Projet Oxford et Intelligence artificielle

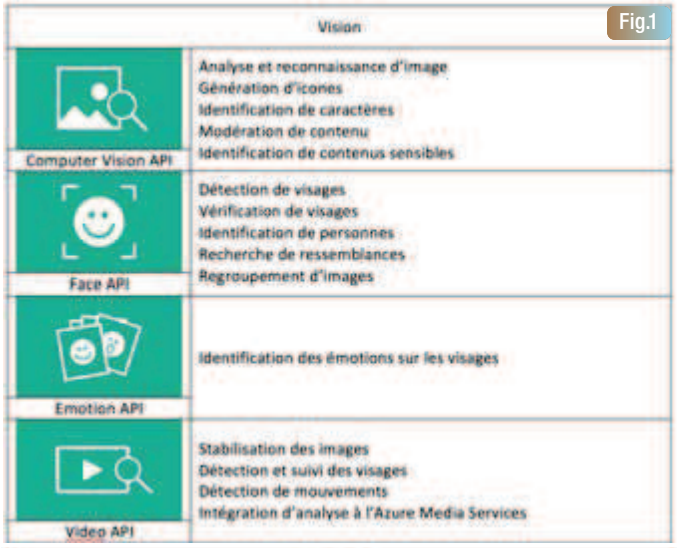
Vous connaissez sans doute les sites comme How-Old.Net, TwinsOrNot.net et le tout dernier MyMoustache.Net. Derrière ces sites très ludiques avec lesquels vous avez sans doute joué avec vos propres photos puis celles de vos amis, se cache un point commun : l'intelligence artificielle ou encore appelée le machine learning basée sur des algorithmes complexes à concevoir et à implémenter pour des développeurs qui ne sont pas des experts de l'AI (Artificial Intelligence). Pour faciliter l'accès à ce domaine, Microsoft a mis à disposition des développeurs le Projet Oxford !

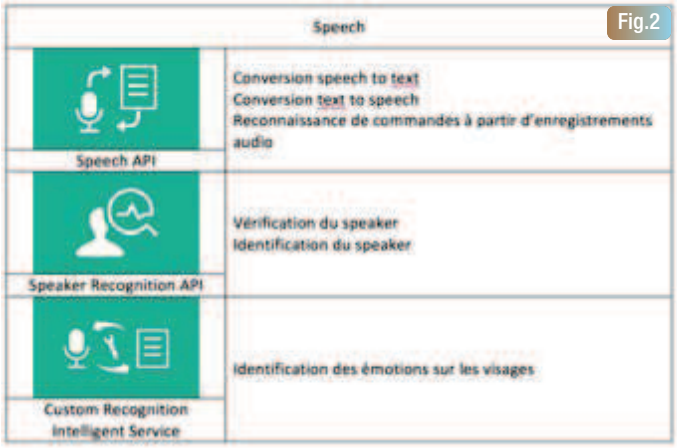
Michel Hubert,
Directeur Technique
Cellenza

Jonathan Pamphile,
Consultant Cloud
Cellenza

Guillaume Demichelli,
Consultant mobilité
Cellenza

Grâce au projet Oxford, les développeurs peuvent facilement ajouter de l'intelligence dans leurs applications. Le Projet Oxford est juste une application d'un projet plus global mené par Microsoft autour de l'intelligence artificielle. Les cas d'usage sont multiples et très Hype mais vos applications vont pouvoir dès à présent pouvoir voir, écouter, parler, comprendre et pourquoi pas commencer à raisonner.





Qu'est ce que le « Projet Oxford » ?

Le « Projet Oxford » est le nom d'une intelligence artificielle développée par Microsoft pour les développeurs afin de leur permettre de traiter, analyser et faire de la reconnaissance à partir d'images, de sons et de vidéos. Elle est basée sur le machine learning, exploite les possibilités et ressources du Cloud Computing et est mise à disposition sous forme d'APIs. Ainsi, les développeurs ont la possibilité d'enrichir les interactions que peuvent avoir les utilisateurs avec leurs applications ou d'en concevoir de nouvelles.

