

## Déployer un modèle d'apprentissage profond avec Flask en tant qu'application Web

**Objectif :** Développer une application web personnalisée pour déployer un modèle apprentissage profond (DL) de reconnaissance d'images animales en toute simplicité

### Préliminaire:

Installez les modules suivants :

- Werkzeug
- Flask
- numpy
- Keras
- gevent
- pillow
- h5py
- tensorflow

### Description de Project :

Voici la structure de projet de déploiement :

Ce PC > Disque local (C:) > Utilisateurs > Utilisateur > TP-1-webApp >				
	Nom	Modifié le	Type	Taille
	__pycache__	2020-04-11 23:37	Dossier de fichiers	
	models	2020-04-11 23:30	Dossier de fichiers	
	static	2020-04-11 04:16	Dossier de fichiers	
	templates	2020-04-11 04:16	Dossier de fichiers	
	app.py	2020-04-11 09:04	Fichier PY	3 Ko
	requirements.txt	2019-12-18 02:48	Document texte	1 Ko
flask	util.py	2020-04-11 08:45	Fichier PY	1 Ko

## 1. Répertoire templates :

Ce PC > Disque local (C:) > Utilisateurs > Utilisateur > TP-1-webApp > templates				
	Nom	Modifié le	Type	Taille
	base.html	2020-04-11 16:51	Chrome HTML Do...	1 Ko
	index.html	2020-04-11 16:35	Chrome HTML Do...	1 Ko

Contient les fichiers html

Fichier **index.html** : page d'accueil. Vous pouvez modifier au besoin :

```
{% extends "base.html" %} {% block content %}

<div class="main">
  <div class="title">
    <h3>Classificateur d'images</h3>
  </div>

  <div class="panel">
    <input id="file-upload" class="hidden" type="file" accept="image/x-png,image/gif,image/jpeg" />
    <label for="file-upload" id="file-drag" class="upload-box">
      <div id="upload-caption">Déposez l'image ici ou cliquez pour sélectionner</div>
      <img id="image-preview" class="hidden" />
    </label>
  </div>
  <div style="margin-bottom: 2rem;">
    <input type="button" value="Soumettre" class="button" onclick="submitImage();" />
    <input type="button" value="Reinitialiser" class="button" onclick="clearImage();" />
  </div>

  <div id="image-box">
    <img id="image-display" />
    <div id="pred-result" class="hidden"></div>
    <svg id="loader" class="hidden" viewBox="0 0 32 32" width="32" height="32">
      <circle id="spinner" cx="16" cy="16" r="14" fill="none"></circle>
    </svg>
  </div>
</div>

{% endblock %}
```

## 2. Répertoire **static**, contient deux fichiers feuilles de styles(.css) et le fichier javascript de l'application

Ce PC > Disque local (C:) > Utilisateurs > Utilisateur > TP-1-webApp > static				
	Nom	Modifié le	Type	Taille
	main.css	2019-12-18 02:48	Document de feui...	3 Ko
	main.js	2020-04-11 08:43	Fichier de JavaScript	4 Ko

3. Un fichier python **util.py** qui contient deux méthodes :
  - **base64\_to\_pil** pour convertir les données d'image base64 en image PIL
  - **np\_to\_base64** pour convertir une image numpy (RGB) en chaîne base64 qui vont être appelée dans **app.py**, comme les montrent les captures d'écrans ci-dessous :

» Ce PC » Disque local (C:) » Utilisateurs » Utilisateur » TP-1-webApp

	Nom	Modifié le	Type	Taille
	__pycache__	2020-04-11 23:37	Dossier de fichiers	
	models	2020-04-11 23:30	Dossier de fichiers	
	static	2020-04-11 04:16	Dossier de fichiers	
	templates	2020-04-11 04:16	Dossier de fichiers	
	app.py	2020-04-11 09:04	Fichier PY	3 Ko
	requirements.txt	2019-12-18 02:48	Document texte	1 Ko
flask	<b>util.py</b>	2020-04-11 08:45	Fichier PY	1 Ko

