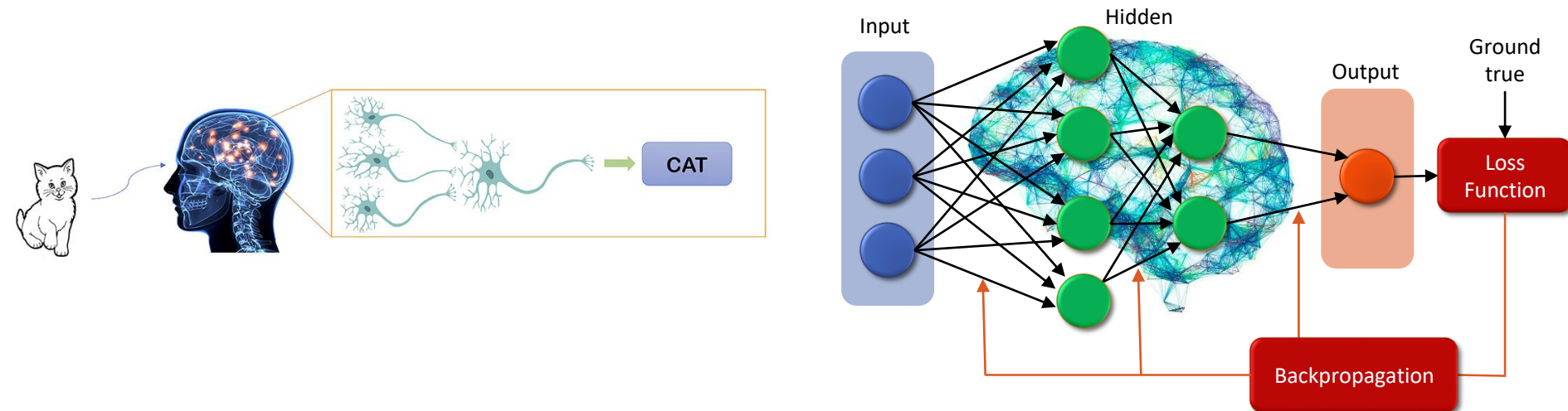


DEEP LEARNING

PRE-TRAINED MODEL VÀ VISUALIZATION



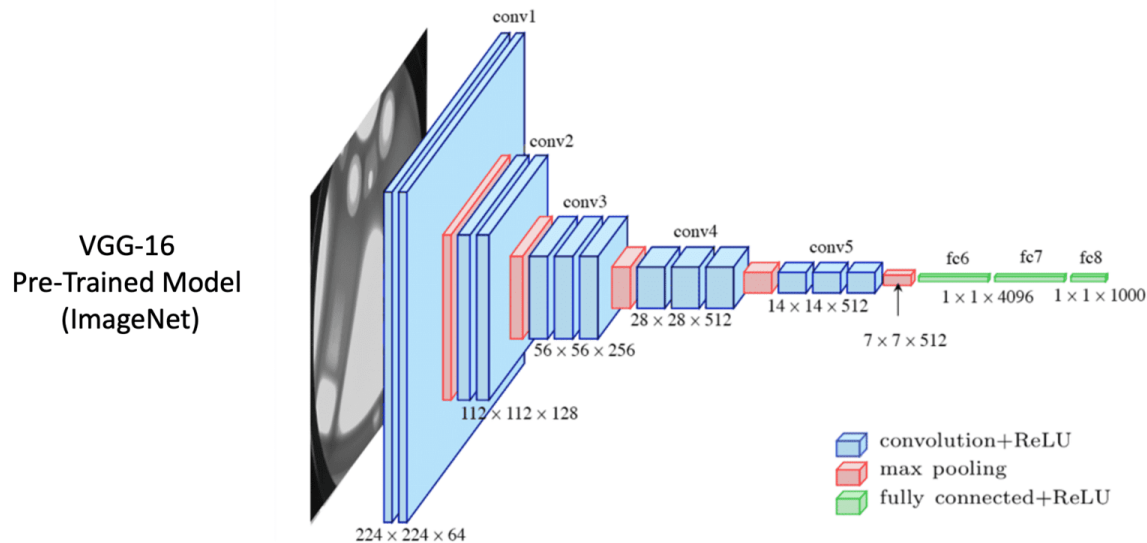
Tôn Quang Toại
Khoa Công nghệ thông tin
Trường đại học Ngoại ngữ - Tin học TP.HCM (HUFLIT)

Nội dung

- Pre-Trained models
- Prediction
- Feature Extraction
- Fine - Tuning
- Trực quan hóa Convolutional Neural Network

Pre-trained model

- **Pre-trained model:** pre-trained model là những model nổi tiếng đã được huấn luyện trước trên tập dữ liệu chuẩn lớn (ImageNet)



Pre-trained model

- Nhận xét
 - Có thể sử dụng các mô hình hiện đại (state-of-the-art) thay vì phát minh lại từ đầu
 - Nếu tập dữ liệu được dùng huấn luyện đủ lớn và đủ tổng quát thì các feature học được từ pre-trained model có thể dùng như mô hình tổng quát của thế giới thị giác.
 - Các đặc trưng đã học có thể dùng cho các bài toán thị giác máy tính khác nhau, ngay cả bài toán mới với các lớp hoàn toàn khác với bài toán ban đầu.

ImageNet

- Thông tin về ImageNet
 - 1.2 triệu ảnh
 - 1000 lớp
 - Có Bounding Box cho bài toán Object Localization
 - Các động vật
 - Các đối tượng hằng ngày
- Danh sách các lớp trong ImageNet

<https://image-net.org/challenges/LSVRC/2012/browse-synsets>

Keras Applications

- **Keras Applications:** Keras Applications là **host** chứa các models nổi tiếng cùng các weights đã được huấn luyện trước trên ImageNet

