

Find3r, El Fantomas, Tipchef et Triguneur présentent



RAPPORT DE SOUTENANCE

ocr^x

A brand new optical character
recognition system.

Table des matières

Introduction	3
1 Préface 🏠	4
1.1 Objectifs de cette soutenance	4
1.2 Répartition des tâches	5
1.3 Planning des soutenances	5
1.4 Barre de développement	6
2 L'équipe 👥	7
2.0.1 Tipchef.	8
2.0.2 Find3r	8
2.0.3 El Fantomas	9
2.0.4 Triguneur	9
3 Le projet 🚀	10
3.1 Processus et traitement de l'image	10
3.1.1 Passage de l'image en noir et blanc	10
3.1.2 Détection des lignes	11
3.1.3 Détection des caractères	12
3.2 Le réseau de neurones	13
3.2.1 Initialisation	13
3.2.2 Apprentissage	13
3.2.3 Sauvegarde des poids	14
3.2.4 Progression	14

3.3	Expérience & interface utilisateur (UX/UI)	15
3.3.1	Fenêtre <i>Main</i>	16
3.3.2	Fenêtre <i>About</i>	16
3.3.3	Fenêtre <i>Result</i>	17
4	Conclusion	18

Introduction



Voici OCRx.

OCRx est un système de reconnaissance optique de caractères conçu dans le cadre du projet d'INFO-SPÉ de l'EPITA¹. Sa durée s'étend sur un semestre lors du Semestre 3 du cycle préparatoire. Ce rapport de soutenance décrit la nature du projet ainsi que les différentes parties du logiciel.

Ce projet est la création d'un système de reconnaissance optique de caractères, sous la forme d'un logiciel développé en C. Un tel système extrait le texte contenu dans une image (photographie, texte dactylographié, document scanné, etc.) et conçoit à partir du document fourni, un document texte contenant les informations textuelles du document initial. Pour ce projet, nous avons utilisé le standard C99 du C.

Le groupe est constitué de *Tom-Eliott Herfray*, de *Paul Viallet*, d'*Edward Cacioppo* ainsi que d'*Alexis Huard*. Le logiciel sera disponible sur notre site à la fin du projet.

Nous vous souhaitons une bonne lecture de notre premier rapport de soutenance! 📖

1. École d'ingénieurs spécialisées en informatique (epita.fr)

2. Site du projet : OCRx.focalstudio.me

Préface

1.1 Objectifs de cette soutenance

Pour cette première soutenance, nous avons suivi les objectifs énoncés dans le cahier des charges, c'est-à-dire la conception d'un réseau de neurones capable d'apprendre la fonction logique XOR ¹ ainsi que d'un système de chargement des images, de suppression des couleurs et de détection-découpage des blocs fournis par l'utilisateur.

De plus, nous avons avancé de façon importante le réseau de neurones en amont, et nous avons également commencé (et bien avancé) une interface graphique (voir la section *UX-UI*) et un site web (qui sera disponible à la dernière soutenance à l'adresse ocrx.focalstudio.me et qui contiendra les sources de notre projet²).

1. *OU Exclusif* pour les francophones

2. Le site est déjà fini, mais pour éviter la triche de la part d'autres groupes, nous évitons de publier notre code sur Internet.

1.2 Répartition des tâches

	Cacioppo	Herfray	Viallet	Huard
Segmentation de l'image	■			
Gestion de l'UI et de l'UX (GUI)		■		
Conception du réseau de neurones			■	
Traitement de l'information				■

