Rapport du projet “Image Mobile”

FashionistApp

I. Introduction

Le projet consiste à développer une application de recherche à partir d’image depuis un smartphone. Il s’agit d’un type d’application déjà très utilisé dans le monde du mobile. L’objectif de ce projet et de mettre en application toutes les connaissances apprises en cours lors des enseignements à distance :

- Service REST
- Application Android
- Traitement d’images

La réalisation de cette application comporte deux partie bien distinct. D’une part, on a le développement de l’application mobile, que nous avons appelé **FashionistApp**, permettant de prendre de photos ou d’en sélectionner une depuis les librairies. D’autre part, une partie serveur qui prend en charge toutes la parties recherche d’image similaire.

Les technologies utilisées sont les suivantes :

- Pour la partie serveur : Python + Django + OpenCV + Keras
- Pour la partie client : Xamarin

Nous avons décidé lors du projet de nous répartir les tâche, de manière à pouvoir avancer plus vite sur le projet. Ainsi, Marouane Mokaddem s’est occupé de toute la partie cliente et du développement mobile de l’application. De son côté, Léo Coletta s’est occupé de toute la partie serveur et du développement de l’API REST.

Nous allons décomposer notre rapport en deux parties. Tout d’abord, nous allons étudier la partie serveur et l’API REST, puis, dans un second temps nous allons comprendre le fonctionnement de la partie cliente et de l’application Android.

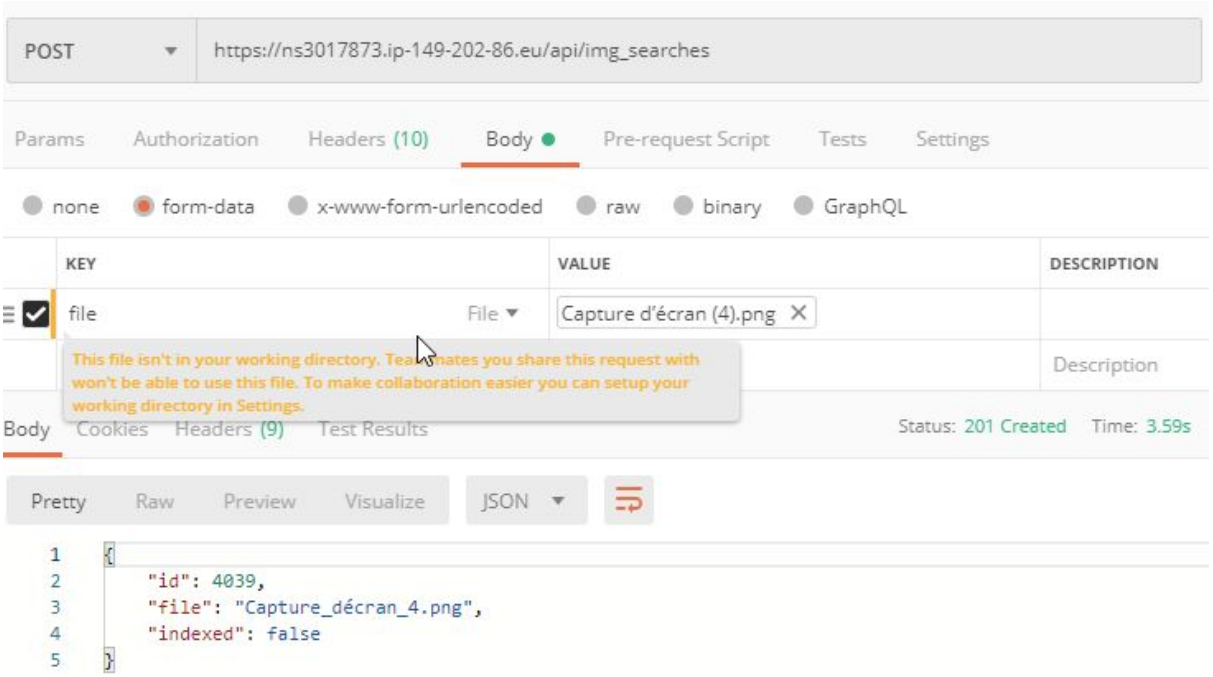
II. Partie serveur : API REST

1) Structure du service

Le service développé à tenu compte de la philosophie de conception REST qui consiste à permettre simplement la manipulation de ressources.
Chaque recherche et résultat sera associé sous la forme d’une ressource exposée en utilisant le format JSON, permettant ainsi de formuler clairement les recherches d’images.
Chaque interaction d’un client avec le serveur REST pour une recherche d’image comporte les étapes suivantes :