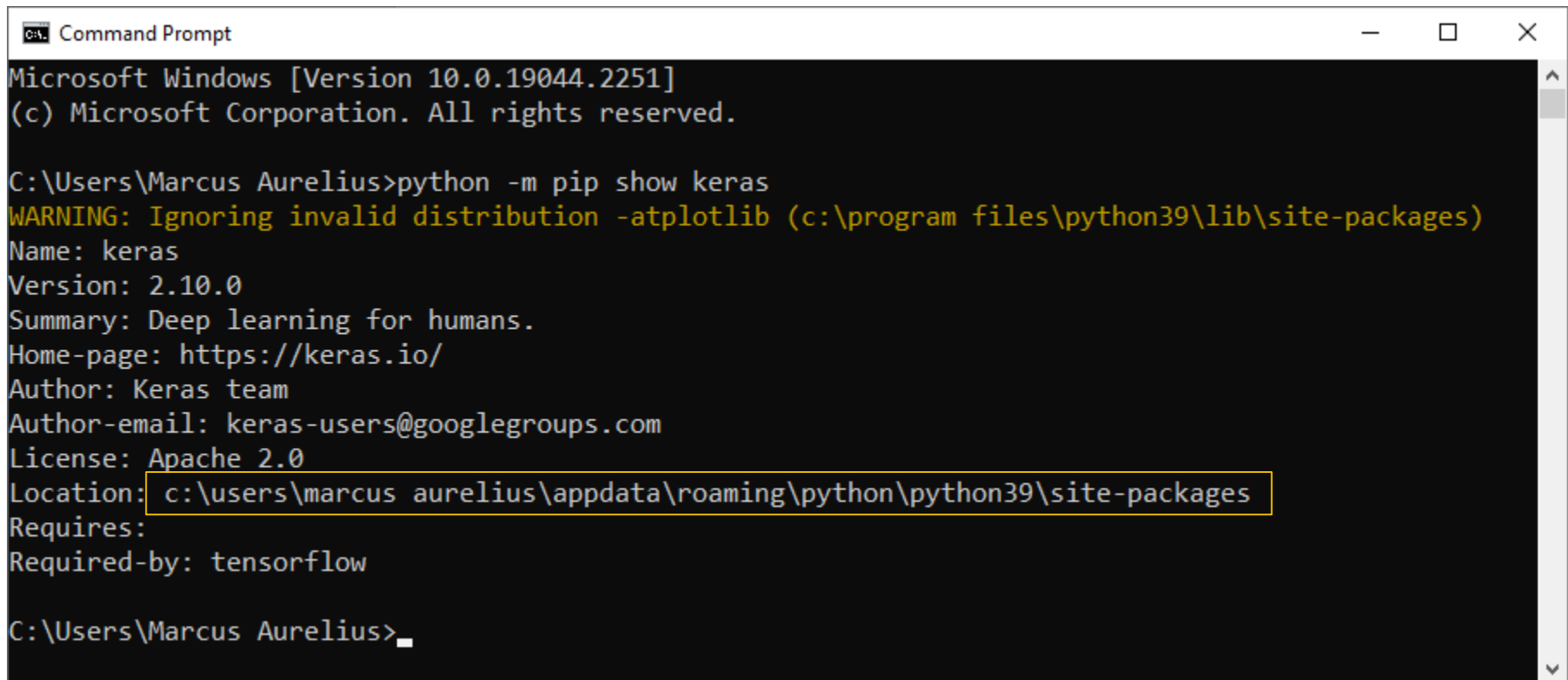
```
tf.keras.applications
```

- Chú ý: Khi tạo và load model thì các weights tự động download và lưu vào **~/.keras/models/**

Keras Applications

- Hiện thông tin cài đặt Keras

```
python -m pip show keras
```



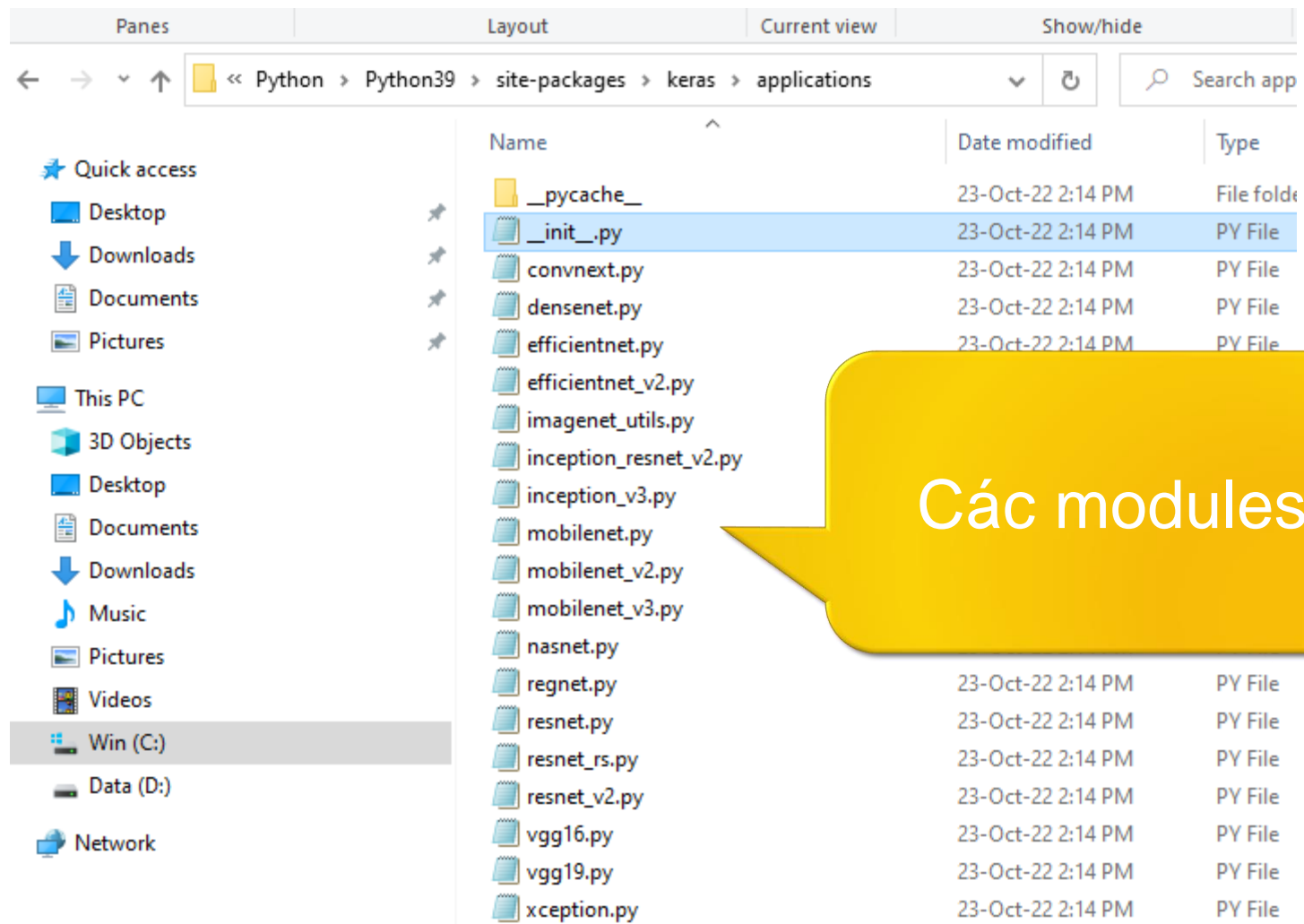
```
Command Prompt
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Marcus Aurelius>python -m pip show keras
WARNING: Ignoring invalid distribution -atplotlib (c:\program files\python39\lib\site-packages)
Name: keras
Version: 2.10.0
Summary: Deep learning for humans.
Home-page: https://keras.io/
Author: Keras team
Author-email: keras-users@googlegroups.com
License: Apache 2.0
Location: c:\users\marcus aurelius\appdata\roaming\python\python39\site-packages
Requires:
Required-by: tensorflow

C:\Users\Marcus Aurelius>
```

Cấu trúc Keras application

- Các modules (Các Model family)



Cấu trúc Keras application

- Các hàm (Các Models)

Các Modules

```
*__init__.py - Notepad
File Edit Format View Help
__init__.py
from keras.applications.convnext import ConvNeXtBase
from keras.applications.convnext import ConvNeXtLarge
from keras.applications.convnext import ConvNeXtSmall
from keras.applications.convnext import ConvNeXtTiny
from keras.applications.convnext import ConvNeXtXLarge
from keras.applications.densenet import DenseNet121
from keras.applications.densenet import DenseNet169
from keras.applications.densenet import DenseNet201
from keras.applications.efficientnet import EfficientNetB0
from keras.applications.efficientnet import EfficientNetB1
from keras.applications.efficientnet import EfficientNetB2
from keras.applications.efficientnet import EfficientNetB3
from keras.applications.efficientnet import EfficientNetB4
from keras.applications.efficientnet import EfficientNetB5
from keras.applications.efficientnet import EfficientNetB6
from keras.applications.efficientnet import EfficientNetB7
from keras.applications.efficientnet_v2 import EfficientNetV2B0
from keras.applications.efficientnet_v2 import EfficientNetV2B1
from keras.applications.efficientnet_v2 import EfficientNetV2B2
from keras.applications.efficientnet_v2 import EfficientNetV2B3
from keras.applications.efficientnet_v2 import EfficientNetV2L
from keras.applications.efficientnet_v2 import EfficientNetV2M
from keras.applications.efficientnet_v2 import EfficientNetV2S
```