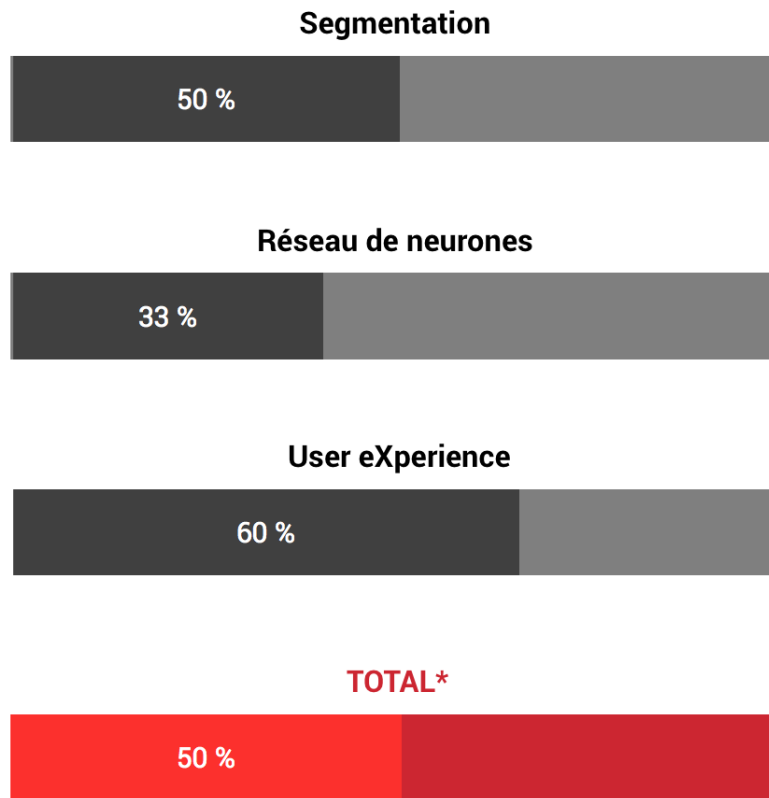
1.3 Planning des soutenances

	Intermédiaire	Finale
Chargement des images et suppression des couleurs	★★	★★
Détection et découpage en blocs	★★	★★
Logique combinatoire et fonction XOR	★★	★★
Chargement du poids du réseau de neurones	★	★★
Jeu d'images pour l'apprentissage	★	★★
Manipulation des fichiers	★	★★
Apprentissage du réseau de neurones		★★
Reconstruction du texte et sauvegarde		★★
Conception du design UI/UX	★	★★
Site web du projet	★	★★

★	Tâche commencée
★★	Tâche finalisée

Nota Bene : Des éléments supplémentaires sont susceptibles d'être ajoutés suivants l'avancement du projet. Dans ce dernier cas, ils seront détaillés au sein du rapport de projet.

1.4 Barre de développement



* À la 1ère soutenance

État de l'avancement actuel du projet à la soutenance intermédiaire.

Nota Bene : L'User eXperience regroupe l'interface utilisateur (GUI), le site web et les documents rédigés en \LaTeX .

L'équipe

Notre équipe se compose actuellement de :

- **Edward CACIOPPO** (*edward.cacioppo@epita.fr*)
- **Tom-Eliott HERFRAY** (*tom-eliott.herfray@epita.fr*)
- **Paul VIALLET** (*paul.viallet@epita.fr*)
- **Alexis HUARD** (*alexis.huard@epita.fr*)

2.0.1 Edward Cacioppo



Edward 'Tipchef' CACIOPPO

« Je m'appelle Edward Cacioppo, j'ai 20 ans et je suis en Info-Spé à EPITA. J'aime personnellement la science-fiction et la fantasy, j'apprécie plus particulièrement les œuvres de Tolkien, Arthur C. Clark et Martin. Ayant déjà travaillé sur un projet durant le S2, j'ai, par conséquent, de l'expérience en travail d'équipe. Mais ici, le projet se fait sans l'utilisation d'un logiciel qui réalise la majeure partie du travail. Ainsi, pour ce projet, je suis en charge de la segmentation avec Alexis, et **j'espère que vous apprécierez le (petit) fruit de notre travail.** 🙌 »

2.0.2 Tom-Eliott Herfray



Tom-Eliott 'Find3r' HERFRAY

« Geek, fan de Sci-Fi depuis mon enfance, je m'intéresse beaucoup aux nouvelles technologies (c'est rare ici, je sais). J'aime le DIY (Do It Yourself) et j'adore concevoir des projets, allant d'un casque de réalité virtuelle à un traducteur vocal intuitif, en passant par un jeu vidéo sur le thème de Star Wars. Ce dernier, projet de première année à EPITA, était une excellente expérience pour (re)découvrir le C# et Unity. Pour cette seconde année, nous entrons dans la cour des grands, et OCRx est l'occasion de passer aux choses sérieuses. Mon mantra ? **Pensez différemment!** 🙌 »

2.0.3 Paul Viallet



Paul 'El Fantomas' VIALLET

« Fan de manga, notamment des oeuvres de Leiji Mastumato, et d'histoire, j'ai pendant longtemps pensé m'orienter vers une filière en rapport avec l'histoire. Mais la découverte de Minecraft en 2010 en a voulu autrement : grâce à ce jeu, j'ai découvert les joies de la programmation et du monde de l'informatique, c'est ainsi que j'ai intégré EPITA. Ma nature touche à tout, du fait que je possède de nombreuses connaissances et que, une fois lancé, je suis capable d'accumuler de nombreuses d'informations. Grâce à OCRx, j'ai d'ores et déjà pu faire d'immenses progrès en C et notamment vis à vis de l'utilisation de gdb. **We work in the dark to serve the light, we are... programmers ☺** »