Et

```

"""Utilities
"""
import re
import base64

import numpy as np

from PIL import Image
from io import BytesIO

def base64_to_pil(img_base64):
    """
    Convertir les données d'image base64 en image PIL
    """
    image_data = re.sub('^data:image/.+;base64,', '', img_base64)
    pil_image = Image.open(BytesIO(base64.b64decode(image_data)))
    return pil_image

def np_to_base64(img_np):
    """
    Convertir une image numpy (RGB) en chaîne base64
    """
    img = Image.fromarray(img_np.astype('uint8'), 'RGB')
    buffered = BytesIO()
    img.save(buffered, format="PNG")
    return u"data:image/png;base64," + base64.b64encode(buffered.getvalue()).decode("ascii")
  
```

4. repertoire **models** dans lequel vous pouvez mettre n'importe quel modèle formé de **TensorFlow/Keras /PyTorch**, présentement il est vide parce qu'on va utiliser un modèle pré-formé(MobileNetV2) importé a partir **keras.applications.mobilenet\_v2**.

Ce PC > Disque local (C:) > Utilisateurs > Utilisateur > TP-1-webApp >

Nom	Modifié le	Type	Taille
__pycache__	2020-04-11 23:37	Dossier de fichiers	
<b>models</b>	2020-04-11 23:30	Dossier de fichiers	
static	2020-04-11 04:16	Dossier de fichiers	
templates	2020-04-11 04:16	Dossier de fichiers	
app.py	2020-04-11 09:04	Fichier PY	3 Ko
requirements.txt	2019-12-18 02:48	Document texte	1 Ko
util.py	2020-04-11 08:45	Fichier PY	1 Ko

5. Fichier python **app.py**, dans lequel on va :
- Déclarez une application flask
  - Servir l'application, comme le montre la capture ci-dessous :

Ce PC > Disque local (C:) > Utilisateurs > Utilisateur > TP-1-webApp

Nom	Modifié le	Type	Taille
__pycache__	2020-04-11 23:37	Dossier de fichiers	
models	2020-04-11 23:30	Dossier de fichiers	
static	2020-04-11 04:16	Dossier de fichiers	
templates	2020-04-11 04:16	Dossier de fichiers	
<b>app.py</b>	2020-04-11 09:04	Fichier PY	3 Ko
requirements.txt	2019-12-18 02:48	Document texte	1 Ko
util.py	2020-04-11 08:45	Fichier PY	1 Ko

Exemple de contenu de fichier **app.py** :

```

import os
import sys

# Flask
from flask import Flask, redirect, url_for, request, render_template, Response, jsonify, redirect
from werkzeug.utils import secure_filename
from event.pywsgi import WSGIServer

# TensorFlow et tf.keras
import tensorflow as tf
from tensorflow import keras

from tensorflow.keras.applications.imagenet_utils import preprocess_input, decode_predictions
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image

# Quelques utilitaires
import numpy as np
from util import base64_to_pil

# Déclarez une application flask
app = Flask(__name__)

# Vous pouvez utiliser un modèle pré-formé de Keras
# Check https://keras.io/applications/
from keras.applications.mobilenet_v2 import MobileNetV2
model = MobileNetV2(weights='imagenet')

print('Modèle chargé. Vérifiez http://127.0.0.1:5000/')

# Modèle enregistré avec Keras model.save ()
MODEL_PATH = 'models/your_model.h5'

# Chargez votre propre modèle formé
# model = load_model(MODEL_PATH)
# model._make_predict_function() # Nécessaire
# print('Modèle chargé. Commencez à servir ...')