Các Hàm

Cấu trúc Keras application

Các Modules

Các Hàm

```
*_init_.py - Notepad
File Edit Format View Help
from keras.applications.convnext import ConvNeXtBase
from keras.applications.convnext import ConvNeXtLarge
from keras.applications.convnext import ConvNeXtSmall
from keras.applications.convnext import ConvNeXtTiny
from keras.applications.convnext import ConvNeXtXLarge
from keras.applications.densenet import DenseNet121
from keras.applications.densenet import DenseNet169
from keras.applications.densenet import DenseNet201
from keras.applications.efficientnet import EfficientNetB0
from keras.applications.efficientnet import EfficientNetB1
from keras.applications.efficientnet import EfficientNetB2
from keras.applications.efficientnet import EfficientNetB3
from keras.applications.efficientnet import EfficientNetB4
from keras.applications.efficientnet import EfficientNetB5
from keras.applications.efficientnet import EfficientNetB6
from keras.applications.efficientnet import EfficientNetB7
from keras.applications.efficientnet_v2 import EfficientNetV2B0
from keras.applications.efficientnet_v2 import EfficientNetV2B1
from keras.applications.efficientnet_v2 import EfficientNetV2B2
from keras.applications.efficientnet_v2 import EfficientNetV2B3
from keras.applications.efficientnet_v2 import EfficientNetV2L
from keras.applications.efficientnet_v2 import EfficientNetV2M
from keras.applications.efficientnet_v2 import EfficientNetV2S
```

Ln 20, Col 62 60% Unix (LF) UTF-8

Cấu trúc Keras application

- Hiển thị nội dung của modules

```
import tensorflow as tf
from tensorflow.keras.preprocessing import image

all_modules = dir(tf.keras.applications)
print(all_modules)
```

Cấu trúc Keras application

```
['ConvNeXtBase', 'ConvNeXtLarge', 'ConvNeXtSmall', 'ConvNeXtTiny', 'ConvNeXtXLarge',  
'DenseNet121', 'DenseNet169', 'DenseNet201', 'EfficientNetB0', 'EfficientNetB1', 'EfficientNetB2',  
'EfficientNetB3', 'EfficientNetB4', 'EfficientNetB5', 'EfficientNetB6', 'EfficientNetB7',  
'EfficientNetV2B0', 'EfficientNetV2B1', 'EfficientNetV2B2', 'EfficientNetV2B3', 'EfficientNetV2L',  
'EfficientNetV2M', 'EfficientNetV2S', 'InceptionResNetV2', 'InceptionV3', 'MobileNet',  
'MobileNetV2', 'MobileNetV3Large', 'MobileNetV3Small', 'NASNetLarge', 'NASNetMobile',  
'RegNetX002', 'RegNetX004', 'RegNetX006', 'RegNetX008', 'RegNetX016', 'RegNetX032',  
'RegNetX040', 'RegNetX064', 'RegNetX080', 'RegNetX120', 'RegNetX160', 'RegNetX320',  
'RegNetY002', 'RegNetY004', 'RegNetY006', 'RegNetY008', 'RegNetY016', 'RegNetY032',  
'RegNetY040', 'RegNetY064', 'RegNetY080', 'RegNetY120', 'RegNetY160', 'RegNetY320',  
'ResNet101', 'ResNet101V2', 'ResNet152', 'ResNet152V2', 'ResNet50', 'ResNet50V2',  
'ResNetRS101', 'ResNetRS152', 'ResNetRS200', 'ResNetRS270', 'ResNetRS350',  
'ResNetRS420', 'ResNetRS50', 'VGG16', 'VGG19', 'Xception', '__builtins__', '__cached__',  
'__doc__', '__file__', '__loader__', '__name__', '__package__', '__path__', '__spec__', '_sys',  
'convnext', 'densenet', 'efficientnet', 'efficientnet_v2', 'imagenet_utils', 'inception_resnet_v2',  
'inception_v3', 'mobilenet', 'mobilenet_v2', 'mobilenet_v3', 'nasnet', 'regnet', 'resnet', 'resnet50',  
'resnet_rs', 'resnet_v2', 'vgg16', 'vgg19', 'xception']
```

Cấu trúc Keras application

- Thông tin các Modules và các Hàm

```
for module in all_modules:
    if module[0].islower():
        if module != 'imagenet_utils':
            print(f"Model Family: {module}")

        models = dir(getattr(tf.keras.applications, module))
        for model in models:
            if model[0].isupper():
                print(f"\t\t__ {model}")

print()
```

Sử dụng pre-trained model

- Có 3 cách sử dụng pre-trained models
 - Prediction: Dùng ngay cho bài toán phân lớp ảnh
 - Transfer learning
 - Feature extraction
 - Fine-tuning: Tinh chỉnh cho dataset khác

Các bước sử dụng pre-trained models

- Pre-trained models có thể được thực hiện theo các bước sau

Bước 1. Chọn pre-trained model và Xác định input size theo model

Bước 2. Load ảnh và Resize ảnh theo kích thước theo model

Bước 3. Sử dụng model để dự đoán

- Bước 3.1 Gọi hàm `preprocess_input()`

Tiền xử lý các ảnh input dùng hàm tiền xử lý chuyên dụng của model,

- Bước 3.2 Gọi hàm `predict()`

Để sinh ra dự đoán

- Bước 3.2 Gọi hàm `decode_predictions()`

Giải mã các dự đoán bằng hàm hậu xử lý chuyên dụng của model,

Các bước sử dụng pre-trained models

- **Bước 1. Chọn model và Xác định input size**

```
IMG_SHAPE = (224, 224, 3)
```

```
# model_family = "densenet"
# model_name = "DenseNet121"           # (224, 224, 3)
# model_name = "DenseNet169"           # (224, 224, 3)
# model_name = "DenseNet201"           # (224, 224, 3)
```

```
# model_family = "EfficientNet"
# model_name = "EfficientNetB0"         # (224, 224, 3)
# model_name = "EfficientNetB1"         # (240, 240, 3)
# model_name = "EfficientNetB2"         # (260, 260, 3)
# model_name = "EfficientNetB3"         # (300, 300, 3)
# model_name = "EfficientNetB4"         # (380, 380, 3)
# model_name = "EfficientNetB5"         # (456, 456, 3)
# model_name = "EfficientNetB6"         # (528, 528, 3)
# model_name = "EfficientNetB7"         # (600, 600, 3)
```

```
# model_family = "inception_resnet_v2"
# model_name = "InceptionResNetV2"     # (299, 299, 3)
```

```
# model_family = "inception_v3"
# model_name = "InceptionV3"           # (299, 299, 3)
```

```
# model_family = "mobilenet"
# model_name = "MobileNet"             # (224, 224, 3)
```

```
# model_family = "mobilenet_v2"
# model_name = "MobileNetV2"           # (224, 224, 3)
```

```
# model_family = "mobilenet_v3"
# model_name = "MobileNetV3"           # (224, 224, 3)
```


Các bước sử dụng pre-trained models

```
model_family = "mobilenet_v3"
model_name    = "MobileNetV3Small"      # (224, 224, 3)
# model_name   = "MobileNetV3Large"     # (224, 224, 3)

# model_family = "nasnet"
# model_name    = "NASNetMobile"        # (224, 224, 3)

# model_family = "resnet"
# model_name    = "ResNet101"           # (224, 224, 3)
# model_name    = "ResNet152"          # (224, 224, 3)
# model_name    = "ResNet50"           # (224, 224, 3)

# model_family = "resnet50"
# model_name    = "ResNet50"            # (224, 224, 3)
```

```
# model_family = "resnet_v2"
# model_name    = "ResNet101V2"        # (224, 224, 3)
# model_name    = "ResNet152V2"        # (224, 224, 3)
# model_name    = "ResNet50V2"         # (224, 224, 3)

# model_family = "vgg16"
# model_name    = "VGG16"              # (224, 224, 3)

# model_family = "vgg19"
# model_name    = "VGG19"              # (224, 224, 3)

# model_family = "xception"
# model_name    = "Xception"           # (299, 299, 3)
```

```
def load_model(input_shape, model_name):
    model = getattr(tf.keras.applications, model_name)(input_shape=input_shape, weights="imagenet")

    return model
```

Các bước sử dụng pre-trained models

- **Bước 2. Load ảnh và Resize ảnh**

```
def load_image(image_path, image_shape):  
    image = tf.io.read_file(IMG_PATH)  
    x = tf.image.decode_image(image, channels=img_shape[2])  
    x = tf.image.resize(x, (img_shape[0], img_shape[1]))  
    x = tf.expand_dims(x, axis=0)  
  
    return x
```