2.0.4 Alexis Huard



Alexis 'Triguneur' HUARD

« Le projet de mon S2 était pour moi une première expérience de projet de groupe, me faisant ainsi découvrir le développement de programme en groupe avec certaines contraintes. Le projet de cette année est pour moi l'occasion de découvrir plus profondément le développement de programme plus complexe comme pourrait l'être l'OCR. De plus, ce projet me permet de découvrir de nouveaux algorithmes comme ceux que l'on utilise pour la segmentation ou pour le machine-learning. **Un jour j'ai réussi à faire un code sans bugs, et puis on m'a dit que je m'étais trompé de sujet. 🤖** »

Le projet 🚀

3.1 Processus et traitement de l'image

Pour cette première soutenance, nous nous sommes tenus à un algorithme de base pour gérer la segmentation. Nous avons ainsi décidé de simplement gérer un texte en noir et blanc sans différenciation entre les paragraphes et les différentes zones de textes dans l'image.

hello there

3.1.1 Passage de l'image en noir et blanc

Pour que la segmentation fonctionne correctement il nous faut avoir une image dénuée de couleurs et n'avoir que du blanc et du noir.

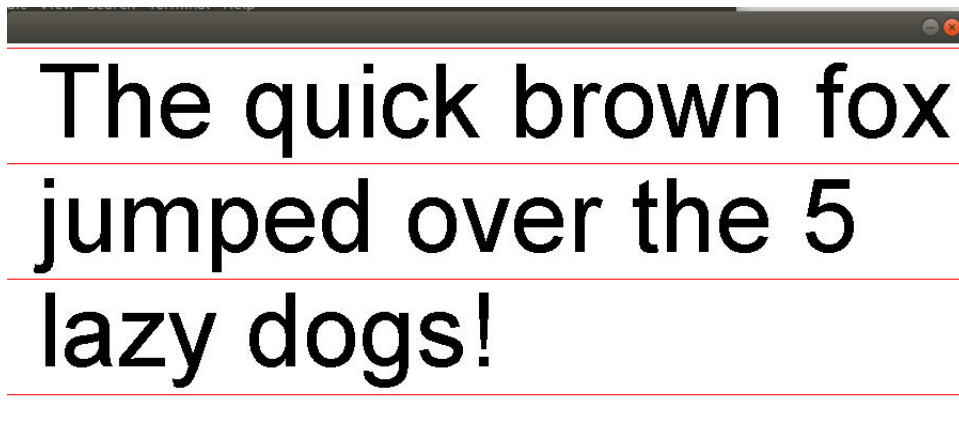
Pour cela nous devons traiter l'image grâce à un algorithme qui s'occupe de la transformer. Le traitement se fait alors en plusieurs étapes, nous récupérons tout d'abord les données RGB de l'ensemble des pixels, et grâce à aux données récupérées, nous calculons le niveau de gris correspondant à chaque pixel.

Finalement, nous avons choisi une teinte de gris qui nous sert de repère pour ainsi décider de la couleur qu'un pixel devra prendre. Un pixel de couleur gris plus clair devient un pixel blanc, et inversement, noir si le gris est plus foncé.

3.1.2 Détection des lignes

La détection des lignes est une étape qui est à la fois cruciale et simple; Elle est cruciale car elle nous permet de découper les caractères, mais simple car il nous suffit de détecter la première ligne pour ainsi aisément détecter le reste des lignes. L'algorithme fonctionne de cette manière, on exécute une étude des pixels horizontalement. C'est-à-dire qu'on traverse l'image en horizontal en étudiant les pixels allant de gauche à droite jusqu'à la fin de l'image, puis arriver à la fin de l'image, nous descendons en vertical d'un pixel et nous recommençons. Tout d'abord, on doit trouver le début de la ligne, en détectant un pixel noir. Lorsque le pixel est trouvé, il suffit alors de descendre d'un pixel en vertical. Mais cette fois, l'algorithme s'arrête lorsque l'on réussit à aller au bord droit de l'image sans rencontrer de pixel noir, donc lorsqu'on a rencontré uniquement des pixels blancs, c'est par conséquent la fin de la ligne.