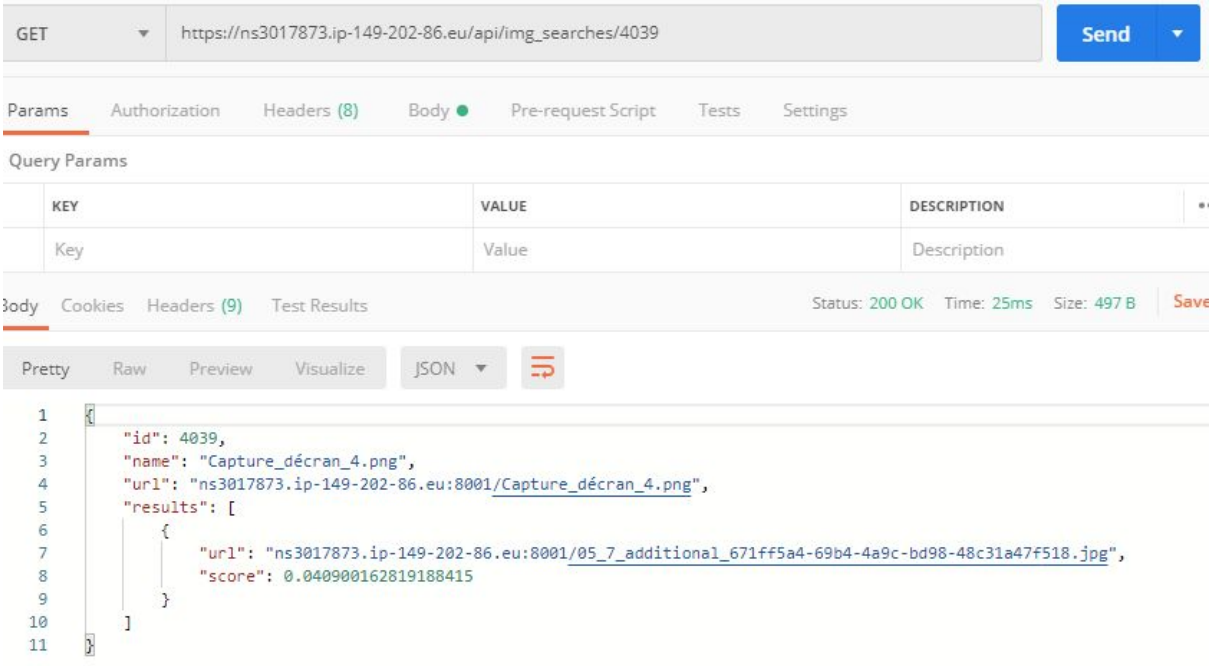
- Le client choisis d’envoyer une requête HTTP avec le verbe HTTP “POST” dans laquelle est contenue l’image fournis par l’utilisateur. A partir de la réception de la requête, le serveur exécute une recherche d’image similaires. Ensuite, les résultats

sont stockés dans une base de donnée de manière à ne pas avoir à effectuer le calcul plusieurs (calcul assez coûteux). Le serveur termine avec l'envoie d'une réponse au client qui contient l'ID correspondant à la ressource matérialisant le résultat.



Capture d'écran de la requête POST "/img_searches" sur POSTMAN

- Une fois la requête effectuée, on peut effectuer une requête HTTP GET sur la ressource afin de récupérer ses informations. Le résultat de la recherche contient l'URL des 5 images les plus similaire à l'image original et dont le score est supérieur à 0.