Các bước sử dụng pre-trained models

- **Bước 3.** Sử dụng model

- Bước 3.1 Tiền xử lý (ảnh)

```
x=preprocess_input(input, data_format=None)
```

- Bước 3.2 Sinh dự đoán

```
predictions = model.predict(x)
```

- Bước 3.3 Hậu xử lý (dự đoán)

```
predictions = decode_predictions(preds, top=5)
```

Các bước sử dụng pre-trained models

```
def generate_predictions(input, model_family, model, top_k=3):
    preprocess = getattr(tf.keras.applications, model_family).preprocess_input
    postprocess = getattr(tf.keras.applications, model_family).decode_predictions

    # 3.1 Tiền xử lý input image
    x = preprocess(input)

    # 3.2 Sinh predictions
    preds = model.predict(x)

    # 3.3 Hậu xử lý
    post_preds = postprocess(preds, top=top_k)[0]

    print('Predicted')
    for i in post_preds:
        print(f"Class Description: {i[1]:<30} Score: {i[2]}")

    return preds
```

Các bước sử dụng pre-trained models

- Thực hiện Inference

1. `load_image()`
2. `load_model()`
3. `generate_predictions()`

```
# Load an image.  
input = load_image(image_path=IMG_PATH, image_shape=IMG_SHAPE)  
  
# Load the model.  
pretrained_model = load_model(input_shape=IMG_SHAPE, model_name=model_name)  
  
# Generate predictions.  
predictions = generate_predictions(input=input,  
                                   model_family=model_family,  
                                   model=pretrained_model,  
                                   top_k=5,  
                                   )
```

Các bước sử dụng pre-trained models

- Cách ngắn gọn

```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np

# Load model
model = ResNet50(weights='imagenet')

# Tiền xử lý ảnh
img_path = '... .jpg'
img = image.load_img(img_path, target_size=(224, 224))

x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

# Dự đoán
preds = model.predict(x)

# Hậu xử lý
predictions = decode_predictions(preds, top=3)
print('Predicted:', predictions[0])
```

SO SÁNH GIỮA CÁC MODELS

Thông tin model đã pre-trained

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6

Thông tin model đã pre-trained

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0

Thông tin model đã pre-trained

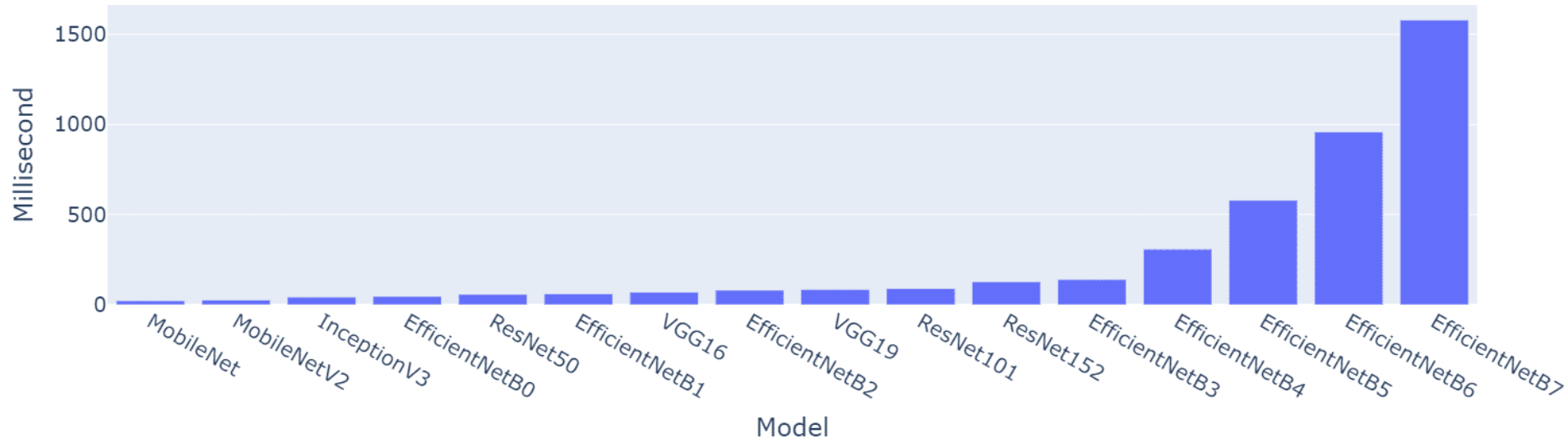
Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1
EfficientNetB5	118	83.6%	96.7%	30.6M	312	579.2	25.3
EfficientNetB6	166	84.0%	96.8%	43.3M	360	958.1	40.4
EfficientNetB7	256	84.3%	97.0%	66.7M	438	1578.9	61.6
EfficientNetV2B0	29	78.7%	94.3%	7.2M	-	-	-
EfficientNetV2B1	34	79.8%	95.0%	8.2M	-	-	-

Thông tin model đã pre-trained

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
EfficientNetV2B2	42	80.5%	95.1%	10.2M	-	-	-
EfficientNetV2B3	59	82.0%	95.8%	14.5M	-	-	-
EfficientNetV2S	88	83.9%	96.7%	21.6M	-	-	-
EfficientNetV2M	220	85.3%	97.4%	54.4M	-	-	-
EfficientNetV2L	479	85.7%	97.5%	119.0M	-	-	-
ConvNeXtTiny	109.42	81.3%	-	28.6M	-	-	-
ConvNeXtSmall	192.29	82.3%	-	50.2M	-	-	-
ConvNeXtBase	338.58	85.3%	-	88.5M	-	-	-
ConvNeXtLarge	755.07	86.3%	-	197.7M	-	-	-
ConvNeXtXLarge	1310	86.7%	-	350.1M	-	-	-

So sánh thời gian dự đoán

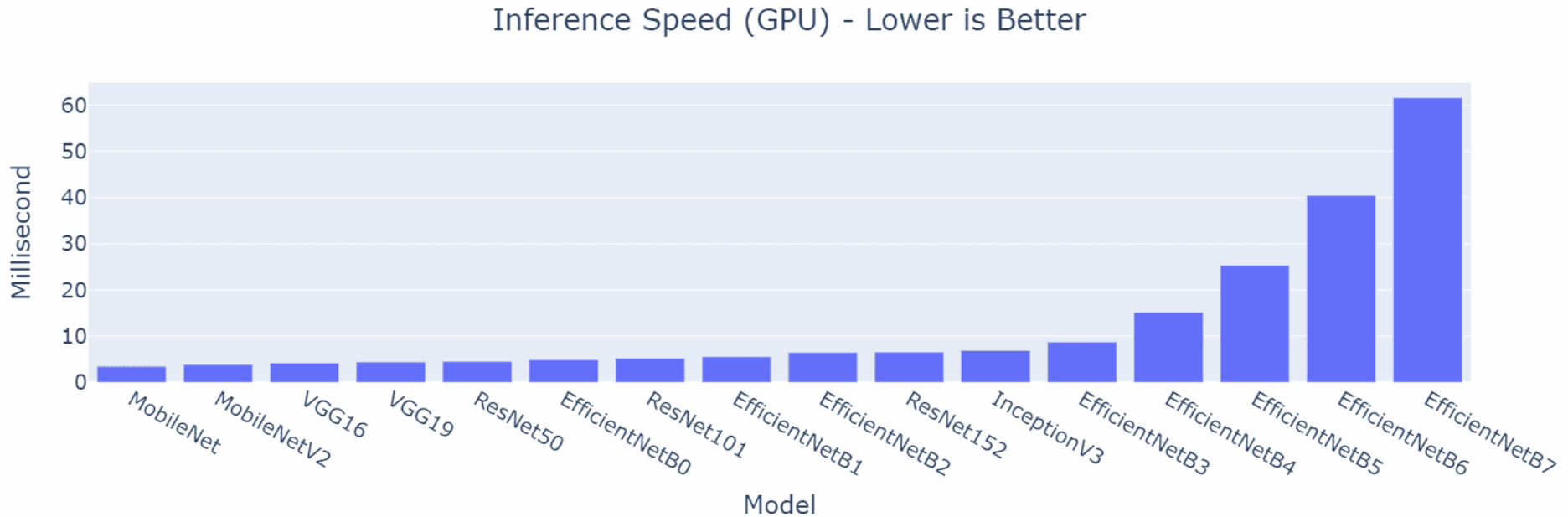
Inference Speed (CPU) - Lower is Better



- **Nhận xét**

- EfficientNet-B7 chính xác nhất nhưng thời gian dự đoán của mô hình này lớn
- EfficientNet-B0 có độ chính xác thấp hơn một chút so với các mô hình, nhưng thời gian chạy nhanh hơn khoảng 10 lần so với EfficientNet-B7
- MobileNetV2 có độ chính xác tương tự VGG-16 và VGG-19 nhưng có thời gian suy luận thấp hơn một chút