Les services

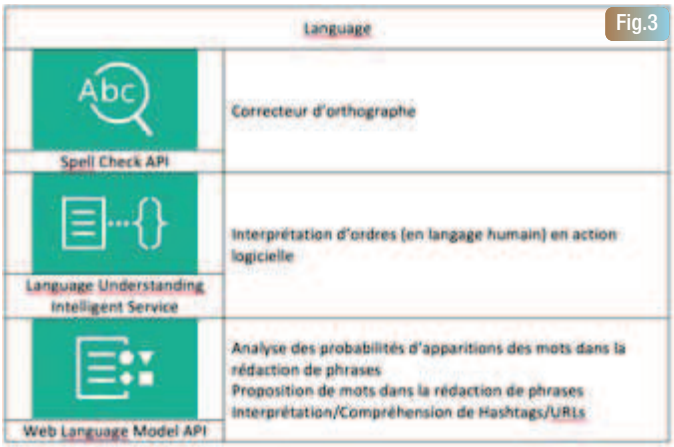
Les services proposés sont regroupés en trois thématiques : « Vision », « Speech » et « Language ». Fig.1, 2 et 3.

Exemple d'utilisation : Service de reconnaissance faciale

Quoi de mieux pour vous parler/présenter l'API Oxford qu'un exemple concret et un peu de code ?

Prenons alors le cas d'une application devant utiliser la reconnaissance faciale pour identifier l'utilisateur à partir de son visage. Notre solution est composée d'une application Universelle et d'une couche de service en API App hébergée sur Azure. Fig.4.

Au lancement de l'application universelle, déployée sur Windows 10 IoT Core, Phone et Tablette, l'utilisateur demande à être authentifié. L'application prend alors une photo de lui, demande l'identification à notre API App, qui lancera la reconnaissance faciale via l'API Face du projet Oxford, et si l'utilisateur est reconnu, récupère ses préférences et affiche à l'écran des widgets qu'il aura sélectionnés et pour lesquels il aura choisi une position à l'écran. Les APIs du projet Oxford sont basées sur du machine learning. Par conséquent, une composante essentielle de la solution est l'apprentissage. Afin d'utiliser le service de reconnaissance faciale, nous allons donc commencer par créer les profils des personnes à reconnaître. Ensuite, au fil du temps, nous allons enrichir la base de connaissance de l'API et l'entrainer. Pour ce faire, nous utiliserons une méthode de training que l'API nous met à disposition qui permettra d'enrichir sa base de connaissance pour améliorer son mécanisme d'identification.



L'initialisation

Création d'un groupe d'identification

La première étape, pour pouvoir utiliser l'API Face du projet Oxford, est de créer un groupe de personnes. C'est dans ce groupe que seront créées les personnes que l'API tentera de reconnaître, ainsi que les photos qui leurs seront associées.

```
var faceClient = new FaceServiceClient(OxfordFaceKey);
var groups = await faceClient.GetPersonGroupsAsync();
var moriotGroup = groups.SingleOrDefault(o => o.Name == MiriotPersonGroup);
if (moriotGroup == null)
{
await faceClient.CreatePersonGroupAsync(MiriotPersonGroup, MiriotPersonGroup);
await faceClient.TrainPersonGroupAsync(MiriotPersonGroup);
}
```

Suite à cela, nous pouvons procéder à la création de nos personnes à identifier dans le groupe.

La création d'une personne à identifier

- La création des personnes à identifier se fait en quatre étapes :
- La détection du visage ;
- La création de la personne dans la base de connaissance de l'API à partir de son visage ;
- La sauvegarde de l'utilisateur dans notre base de données ;
- L'entrainement de l'API. Fig.5.

