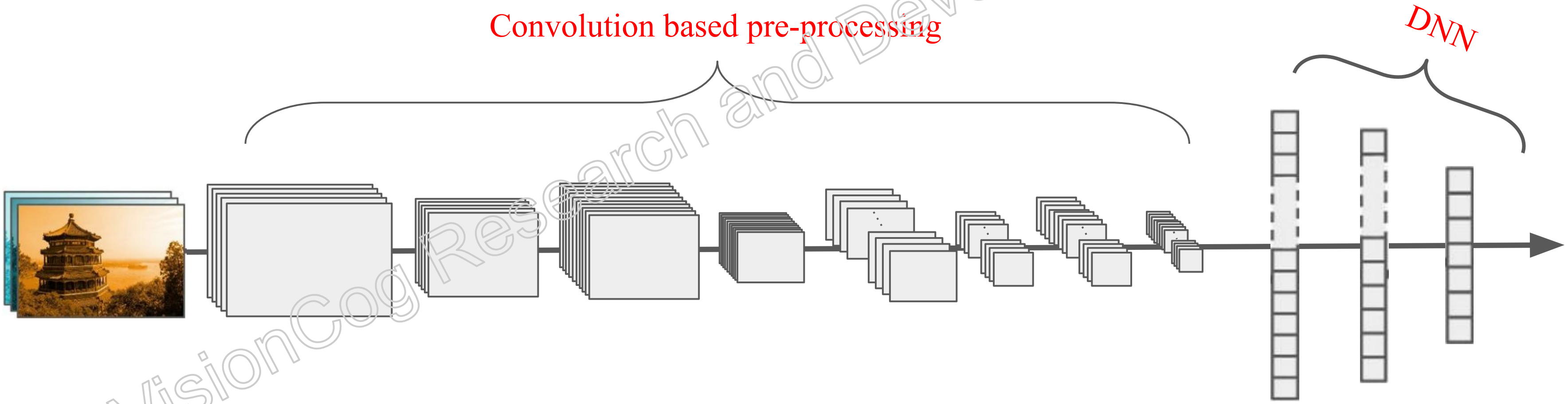
- Successful extraction of salient features is key for any computer vision task.

- A model which can learn local features will be robust enough in extracting salient features.

- Convolutional Neural Networks (CNN or ConvNets)
  - DNN modified to capture local features