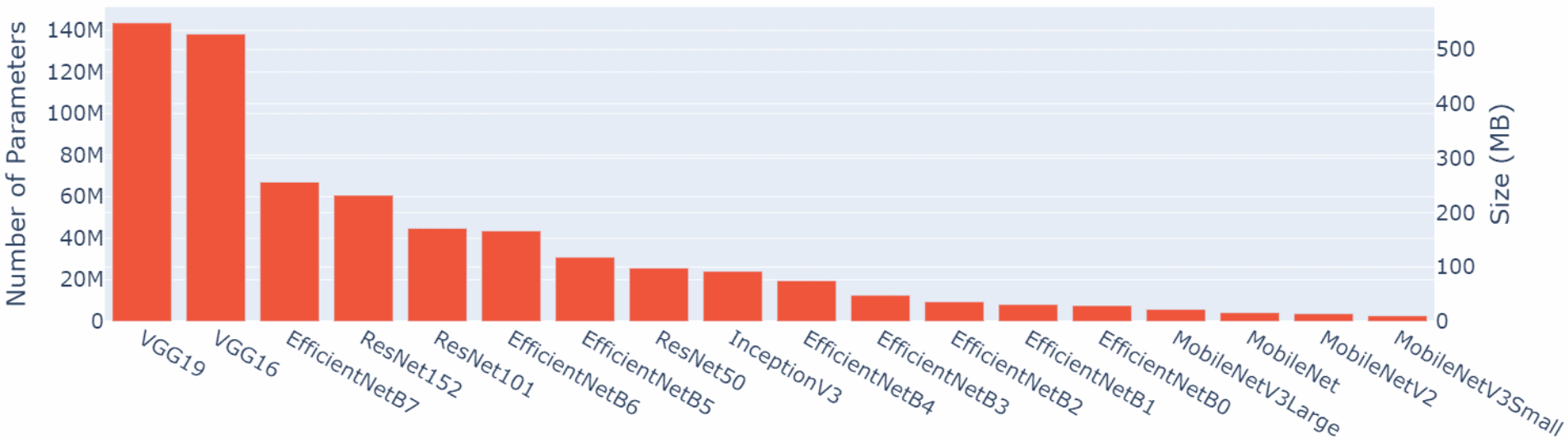
So sánh thời gian dự đoán



- Nhận xét: Sử dụng GPU tốc độ dự đoán giảm đáng kể

So sánh kích thước và tham số

Parameter and Model Size Comparison



- Nhận xét
 - MobileNetV3Small có kích thước nhỏ nhất (10 MB),
 - VGG-19 có kích thước lớn nhất (549MB).

TRANSFER LEARNING

Transfer Learning

- **Transfer learning:** Chuyển kiến thức của một mô hình để thực hiện một nhiệm vụ mới.



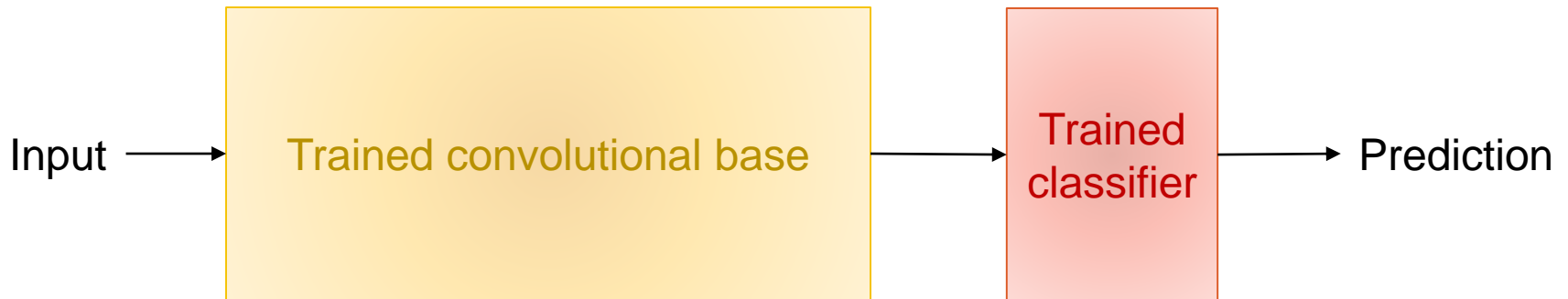
- **Động cơ dùng Transfer learning**
 - Tốn nhiều dữ liệu, thời gian, tài nguyên để training mạng từ đầu
 - Huấn luyện trên ImageNet có thể mất hàng tuần.
 - Trừ khi bạn có 256 GPU (tốn 1 giờ)
 - Cách rẻ, nhanh để huấn luyện mạng

Transfer Learning

- Kỹ thuật transfer learning
 - Feature extraction
 - Fine-tuning: Tinh chỉnh cho dataset khác