La détection du visage

La détection consiste à repérer les visages présents sur la photo.
La détection nous renvoie une liste de visages (classe « Face »). Chaque « Face » contient un certain nombre d'informations telles que :

- Un identifiant ;
- Sa position sur la photo ;
- Sa taille ;
- Le sexe de la personne ;
- Son âge aproximatif ;
- Souriant ? Barbu ? Moustachu ?
- Les traits de son visage ;
- Etc.

Pour cette étape, nous avons fait le choix de ne traiter une photo que si la personne est seule dessus. Dans le cas contraire, nous renvoyons une erreur de type « BadRequest ».

```
List<Face> faces;
using (var stream = new MemoryStream(request.Image))
  faces = (await faceClient.DetectAsync(stream)).ToList();

var faceIds = faces.Select(o => o.FaceId).ToList();

if (faceIds.Count != 1)
return BadRequest();
```

La création et la sauvegarde de la personne

Dans cette étape, nous allons commencer par rajouter à notre groupe de personnes notre nouvel utilisateur, à partir du visage que nous avons détecté. Lors de l'appel de la méthode « CreatePersonAsync », nous renseignons le nom de cette personne et le groupe auquel nous la rajoutons.

```
var result = await faceClient.CreatePersonAsync(MiriotPersonGroup, request.Name);
if (result == null)
return InternalServerError();

await _storage.CreateOrUpdateUserInTableStorageAsync(result.PersonId, request.Name,
request.Widgets);
```

Le retour de la création est un résultat de création (classe « CreatePersonResult »). Cette classe une propriété « PersonId » de type Guid, qui est l'identifiant de la personne.
Une fois le « PersonId » obtenu, nous sauvegardons les données de notre nouvel utilisateur dans le stockage que nous avons choisi. Ici, le stockage est fait dans une table Azure (stockage clé/valeur, avec comme clé le « PersonId », et comme valeur l'objet « User » sérialisé en Json).

```
public async Task CreateOrUpdateUserInTableStorageAsync(Guid personId, string name, string widgets)
{
var table = _tableClient.GetTableReference(StorageAccount.UsersTableName);
var userOperation = TableOperation.Retrieve<UserEntity>("User", personId.ToString());
var userOperationResult = (await table.ExecuteAsync(userOperation)).Result as UserEntity;

if (userOperationResult != null)
{
userOperationResult.Name = name;
userOperationResult.Widgets = widgets ?? userOperationResult.Widgets;

var insertOrReplaceOperation = TableOperation.InsertOrReplace(userOperationResult);
await table.ExecuteAsync(insertOrReplaceOperation);
}
else
{
var setting = new UserEntity(personId, name, string.Empty);
```