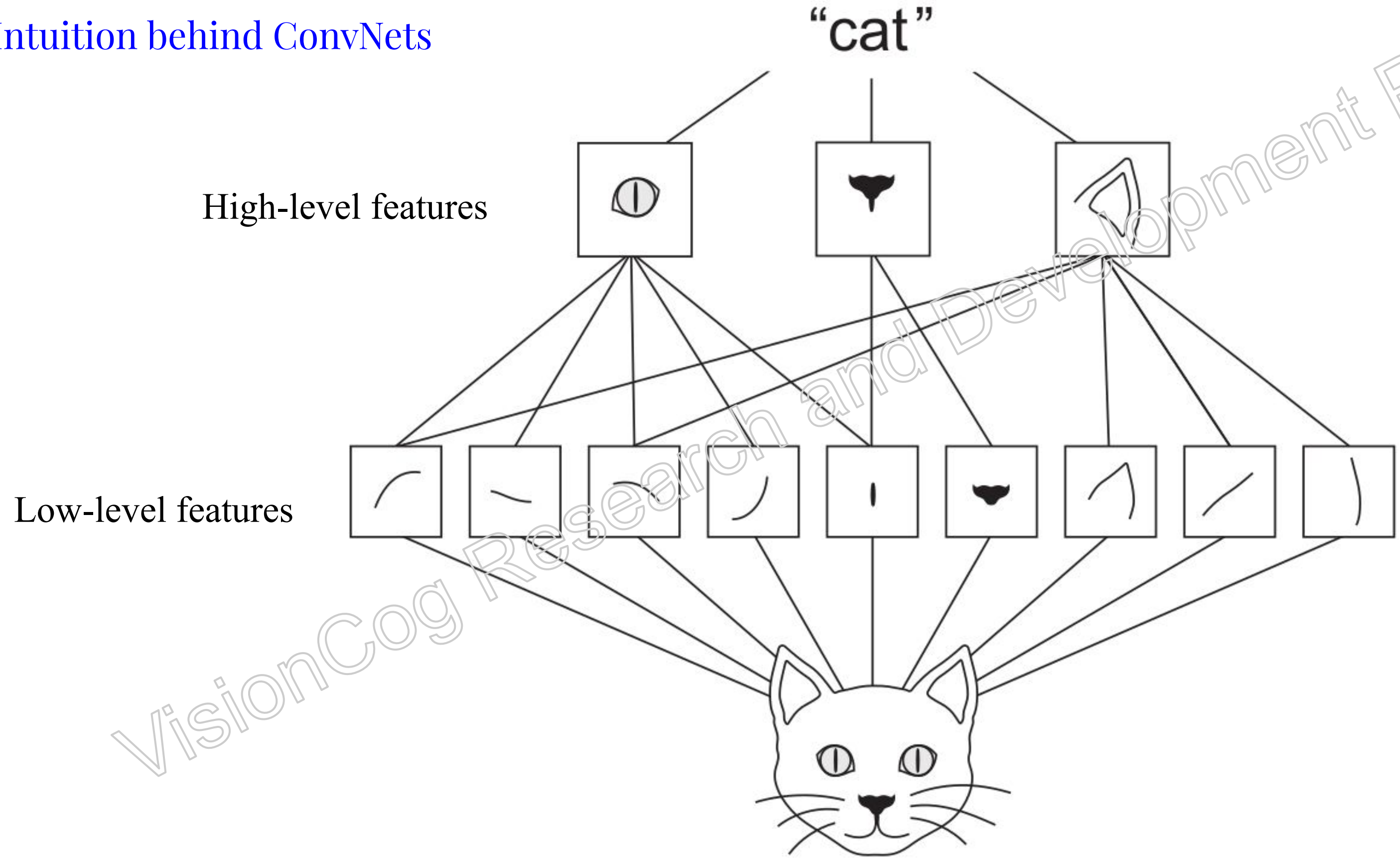
ConvNets

Intuition behind ConvNets



ConvNets

Intuition behind ConvNets



Building blocks of CNN

Convolutions

Mathematical operation which slides one function over the other and measures integral of pointwise multiplication (i.e., weighted sum of the inputs).

Strides

How quickly window slides.

Stride 2 means, window moves by 2 pixels at a time.