```

### Démarche de déploiement:

- Installer les modules ci-dessus (les modules dans le fichier requirements.txt)

Ce PC > Disque local (C:) > Utilisateurs > Utilisateur > TP-1-webApp

Nom	Modifié le	Type	Taille
__pycache__	2020-04-11 23:37	Dossier de fichiers	
models	2020-04-11 23:30	Dossier de fichiers	
static	2020-04-11 04:16	Dossier de fichiers	
templates	2020-04-11 04:16	Dossier de fichiers	
app.py	2020-04-11 09:04	Fichier PY	3 Ko
requirements.txt	2019-12-18 02:48	Document texte	1 Ko
util.py	2020-04-11 08:45	Fichier PY	1 Ko

- Exécutez le script
- Accédez à <http://localhost:5000>

Réalisation :

1. Ouvrir un terminal anaconda
2. Installer les modules nécessaires
3. Taper **cd TP-1-webApp**

```
Anaconda Prompt (Anaconda3)

(base) C:\Users\Utilisateur>cd TP-1-webApp
```

4. Tapez **dir** pour vérifier le contenu de répertoire :

mor amami > TP-1-webApp >

Nom	Modifié le	Type	Taille
models	2020-04-11 23:30	Dossier de fichiers	
static	2020-04-11 04:16	Dossier de fichiers	
templates	2020-04-11 04:16	Dossier de fichiers	
app.py	2020-04-11 09:04	Fichier PY	3 Ko
requirements.txt	2019-12-18 02:48	Document texte	1 Ko
util.py	2020-04-11 08:45	Fichier PY	1 Ko

## Déploiement

5. Exécutez l'application

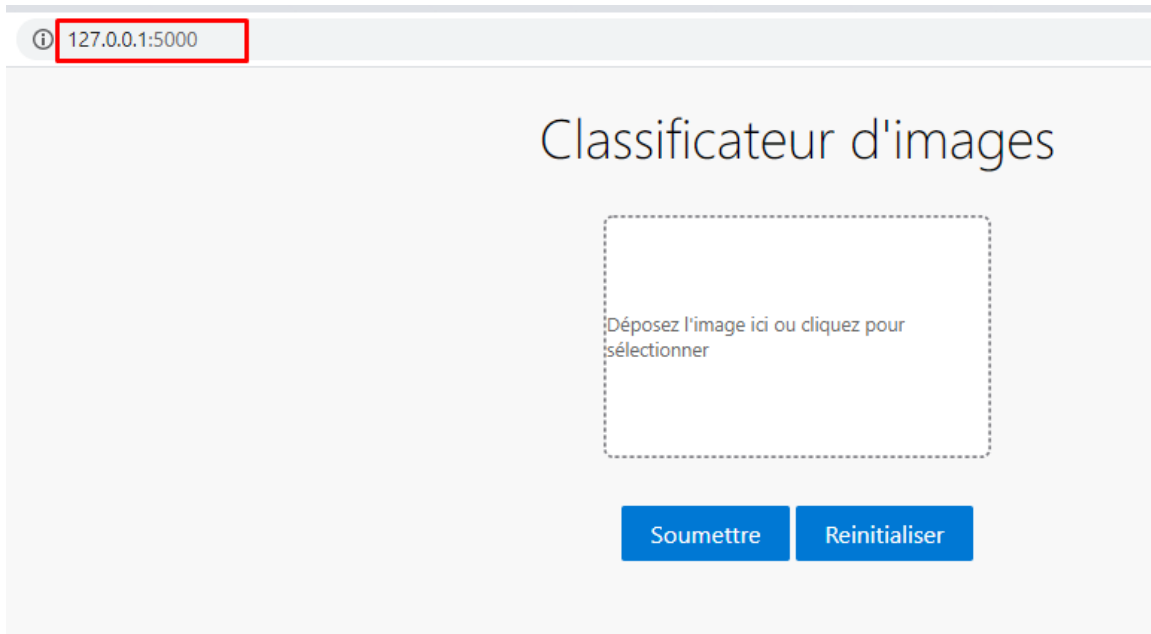
Tapez la commande suivante **python app.py**, comme le montre la capture d'écran ci-dessous :

```
(base) C:\Users\Utilisateur\TP-1-webApp>python app.py
Using TensorFlow backend.
2020-04-11 23:37:01.578531: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
Modèle chargé. Vérifiez http://127.0.0.1:5000/
```