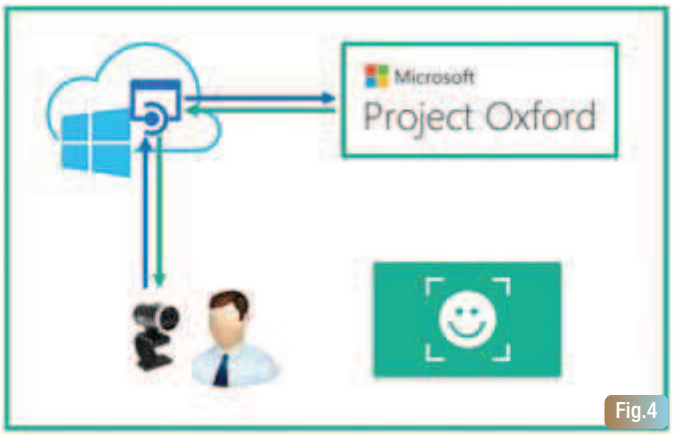


Fig.4

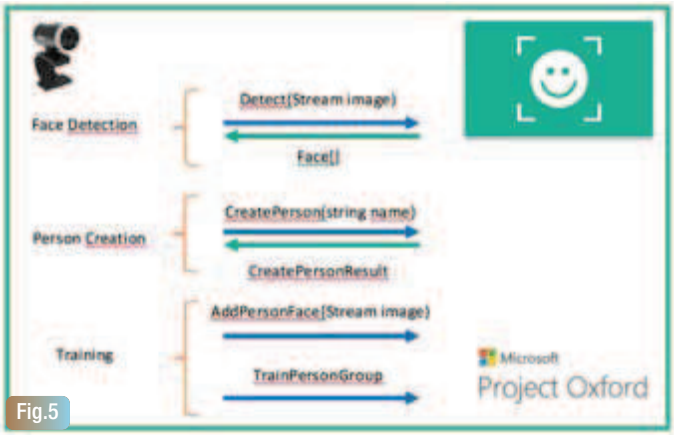


Fig.5

```
var insertOperation = TableOperation.Insert(setting);
await table.ExecuteAsync(insertOperation);
}
}
```

L'entrainement

Une fois notre nouvel utilisateur créé, nous allons alimenter la base de connaissance de l'API Oxford avec sa photo.

```
await _storage.AddPersonFaceAsync(result.PersonId, request.Image);
```

Cette étape se fait de manière asynchrone (afin qu'elle ne bloque pas le processus de création de compte). Nous sauvegardons l'image dans un stockage de type blob sur Azure et mettons dans une queue Azure un message pour que le traitement soit effectué par un WebJob.

```
public async Task AddPersonFaceAsync(Guid id, byte[] image)
{
var container = _blobClient.GetContainerReference(StorageAccount.UsersImagesContainer
Name);
var blob = container.GetBlockBlobReference($"{DateTime.UtcNow.ToString("yyMMddhh
mmss")}_{id}.jpg");

using (var stream = new MemoryStream(image))
await blob.UploadFromStreamAsync(stream);

var request = new UserFaceUpdateRequest
{
Id = id,
Image = blob.Name
};

var queue = _queueClient.GetQueueReference(StorageAccount.UsersQueueName);

await queue.AddMessageAsync(new CloudQueueMessage(JsonConvert.SerializeObject
(request)));
}
```

Notre WebJob, à l'ajout d'un message dans la queue Azure, récupère le message et le traite en ajoutant la photo à la liste des photos de l'utilisateur précédemment créé en appelant la méthode « AddPersonFaceAsync ». Il envoie alors le nom du groupe auquel a été ajouté la personne, son identifiant et la photo à rajouter.

Une fois la photo ajoutée, on appelle la méthode « TrainPersonGroupAsync », permettant de lancer l'apprentissage de l'API.

```
public async static Task ProcessQueueMessage([QueueTrigger("users")] UserFaceUpdate
Request request, TextWriter log)
{
var cloudAccount = CloudStorageAccount.Parse(StorageConnectionString);
var blobClient = cloudAccount.CreateCloudBlobClient();
var container = blobClient.GetContainerReference(ContainerName);
var blob = container.GetBlockBlobReference(request.Image);

var faceClient = new FaceServiceClient(OxfordKey);

using (var stream = await blob.OpenReadAsync())
await faceClient.AddPersonFaceAsync(MiriotPersonGroup, request.Id, stream);

await faceClient.TrainPersonGroupAsync(MiriotPersonGroup);
}
```

La reconnaissance faciale

Une fois les profils de nos utilisateurs créés (et sauvegardés via notre API App), nous pouvons lancer la reconnaissance. Nous avons choisi de la faire en trois étapes :

- La détection des visages ;
- L'identification des personnes à partir des visages ;
- L'entrainement. Fig.6.