Pooling

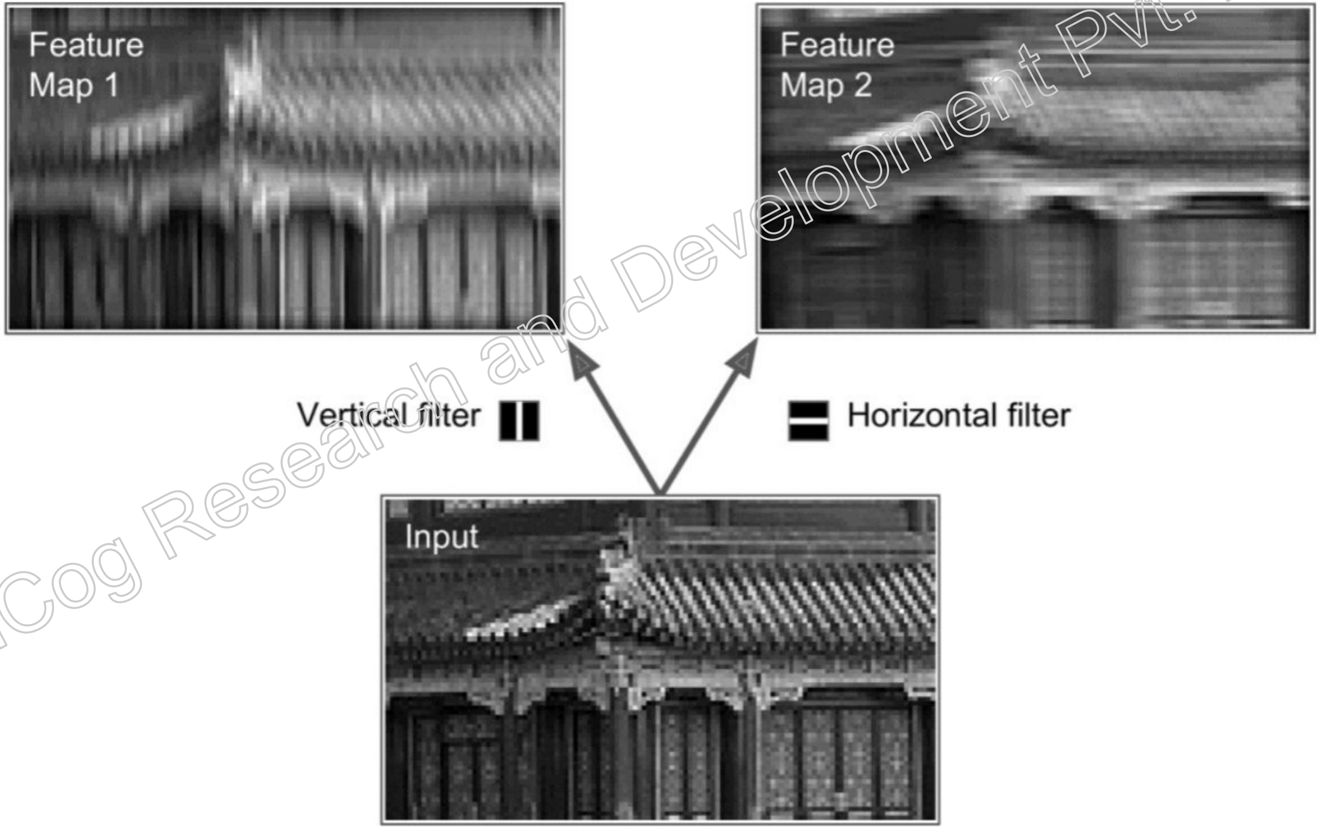
Downsampling feature maps.

DNN

Fully connected DNN for classification

ConvNets

Convolutions



ConvNets

1	0	0	0	0	0
0	1	1	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	0	0	1	0	1
0	0	1	0	0	1

convolution
stride=1

1	1	2	0
1	3	0	0
2	0	0	3
0	0	3	0

0	0	1
0	1	0
1	0	0

: kernel

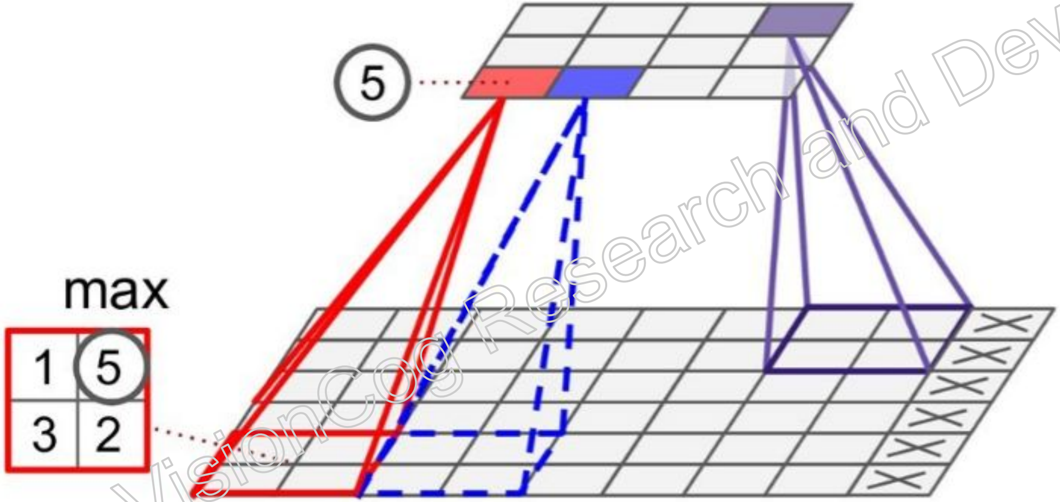
0	0	0	1	0	0
0	1	0	0	0	0
0	0	1	1	1	0
0	0	0	1	0	0
0	0	1	0	0	1
0	0	0	0	1	0