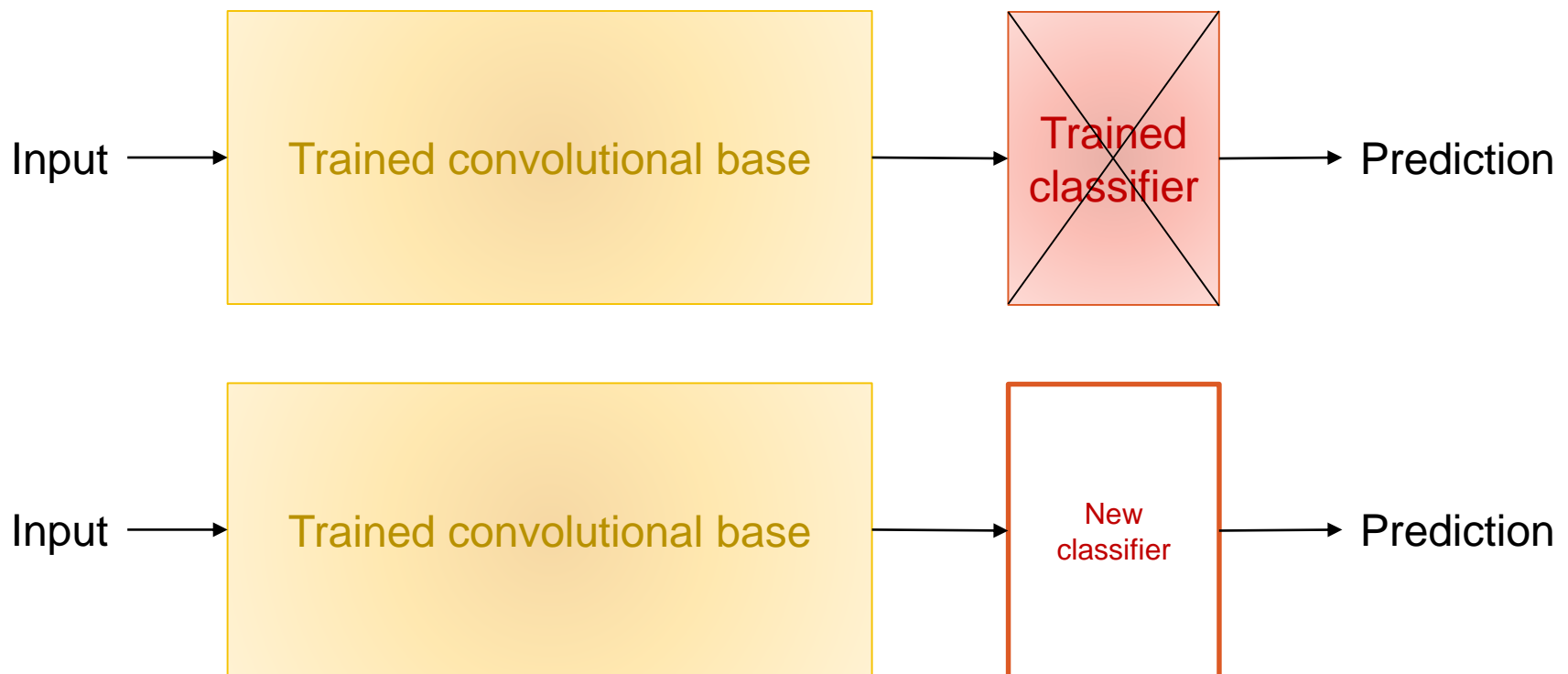
Feature extraction

- Các thành phần trong CNN
 - Convolution base:
 - Classifier:



Feature extraction

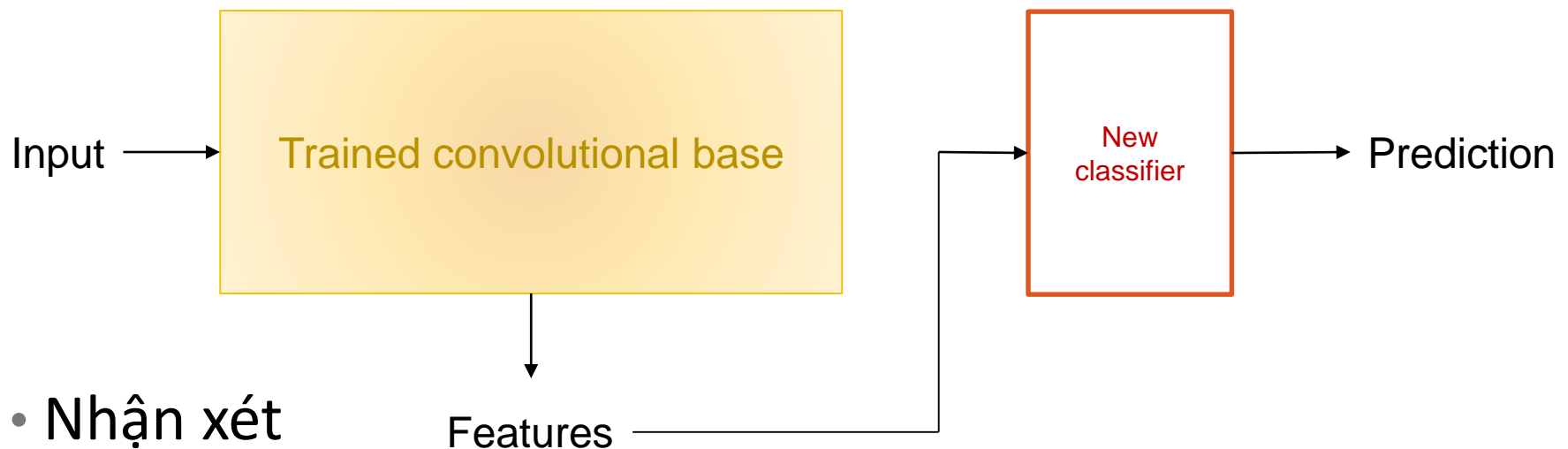
- Các bước thực hiện
 - Bước 1. Bỏ tầng top (tầng phân lớp (classifier))
 - Bước 2. Thêm tầng classifier mới



Feature extraction

- Cách feature extraction 1

- Tách mạng làm 2 phần: convolutional base, classifier mới
- Đưa dữ liệu train vào convolution base, được các features
- Dùng các features này để train classifier



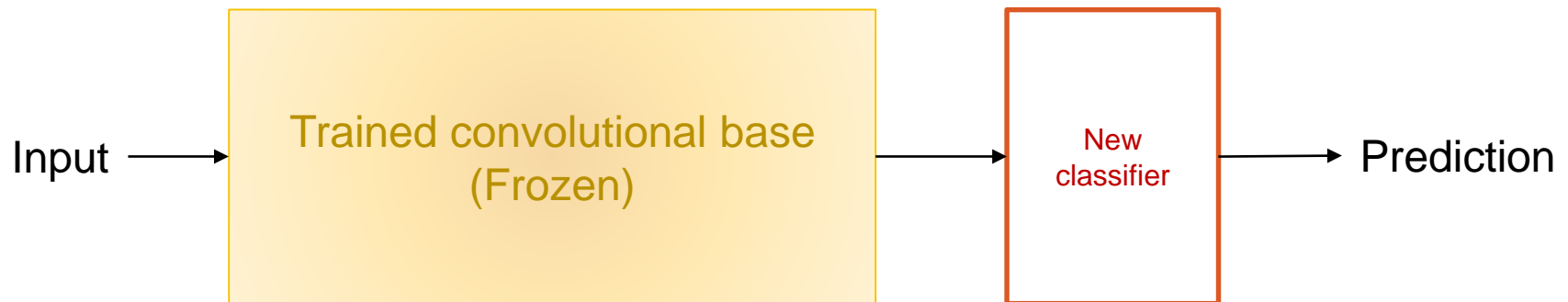
- Nhận xét

- Nhanh
- Không dùng được data augmentation

Feature extraction

- Cách feature extraction 2

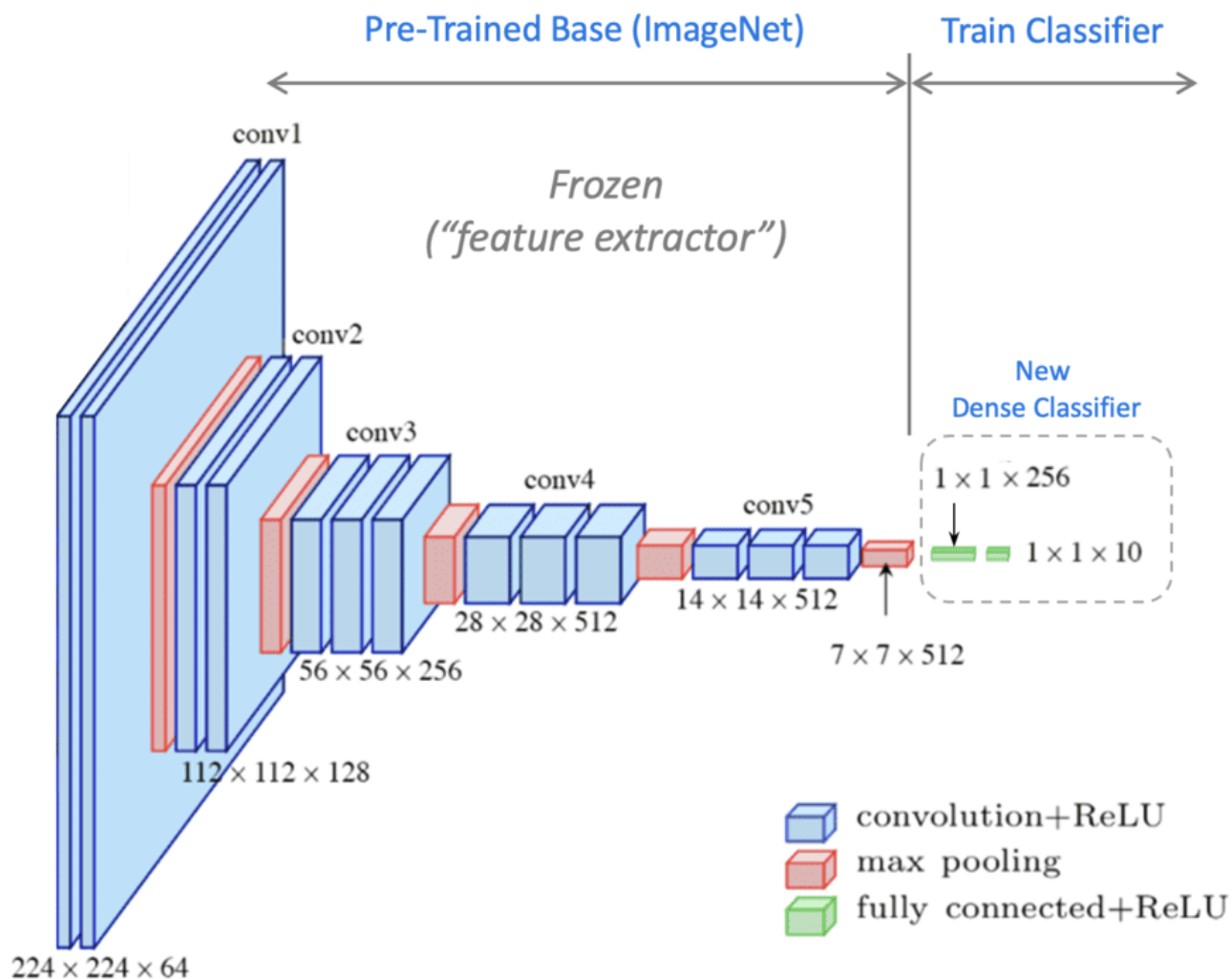
- Giữ mạng làm một khối gồm 2 phần
 - Convolutional base
 - Classifier mới
- Đóng băng convolution base
- Đưa dữ liệu vào huấn luyện phần classifier