La détection des visages

La détection du visage est similaire à la partie précédente, à la différence qu'ici, nous avons choisi de ne pas nous restreindre à une seule personne présente sur la photo. Dans le cas où nous ne parvenons à détecter aucun visage, nous renvoyons une erreur « NotFound ».

```
var faceClient = new FaceServiceClient(OxfordFaceKey);
List<Face> faces;

using (var stream = new MemoryStream(request.Image))
faces = (await faceClient.DetectAsync(stream)).ToList();

if (faces.Count == 0)
return NotFound();
```

L'identification des personnes

L'identification consiste à reconnaître les personnes à partir des visages détectés. Elle se fait en appelant la méthode « IdentifyAsync », en lui passant le nom du groupe où chercher la personne, ainsi que la liste des identifiants des visages détectés.

```
var result = (await faceClient.IdentifyAsync(MiriotPersonGroup,
            faces.Select(o => o.FaceId).ToArray())).ToList();

if (result.Count == 0 || !result.Any(o => o.Candidates.Any()))
return NotFound();
```

Le retour de l'identification est une liste de résultats d'identification (classe « IdentifyResult »). Cette classe contient une liste de candidats potentiels (classe « Candidate ») ayant une propriété « PersonId » de type Guid, qui est l'id de la personne que Oxford pense avoir identifiée, ainsi qu'une propriété « Candidate » de type double qui correspond au niveau de confiance. Dans notre cas, nous avons choisi de ne remonter qu'une seule personne identifiée et avons choisi arbitrairement de ne garder que la personne identifiée ayant le meilleur indice de confiance. Dans le cas où aucune personne n'a été identifiée, nous renvoyons une erreur « NotFound ».

```
var moreConfidentPerson = result.SelectMany(p => p.Candidates)
            .OrderByDescending(o => o.Confidence).First();
```

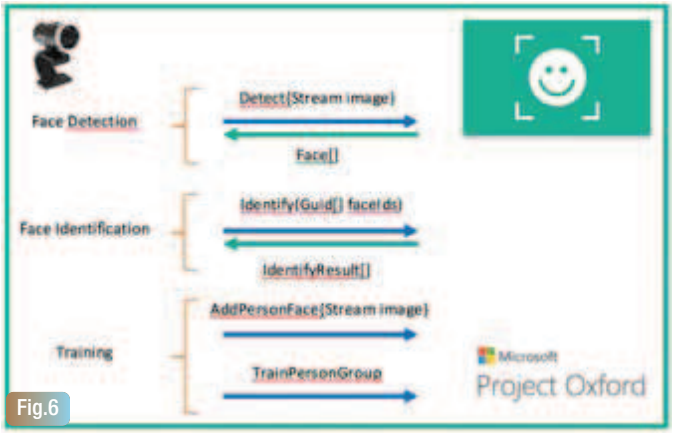


Fig.6

```
var user = await _storage.GetUserFromTableStorageAsync(moreConfidentPerson.PersonId);
```

Ainsi, une fois le « PersonId » obtenu, nous récupérons les données de notre utilisateur identifié depuis le stockage que nous avons choisi. Pour rappel, dans notre cas, le stockage a été fait dans une table Azure (stockage clé/valeur, avec comme clé le « PersonId », et comme valeur l'objet « User » sérialisé en Json).

```
public async Task<User> GetUserFromTableStorageAsync(Guid personId)
{
var table = _tableClient.GetTableReference(StorageAccount.UsersTableName);
var userOperation = TableOperation.Retrieve<UserEntity>("User", personId.ToString());
var userOperationResult = (await table.ExecuteAsync(userOperation)).Result as UserEntity;

if (userOperationResult != null)
{
return new User
{
Id = Guid.Parse(userOperationResult.RowKey),
Name = userOperationResult.Name,
Widgets = JsonConvert
.DeserializeObject<List<Widget>>(userOperationResult.Widgets)
};
}

return null;
}
```

L'entraînement

Tout comme lors de la création de l'utilisateur, nous avons choisi d'enrichir la base de connaissance de l'API de reconnaissance faciale lorsqu'un utilisateur est identifié. Pour cela, après l'identification, nous rajoutons la photo prise à la liste des photos connues par l'API pour notre utilisateur par le même mécanisme asynchronisé présenté lors de la création de l'utilisateur. Ainsi, au fil du temps, plus l'utilisateur se sert de l'application et demande à être identifié, plus le mécanisme de reconnaissance de l'API Oxford s'améliorera.

```
await _storage.AddPersonFaceAsync(moreConfidentPerson.PersonId, request.Image);
```

Bonus : Service de reconnaissance d'émotions

L'utilisation du service de reconnaissance d'émotions est très proche de celui de reconnaissance faciale. Pour l'utiliser, il suffit d'appeler la méthode « RecognizeAsync » de l'Emotion API, en passant en paramètre la photo sur laquelle détecter les émotions.