convolution
stride=1

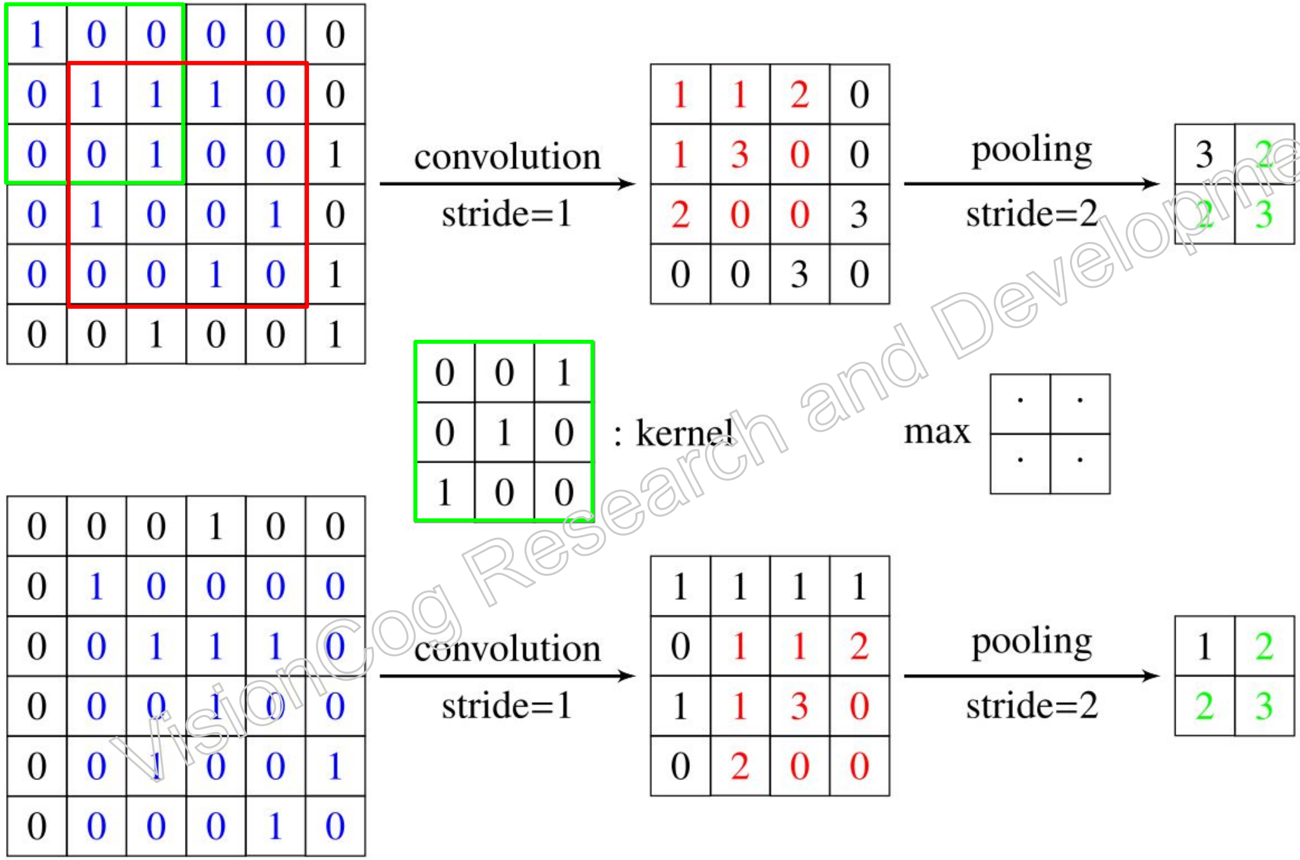
1	1	1	1
0	1	1	2
1	1	3	0
0	2	0	0

ConvNets

Pooling - MaxPooling