Puis, Ouvrez dans un navigateur :

<http://localhost:5000>











Comme le montre la capture d'écran ci-dessous





Maintenant, pour faire des prédictions. Vous pouvez déposer une image d'un animal à partir de dossier : **images de TP-1**

Nom	Modifié le	Type	Taille
 <b>images</b>	2020-04-12 00:23	Dossier de fichiers	
 TP-1-webApp	2020-04-12 00:25	Dossier de fichiers	

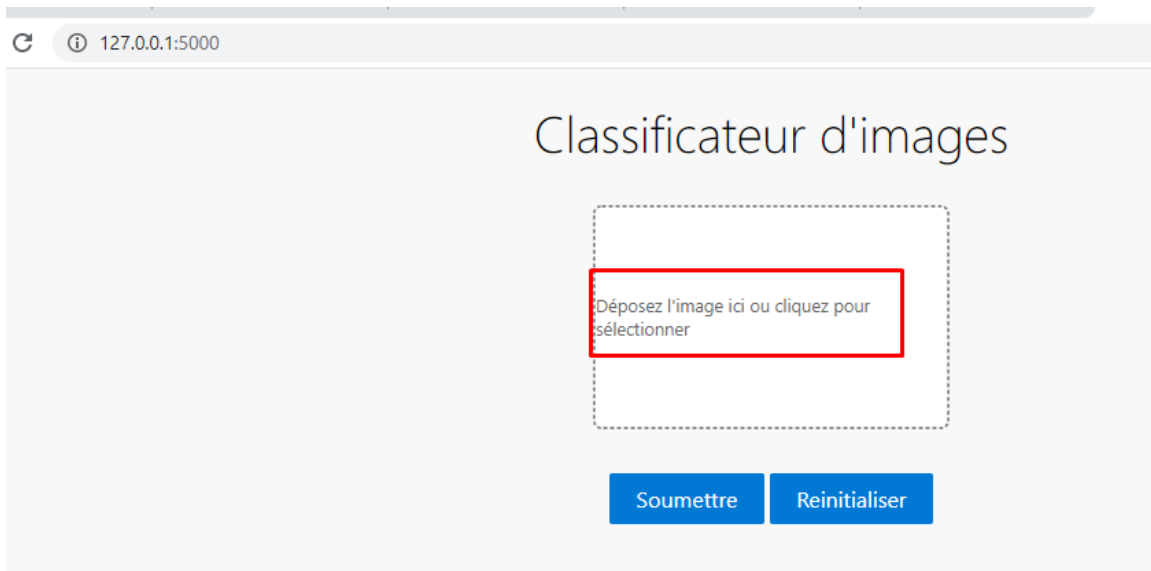











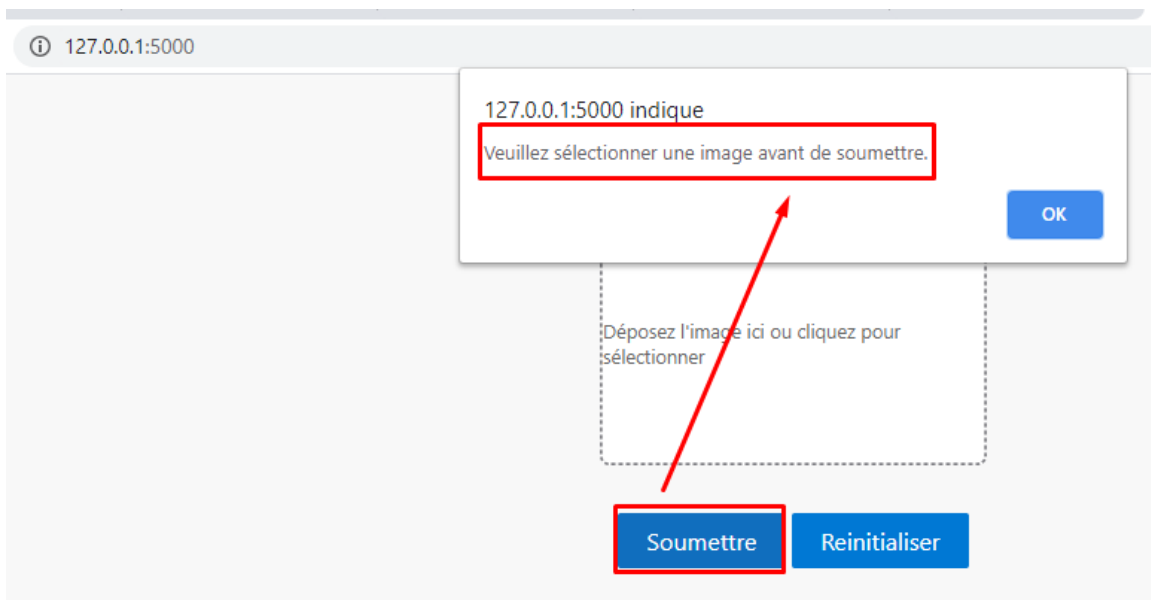
\_109100470\_ed01  
bd7f-0c64-4f6b-9  
d29-f43261119ad  
e.jpg  
 0.jpg  
 1.jpg  
 13.jpg  
 14.jpg  
 chat.jpg  
 horse-galloping-i  
n-grass-68889976  
9-587673275f9b5  
84db3a44cdf.jpg  
 images (1).jpg  
 images.jpg  
 téléchargement  
(1).jpg  
 téléchargement  
(2).jpg  
 téléchargement.j  
pg

Ou bien télécharger vos propres images.

Voici, quelques exemples de prédictions :