- Nhận xét

- Chậm
- Dùng được data augmentation

Feature extraction



VGG16

```
tf.keras.applications.vgg16.VGG16(include_top=True,  
                                   weights='imagenet',  
                                   input_tensor=None,  
                                   input_shape=None,  
                                   pooling=None,  
                                   classes=1000,  
                                   classifier_activation='softmax',  
                                   )
```

- **include_top**: Có thêm tầng fully-connected vào đỉnh của network không
- **weights**
 - None (random initialization)
 - 'imagenet' (pre-training trên ImageNet)
 - Đường dẫn đến weights file
- **classes**: chỉ dùng khi include_top = True
- **classifier_activation**: chỉ dùng khi include_top = True

VGG Convolutional Base

```
input_shape = (...)  
  
print('Loading model ...')  
vgg16_conv_base = tf.keras.applications.vgg16.VGG16(input_shape=input_shape,  
                                                    include_top=False,  
                                                    weights='imagenet'  
                                                    )  
  
vgg16_conv_base.trainable = False  
  
print(vgg16_conv_base.summary())
```


Thêm tầng classification

```
inputs = tf.keras.Input(shape=input_shape)

x = tf.keras.applications.vgg16.preprocess_input(inputs)

x = vgg16_conv_base(x)

x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)

outputs = layers.Dense(NUM_CLASSES, activation='softmax')(x)

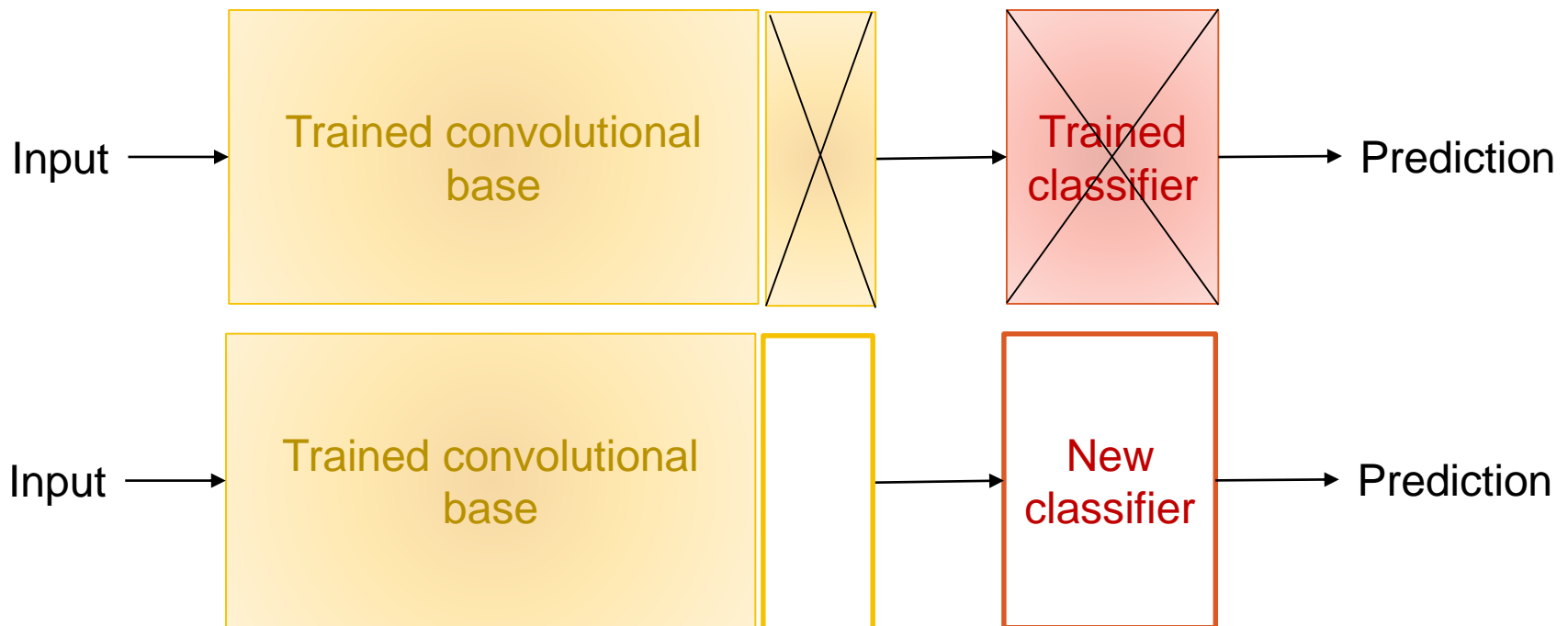
# The final model.
vgg16_model = keras.Model(inputs, outputs)

print(vgg16_model.summary())
```