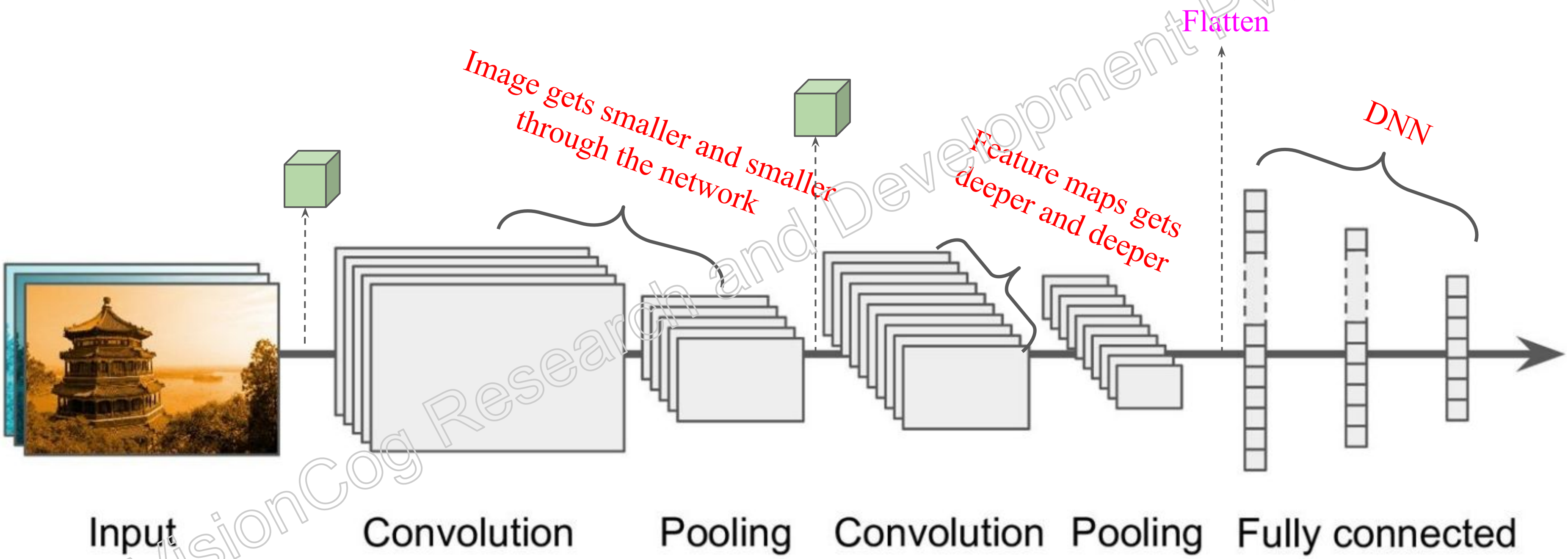


ConvNets


convolution
stride=1
pooling
stride=2
: kernel
max
convolution
stride=1
pooling
stride=2