Requête GET "/img_searches/4039" sur POSTMAN

2) Algorithme de recherche

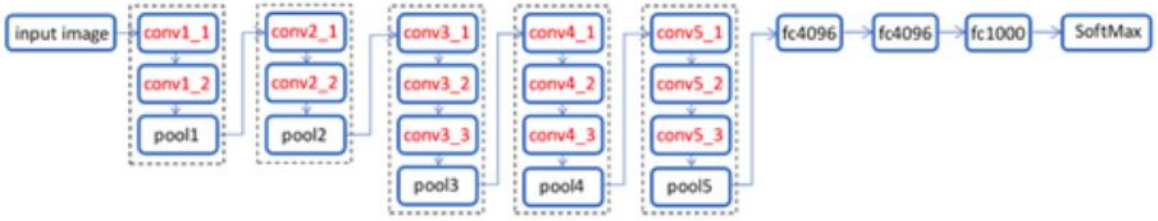
L'algorithme employé pour la recherche d'image similaire se base sur les connaissances vu en cours. Il se base d'une part sur du machine learning (réseau neuronale convolutif) afin de transformer les images en vecteur. D'autre part, il utilise un algorithme d'indexation qui va permettre d'évaluer la distance entre ces vecteurs.

a. Le réseau neuronale convolutif

Dans le domaine du deep learning, un réseau neuronal convolutif (CNN ou ConvNet) est une classe de réseaux neuronaux profonds, le plus souvent appliqué à l'analyse de l'imagerie visuelle. Ils ont des applications dans la reconnaissance d'images et de vidéos, les systèmes de recommandation, la classification d'images, l'analyse d'images médicales, le traitement du langage naturel et les séries temporelles financières.

Afin de simplifier le développement de la partie serveur, nous avons décidé d'adapter un modèle pré entraîné. De cette manière, nous évitons également d'utiliser des ordinateurs très puissant nécessaire à l'apprentissage automatique de réseau neuronaux.

Parmis les modèles pré entraînés existants, nous avons choisis de prendre le CNN appelé VGG. VGG est un réseau neuronale convolutif proposé par Karen Simonyan et Andrew Zisserman de "Oxford Robotics Institue" durant l'année 2014. Il a été dans plusieurs compétition de reconnaissance d'image et a obtenu une précision de 92,7%.C'est également l'un des plus grosse base de données d'image disponible aujourd'hui, contenant approximativement 14 million d'image labelisé par des humains.



Architecture de VGG 16

b. Indexation des images

Pour indexer la base de donnée d'image donnée par l'exercice, nous transformons toutes les photos en vecteurs.

Grâce au réseau neuronal, nous recevons une liste de mot et leur score. Notre algorithme d'indexation élimine tout de suite les mots ayant un score très faible (inférieur à 0.001), cela nous permet d'économiser de la place en base de donnée.

A partir de cela, lorsqu'une recherche est effectuée, nous transformons l'image recherché en vecteur et nous effectuons la distance de manhattan par rapport à toutes les autres images. Nous gardons dans les résultats seulement les 5 plus gros ayant un score supérieur à 0.

Partie Client : application native iOS/Android

L'application mobile, qui permet à l'utilisateur de communiquer avec la partie serveur, a été réalisée en **C#** avec le framework **Xamarin**, qui fait partie du framework **.NET**, et qui permet de créer des applications 100% natives et multiplateformes (iOS, Android, Windows 10, Xbox, etc) en se basant essentiellement sur une partie commune du code, et sur peu de code propre à chaque plateforme. Nous allons montrer comment nous avons réalisé une application **Android** et **iOS**, et comment nous l'avons testée sur **iOS Simulator**.

Architecture de l'application

Les applications .NET se basent sur le principe du **CodeBehind**, c'est-à-dire une partie graphique et une partie code, les 2 étant liées par défaut (à chaque fichier **.xaml** est associé un fichier **.cs**). L'interface commune a été réalisée en **XAML**, un langage Microsoft basé sur XML et qui permet de déclarer tous les éléments visuels et leur associations aux données de la partie code.

```
<Image x:Name="photoTakenViewer"></Image>
<Label x:Name="debugger" Text="DEBUG" IsVisible="False" BackgroundColor="LightGray" TextColor="Da
<Label x:Name="debugger2" Text="DEBUG2" IsVisible="False" BackgroundColor="LightGray" TextColor="
<Image x:Name="imageViewer" VerticalOptions="Center" />
```

Un exemple de déclaration d'éléments visuels (label, images) dans **MainPage.xaml**

```
if (choice)
{
    debugger.Text = "Sending...";
    activityIndicator.IsVisible = true;
```

Un exemple de référencement d'un de ces éléments dans **MainPage.xaml.cs** par la propriété **x:Name** spécifiée dans le XAML.