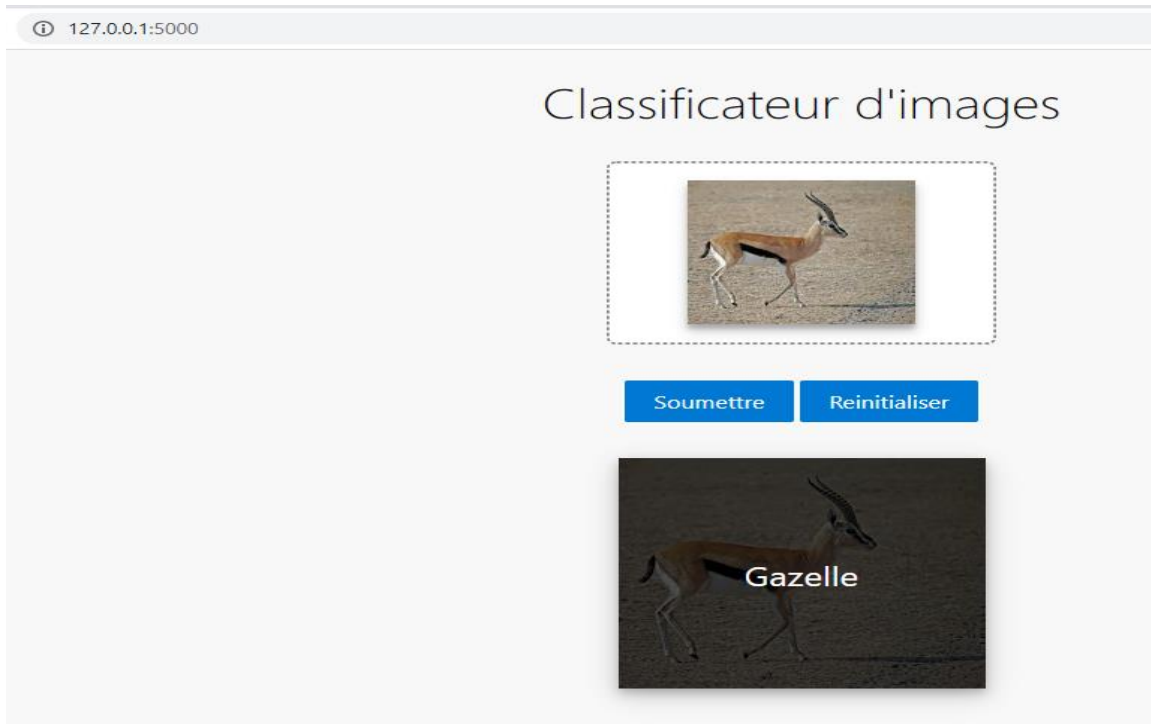


**Remarque :** si vous cliquez sur le bouton soumettre avant de choisir l'image, vous recevez le message suivant :



Résultat obtenu :





Comme vous remarquez ci-dessous, on appelle la méthode **GET** (request) lors de sélection de l'image et **POST** pour la prédiction(réponse)

```
127.0.0.1 - - [2020-04-11 23:38:23] "GET / HTTP/1.1" 200 1470 0.029985
127.0.0.1 - - [2020-04-11 23:39:03] "POST /predict HTTP/1.1" 200 151 3.172785
```

**Remarque 2 :** comment utilisez vos propres modèles :

Il est également facile de personnaliser et d'inclure vos modèles dans cette application.

**Utilisez votre propre modèle**

Après avoir **entraîné** et **enregistré** votre propre modèle (dans un fichier sous l'extension .h5 en utilisant la méthode `model.save()`), placez le (.h5) sous le répertoire model, comme le montre la capture d'écran ci-dessous :

PC > Disque local (C:) > Utilisateurs > Utilisateur > TP-1-webApp >				
Nom	Modifié le	Type	Taille	
__pycache__	2020-04-11 23:37	Dossier de fichiers		
<b>models</b>	2020-04-11 23:30	Dossier de fichiers		
static	2020-04-11 04:16	Dossier de fichiers		
templates	2020-04-11 04:16	Dossier de fichiers		
app.py	2020-04-11 09:04	Fichier PY	3 Ko	
requirements.txt	2019-12-18 02:48	Document texte	1 Ko	