ConvNets

CNN architecture



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
```

ConvNets

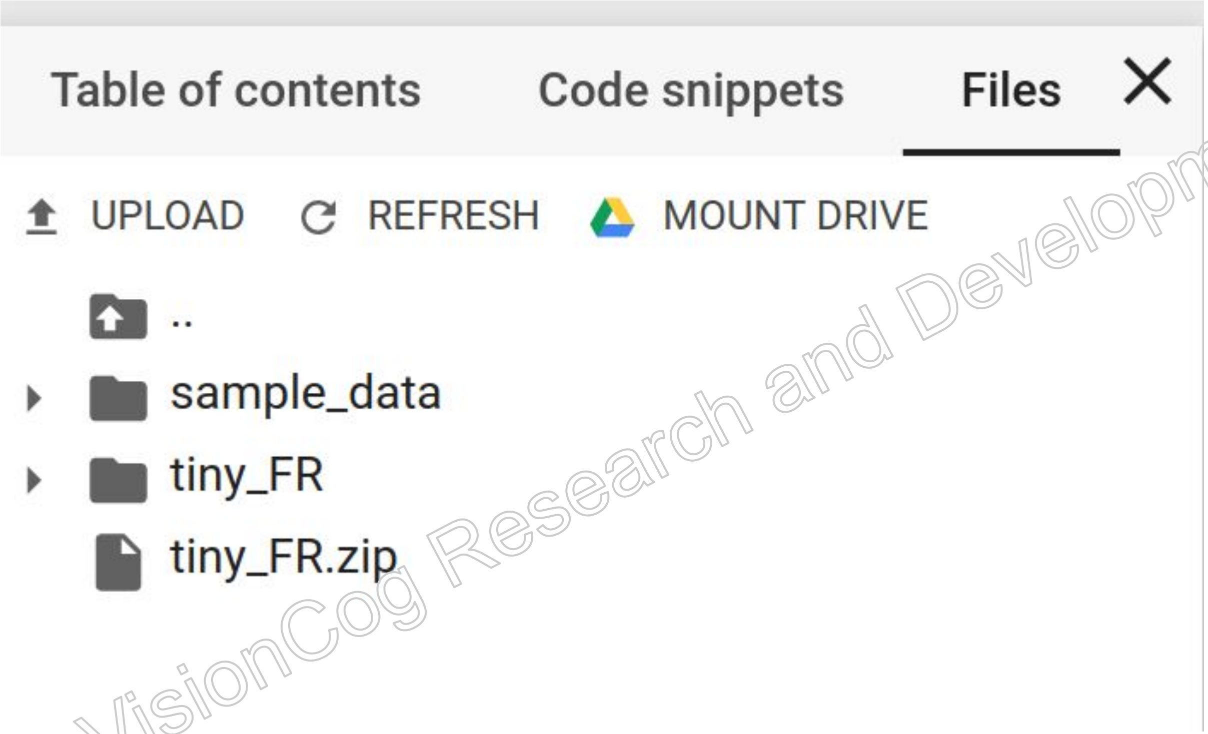


Table of contents
Code snippets
Files
UPLOAD
REFRESH
MOUNT DRIVE
..
sample_data
tiny_FR
tiny_FR.zip

```
# 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
```

```
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']
```

```
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)
```

```
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

```
# Create the network using Functional API method

input_ = keras.layers.Input(shape = X_train.shape[1:])

x = keras.layers.Conv2D(filters=32, kernel_size=5, padding='same', activation='relu')(input_)
x = keras.layers.MaxPool2D(pool_size=2)(x)
x = keras.layers.Conv2D(filters=64, kernel_size=3, padding='same', activation='relu')(x)
x = keras.layers.MaxPool2D(pool_size=2)(x)

x = keras.layers.Flatten()(x)
x = keras.layers.Dense(units=1000, activation='relu')(x)
x = keras.layers.Dense(units=100, activation='relu')(x)

output_ = keras.layers.Dense(units=5, activation='softmax')(x)

model_CNN = keras.models.Model(inputs=[input_], outputs=[output_])
```

ConvNets

```
model_CNN.summary()

Model: "model"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_1 (InputLayer)        [(None, 299, 299, 3)]     0
_________________________________________________________________
conv2d (Conv2D)             (None, 299, 299, 32)      2432
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 149, 149, 32)     0
_________________________________________________________________
conv2d_1 (Conv2D)           (None, 149, 149, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 64)       0
_________________________________________________________________
flatten (Flatten)           (None, 350464)            0
_________________________________________________________________
dense (Dense)               (None, 1000)              350465000
_________________________________________________________________
dense_1 (Dense)             (None, 100)               100100
_________________________________________________________________
dense_2 (Dense)             (None, 5)                 505
=================================================================
Total params: 350,586,533
Trainable params: 350,586,533
Non-trainable params: 0
_________________________________________________________________
```