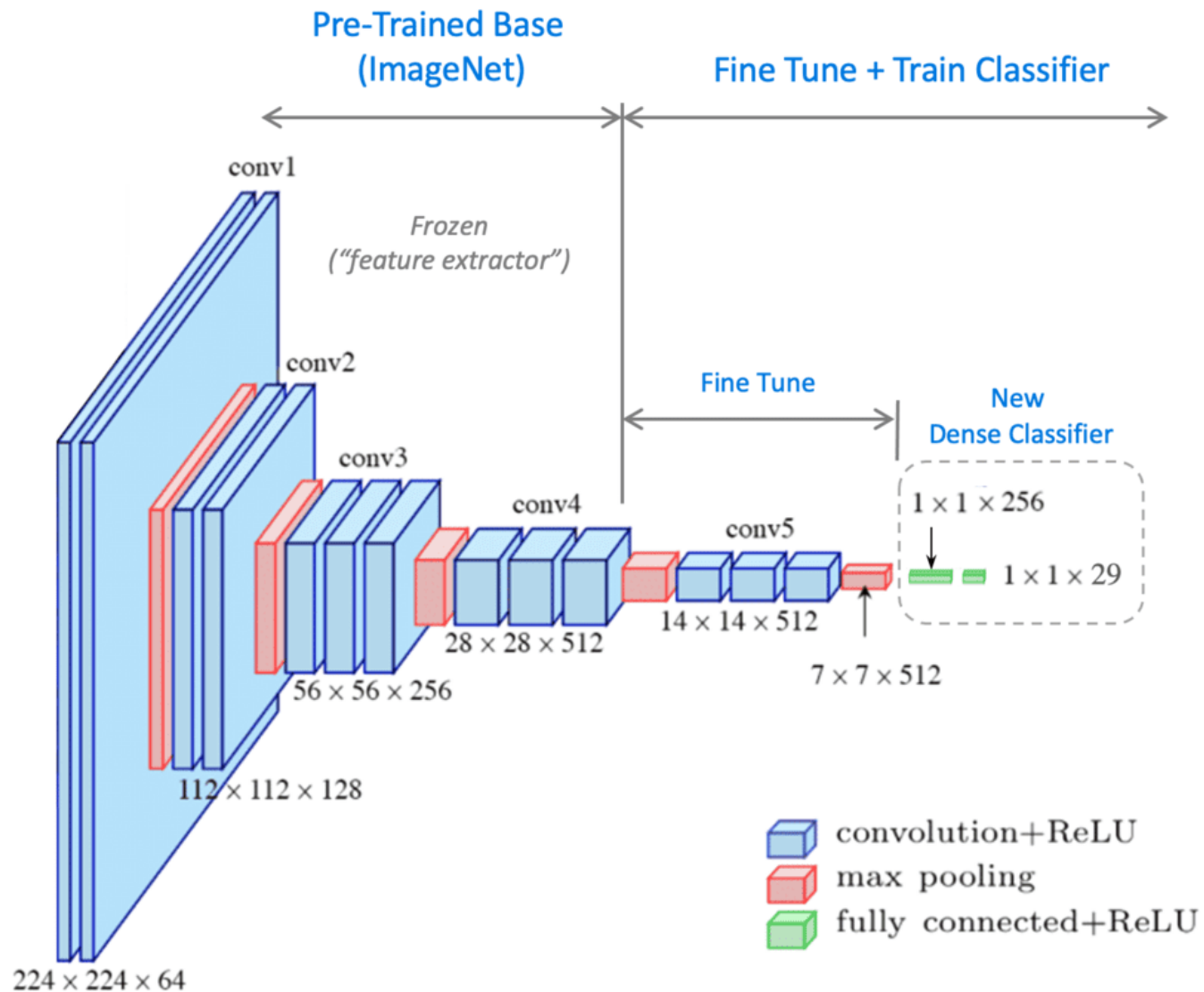
Fine-Tuning

- Các bước thực hiện

- Bước 1. Bỏ tầng phân lớp và một số tầng convolution ở gần tầng phân lớp
- Bước 2. Sử dụng các tầng convolutions còn lại
- Bước 3. Thêm một số tầng convolution mới và tầng phân lớp



Fine-Tuning



VGG Convolutional Base

```
input_shape = (...)  
  
print('Loading model ...')  
vgg16_conv_base = tf.keras.applications.vgg16.VGG16(input_shape=input_shape,  
                                                    include_top=False,  
                                                    weights='imagenet'  
                                                    )  
  
vgg16_conv_base.trainable = True  
  
print(vgg16_conv_base.summary())
```

Unfreeze 4 tầng cuối để Fine-Tuning

```
# Số layers muốn fine tune
num_layers_fine_tune = 4
num_layers = len(vgg16_conv_base.layers)

# Freeze Các tầng phía trước.
for model_layer in vgg16_conv_base.layers[:num_layers - num_layers_fine_tune]:
    print(f"FREEZING LAYER: {model_layer}")
    model_layer.trainable = False

print(vgg16_conv_base.summary())
```

Thêm tầng classification

```
inputs = tf.keras.Input(shape=input_shape)

x = tf.keras.applications.vgg16.preprocess_input(inputs)

x = vgg16_conv_base(x)

x = layers.Flatten()(x)
x = layers.Dense(256, activation='relu')(x)

outputs = layers.Dense(NUM_CLASSES, activation='softmax')(x)

# The final model.
vgg16_model = keras.Model(inputs, outputs)

print(vgg16_model.summary())
```

DIỄN GIẢI NHỮNG ĐIỀU CNN ĐÃ HỌC

Diễn giải Convolution đã học

- Bài toán “diễn giải” mô hình deep learning
 - Deep learning đã học gì, chú ý điều gì
 - Human đã học gì



Bổ sung cho nhau

- Deep learning có phải là “black box”?

Đúng	Sai
Đối với những model khác	Đối với CNN

- Từ năm 2013, một loạt các kỹ thuật đã được phát triển

Diễn giải Convolution đã học

- 3 kỹ thuật trực quan hóa thông dụng
 - Trực quan hóa các kết quả của tầng trung gian (intermediate convnet outputs)
 - Trực quan hóa các filters
 - Trực quan heatmaps của class activation

KỸ THUẬT

GRAD-CAM CLASS ACTIVATION

Grad-CAM class activation

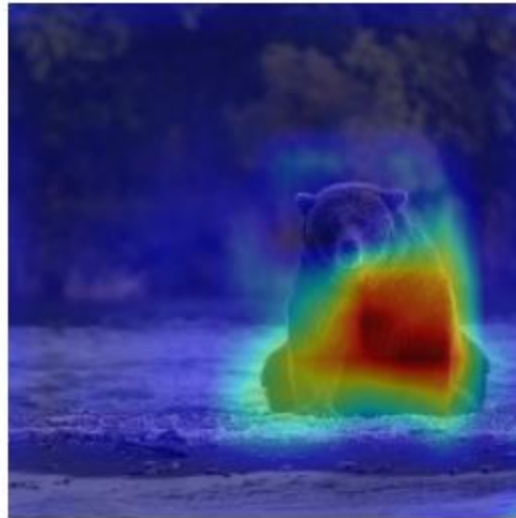
- **CAM (Class Activation Map):** CAM là vùng quan trọng của ảnh dùng để dự đoán lớp
- **Grad-CAM (Gradient-weighted Class Activation Mapping):**
 - Grad-CAM tính đạo hàm của lớp được chỉ định,
 - Sau đó chuyển đạo hàm vào tầng convolution cuối cùng để tạo ra một bản đồ vị trí làm nổi bật các vùng quan trọng trong ảnh đã dùng để dự đoán lớp được chỉ định

Grad-CAM class activation

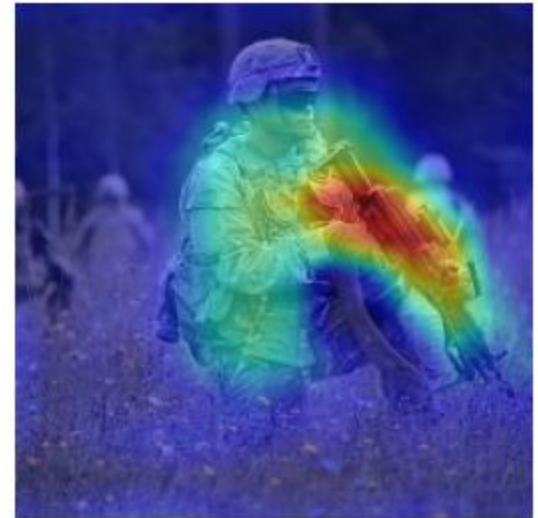
Goldfish



Bear



Assault rifle



Grad-CAM class activation

- Cài đặt tf-explain

```
pip install tf-explain
```

- Load model và Image

```
import tensorflow as tf
from tensorflow.keras.applications import vgg16
from tensorflow.keras.preprocessing import image
import tf_explain

IMAGE_PATH = "cam.jpg"

# Load pretrained model hay model của chúng ta
model = vgg16.VGG16(weights="imagenet", include_top=True)

# Load a sample image
img = image.load_img(IMAGE_PATH, target_size=(224, 224))
img = image.img_to_array(img)
data = ([img], None)
```

Grad-CAM class activation

- GradCAM

```
# Bắt đầu explainer
explainer = tf_explain.core.GradCAM()

grid = explainer.explain(data, model, class_index=281)
# 281 index của cat trong ImageNet

explainer.save(grid, ".", "grad_cam.png")
```