util.py	2020-04-11 08:45	Fichier PY	1 Ko	

Ce PC > Disque local (C:) > Utilisateurs > Utilisateur > TP-1-webApp >				
Nom	Modifié le	Type	Taille	
__pycache__	2020-04-11 23:37	Dossier de fichiers		
models	2020-04-11 23:30	Dossier de fichiers		
static	2020-04-11 04:16	Dossier de fichiers		
templates	2020-04-11 04:16	Dossier de fichiers		
<b>app.py</b>	2020-04-11 09:04	Fichier PY	3 Ko	
requirements.txt	2019-12-18 02:48	Document texte	1 Ko	
util.py	2020-04-11 08:45	Fichier PY	1 Ko	

```

from tensorflow.keras.applications.imagenet_utils import preprocess_input, decode_predictions
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image

# Quelques utilitaires
import numpy as np
from util import base64_to_pil

# Déclarez une application flask
app = Flask(__name__)

# Vous pouvez utiliser un modèle pré-formé de Keras
# Check https://keras.io/applications/
from keras.applications.mobilenet_v2 import MobileNetV2
model = MobileNetV2(weights='imagenet')

print('Modèle chargé. Vérifiez http://127.0.0.1:5000/')

# Modèle enregistré avec Keras model.save ()
MODEL_PATH = 'models/your_model.h5'

# Chargez votre propre modèle formé
# model = load_model(MODEL_PATH)
# model.make_predict_function() # Nécessaire
# print('Modèle chargé. Commencez à servir ...')

def model_predict(img, model):
    img = img.resize((224, 224))

```