ConvNets

```
model_CNN.compile(loss='sparse_categorical_crossentropy',
                  optimizer='adam', metrics=['accuracy'])

history_CNN = model_CNN.fit(x = X_train_std, y = y_train, epochs=25,
                            validation_split=0.1, batch_size=32)
```

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

Epoch 1/25
360/360 [===]-7s 18ms/sample - loss: 27.5400 - accuracy: 0.2361 - val_loss: 2.1541 - val_accuracy: 0.0750
Epoch 2/25
360/360 [===]-2s 6ms/sample - loss: 1.4595 - accuracy: 0.3083 - val_loss: 1.5361 - val_accuracy: 0.2250
Epoch 3/25
360/360 [===]-2s 6ms/sample - loss: 1.3308 - accuracy: 0.5194 - val_loss: 1.4175 - val_accuracy: 0.4250

...

Epoch 24/25
360/360 [===]-2s 6ms/sample - loss: 1.1731e-04 - accuracy: 1.0000 - val_loss: 3.4437 - val_accuracy: 0.3500
Epoch 25/25
360/360 [===]-2s 6ms/sample - loss: 1.0730e-04 - accuracy: 1.0000 - val_loss: 3.4645 - val_accuracy: 0.3500
```

ConvNets

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

Epoch 1/25
360/360 [===]-7s 18ms/sample - loss: 27.5400 - accuracy: 0.2361 - val_loss: 2.1541 - val_accuracy: 0.0750
Epoch 2/25
360/360 [===]-2s 6ms/sample - loss: 1.4595 - accuracy: 0.3083 - val_loss: 1.5361 - val_accuracy: 0.2250
Epoch 3/25
360/360 [===]-2s 6ms/sample - loss: 1.3308 - accuracy: 0.5194 - val_loss: 1.4175 - val_accuracy: 0.4250

...

Epoch 24/25
360/360 [===]-2s 6ms/sample - loss: 1.1731e-04 - accuracy: 1.0000 - val_loss: 3.4437 - val_accuracy: 0.3500
Epoch 25/25
360/360 [===]-2s 6ms/sample - loss: 1.0730e-04 - accuracy: 1.0000 - val_loss: 3.4645 - val_accuracy: 0.3500
```

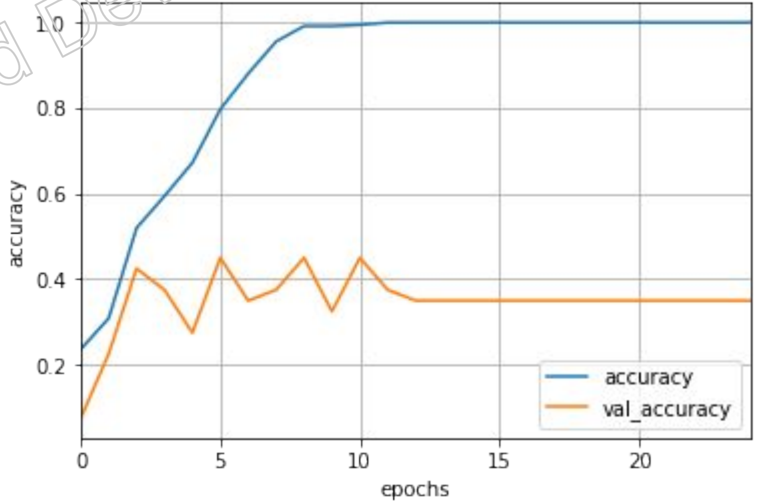
ConvNets

```
keys = ['accuracy', 'val_accuracy']
progress = {k:v for k,v in history_CNN.history.items() if k in keys}

import pandas as pd
pd.DataFrame(progress).plot()

plt.xlabel("epochs")
plt.ylabel("accuracy")

plt.grid(True)
plt.show()
```



ConvNets

```
test_loss, test_accuracy = model_CNN.evaluate(X_test_std, y_test)
# 100/1 [===] - 0s 3ms/sample - loss: 5.2441 - accuracy: 0.5500

print("Test-loss: %f, Test-accuracy: %f" % (test_loss, test_accuracy))
# Test-loss: 2.738503, Test-accuracy: 0.550000
```