Ces éléments sont essentiellement des labels de texte, un afficheur photos, des boutons et des menus qui permettent de sélectionner différentes options. Les éléments visuels seront automatiquement "traduits" en éléments natifs iOS et Android. L'application contient un total de 3 pages : une page principale avec 3 onglets (accueil, résultats et paramètres), une page résultat avec l'affichage d'une requête et une page zoom qui permet de visualiser la version agrandie d'une photo.

Coté C#, le premier aspect dont on s'est occupé était celui de récupérer une image depuis la galerie et de l'afficher ; pour cela, Android et iOS possèdent 2 méthodes différentes, ce qui rend presque impossible écrire du code unique pouvant être exécuté sur les 2 plateformes.

```
// Present UIImagePickerController;
UIWindow window = UIApplication.SharedApplication.KeyWindow;
var viewController = window.RootViewController;
viewController.PresentModalViewController(imagePicker, true);
```

On peut remarquer la présence de classes **UIWindow** et **UIApplication**, propres au **UIKit** d'iOS, traduites en C# depuis Swift.

```
// Start the picture-picker activity (resumes in MainActivity.cs)
MainActivity.Instance.StartActivityForResult(
    Intent.CreateChooser(intent, "Select Picture"),
    MainActivity.PickImageId);
```

Alors qu'Android fait utilisation des **Intents**, propres à la plateforme.

```
public interface IPhotoPickerService
{
    Task<Stream> GetImageStreamAsync();
}
```

Mais finalement les 2 plateformes finissent par “se mettre d’accord” sur quel objet retourner une fois l’image sélectionnée : un **Stream**.

Ce Stream est récupéré pour l’envoyer directement au serveur une fois que l’utilisateur confirme l’envoi. Pour cela, nous avons fait l’utilisation de l’API **Refit**, une adaptation en C# de l’API Retrofit en Java. Cela nous permet de faciliter énormément les tâches, notamment au niveau du format du fichier à envoyer au serveur (le **StreamPart**, qui n’est pas géré nativement par les ressources REST en C#).

```
string filename = DateTime.Now.ToString(dateFormat) + ".jpg";
response = await imageServerApi.SendImage(new StreamPart(stream, filename));
queryId = response.id.ToString();
```

L’envoi du fichier en StreamPart qui s’effectue de manière assez simple et modulaire.

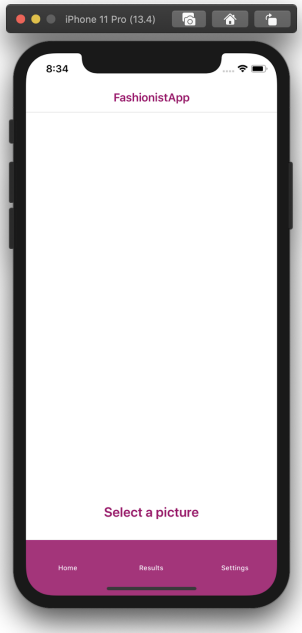
Le **response** récupéré à partir de la requête permet d’obtenir un **ID** : cet ID représente le calcul effectué côté serveur, et en lançant une requête GET avec cet ID, l’application récupère le JSON avec les 5 photos plus similaires, avec leur emplacement sur le serveur ainsi que leur similarité. L’étape finale consiste à mettre en forme le JSON et afficher les images reçues.

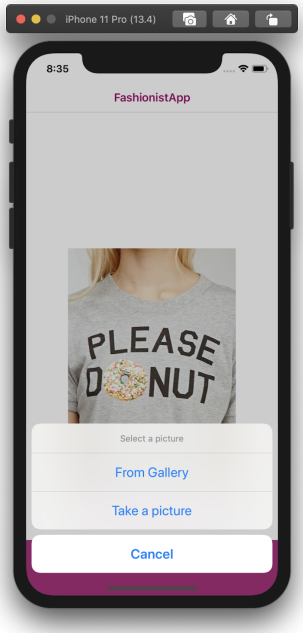
```
foreach (Result r in latestQuery)
{
    r.fullUrl = "http://" + r.url;
    r.similarityPercentage = (Math.Round(r.score, 3) * 100).ToString() + " %";
```