En retour de cet appel, nous récupérons une liste d'émotions (classe « Emotion »). Cette classe contient une propriété « FaceRectangle », que l'on pourra comparer à la propriété du même nom que l'on retrouve dans le résultat de la détection des visages. On y retrouve également une propriété « Scores », qui est une classe contenant les différentes émotions que peut reconnaître Oxford, chacune indiquant le niveau de l'émotion détectée :

- Anger ;
- Contempt ;
- Disgust ;
- Fear ;
- Happiness ;
- Neutral ;
- Sadness ;
- Surprise.

Dans notre cas, pour identifier l'émotion de la personne que nous avons le mieux reconnue sur la photo, nous récupérons l'émotion correspondant à son visage (à partir de la position de son visage grâce à la propriété « FaceRectangle »), et nous prenons l'émotion la plus prononcée que Oxford aura identifiée.

```
List<Emotion> detectedEmotions;
var detectedEmotions = new EmotionServiceClient(OxfordEmotionKey);

using (var stream = new MemoryStream(request.Image))
detectedEmotions = (await emotionClient.RecognizeAsync(stream)).ToList();

var selectedEmotion = detectedEmotions
.SingleOrDefault(o => o.FaceRectangle.Top == face.FaceRectangle.Top
&& o.FaceRectangle.Left == face.FaceRectangle.Left);
if (selectedEmotion != null)
{
var emotions = new Dictionary<UserEmotion, float>
{
{UserEmotion.Anger, selectedEmotion.Scores.Anger},
{UserEmotion.Contempt, selectedEmotion.Scores.Contempt},
{UserEmotion.Disgust, selectedEmotion.Scores.Disgust},
{UserEmotion.Fear, selectedEmotion.Scores.Fear},
{UserEmotion.Happiness, selectedEmotion.Scores.Happiness},
{UserEmotion.Neutral, selectedEmotion.Scores.Neutral},
{UserEmotion.Sadness, selectedEmotion.Scores.Sadness},
{UserEmotion.Surprise, selectedEmotion.Scores.Surprise}
};

user.Emotion = emotions.OrderByDescending(o => o.Value).First().Key;
}
```

Les tarifs

Service de reconnaissance faciale

A ce jour, le service de reconnaissance faciale est disponible en deux niveaux de service :

- Version gratuite propose 5000 transactions par mois ;
- Version standard permettant 10 transactions par seconde facturée 1,5 $ les 1000 transactions.

Service de reconnaissance d'émotions

A ce jour, le service de reconnaissance d'émotion est lui disponible en trois niveaux de service :

- Version gratuite propose 5000 transactions par mois ;
- Version basique permettant 10 transactions par seconde facturée 0,10 $ les 1000 transactions, avec les zones de détection identifiées ;
- Version standard permettant 10 transactions par seconde facturée 0,25 $ les 1000 transactions.

Conclusion

Cet article vous démontre la facilité de mise en œuvre d'intelligence dans vos applications pour un développeur, il « suffit » d'appeler des APIs. Nous en sommes actuellement aux prémices et de nombreux éditeurs proposent de plus en plus leurs services comme Microsoft. Il reste néanmoins un point contraignant : être connecté à Internet, car ces services ne fonctionnent pas en mode déconnecté parce qu'ils nécessitent notamment la puissance du Cloud pour fonctionner. Cet article met en exergue la reconnaissance visuelle, mais essayez les APIs autour de la voix (Speech) comme la reconnaissance vocale et vous serez rapidement séduit. A vous de créer les applications autour de ces API.