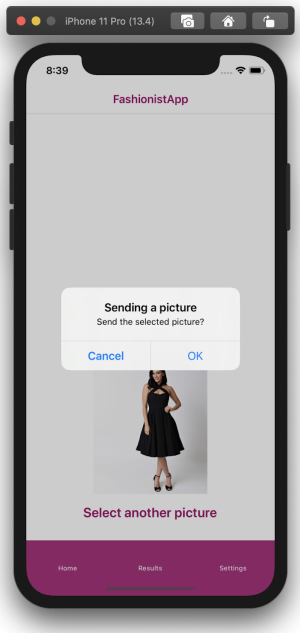
Avec un **foreach**, chaque résultat de requête est mis dans une classe **Result** qui inclut les champs adaptés, ce qui facilite sa gestion et son affichage dans un tableau.

Expérience utilisateur

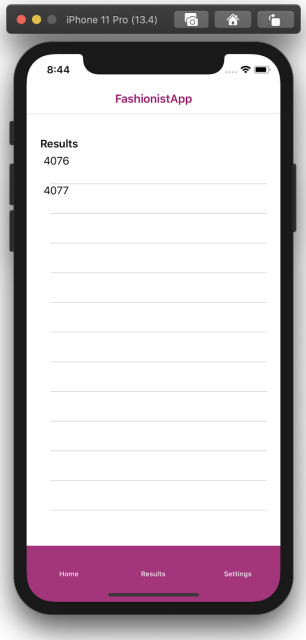
L’utilisateur aura une page d’accueil assez simple grâce à laquelle il pourra uploader une nouvelle photo (un menu pop-up s’affiche pour qu’il puisse sélectionner la Galerie ou la caméra) ou accéder à la liste des recherches qu’il a effectué.

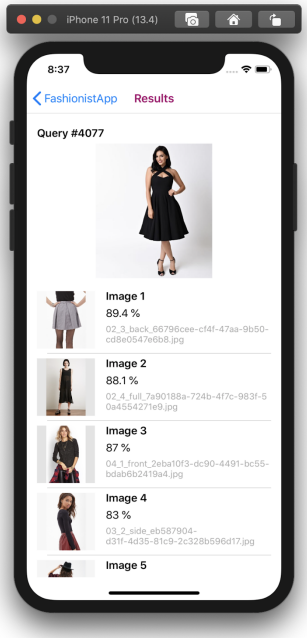


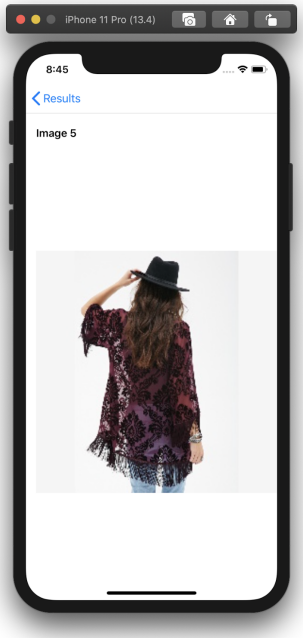




Une fois l'image sélectionnée, l'utilisateur pourra confirmer l'envoi de l'image sur le serveur, ou de revenir en arrière pour sélectionner une autre image. Quand l'image est envoyée sur le serveur, et que la réponse est reçue, la page de résultats s'affiche avec la photo envoyée ainsi que les 5 vêtements plus similaire avec leur pourcentage de similarité. Il est aussi possible de zoomer sur chaque image et d'accéder à la liste des dernières recherches.







Bilan et conclusion

Ce projet nous a permis d'appliquer toutes nos connaissances sur le CNN mais aussi sur les services REST, sur le langage Python et sur le développement mobile.

Concernant l'application, nous avons décidé de passer à "un niveau supérieur" en essayant de tester nos compétences en termes de développement mobile. Nous avons choisi une technologie, Xamarin, que nous n'avons pas abordé pendant les cours et par conséquent il nous était difficile d'obtenir des conseils de la part des enseignants. Il était primordial pour nous donc de savoir s'auto-documenter, mais cela a été facilité relativement par notre expérience précédente dans ce framework.

Nous avons été satisfaits de pouvoir rendre un travail où nous avons appliqué toute notre expérience à l'IMT mais aussi tout ce que l'on a appris à partir de nos projets personnels.