L'algorithme *return* donc la distance entre le bord haut du texte et la fin horizontale de la ligne, on trace alors un trait horizontal rouge de la distance que l'algorithme nous renvoie. On utilise ensuite cette distance pour tracer les traits rouge entre chaque ligne puisque la distance qu'on reçoit est la même entre la fin de deux lignes. Cet algorithme est sans faille sachant que la première ligne commence toujours avec une majuscule qui est plus grande que les minuscules et ne possède pas de "point" comme les lettres *i* et *j*.



Détection des lignes

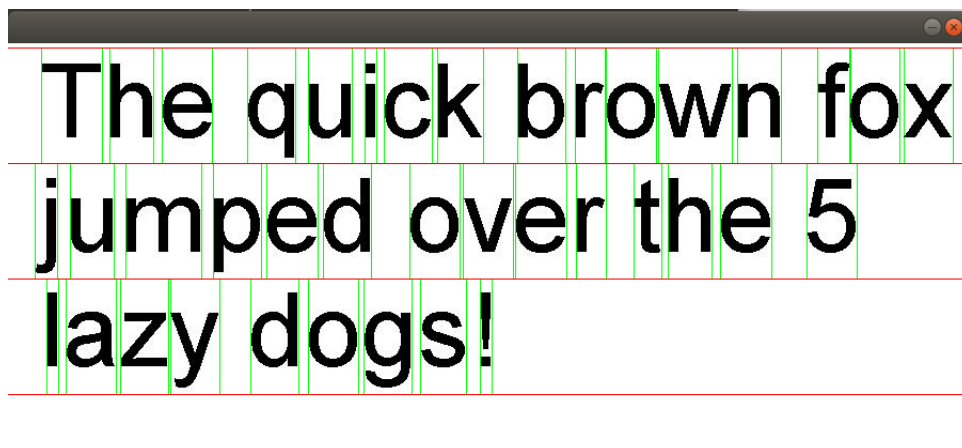
3.1.3 Détection des caractères

C'est ici que les traits horizontaux rouges vont nous servir, car cette fois l'algorithme devra traverser verticalement l'image pour trouver un trait rouge et donc, une ligne, pour ensuite faire un second voyage en vertical, mais cette fois depuis le trait rouge, jusqu'au

prochain trait rouge, c'est-à-dire la fin de la ligne. Le procédé est simple : il faut tout d'abord trouver un caractère, l'algorithme devra donc trouver un pixel noir, et pour cela : Il traverse verticalement et s'il ne trouve pas de pixel noir, il se décale d'un pixel vers la droite, et ainsi de suite. Lorsqu'il trouve un pixel noir, il trace alors un trait vertical vert allant des 2 traits rouge.

Ainsi, quand l'algorithme sait qu'il a découvert un caractère, il cherche à rencontrer le trait rouge qui signifie la fin de la ligne sans rencontrer de pixel noir, toujours en se déplaçant d'un pixel vers la gauche. Ainsi, lorsque l'algorithme valide le fait de n'avoir pas rencontré de pixel noir, il trace un trait vertical vert qui implique donc la fin du caractère.

L'algorithme va ensuite sur la même ligne lancer la recherche d'un autre caractère, et s'il arrive à la fin du texte, il va alors chercher, à partir du trait rouge, une nouvelle ligne et répéter le même procédé, ainsi de suite jusqu'à avoir fini le texte.



Détection des caractères

3.2 Le réseau de neurones

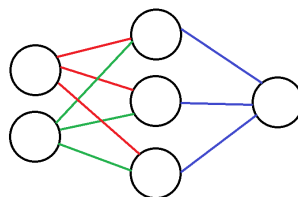
Partie centrale du projet, le réseau de neurones permet à la suite d'un apprentissage de reconnaître un *pattern* spécifique avec des variations. Pour cette première soutenance, nous avons réalisé un simple réseau de neurones pouvant reconnaître la fonction XOR.

3.2.1 Initialisation

Le réseau est sauvegardé dans une `struct`, possédant volontairement plus de variables que actuellement utilisées. Dans la mesure où la fonction XOR est relativement simple, le réseau est pour l'instant configuré pour avoir uniquement trois couches de neurones. Afin de stocker les poids reliant les différents neurones, nous utilisons des matrices, où `ih` signifie "*input*" -> "*hidden*" et `ho` "*hidden*" -> "*output*". Les valeurs de ces matrices sont aléatoires ou peuvent être chargées depuis un fichier de configuration. La première couche comporte 2 neurones (correspondantes aux 2 entrées de la fonction XOR), la seconde couche comporte 3 neurones puis 1 sortie.

3.2.2 Apprentissage

Pour l'apprentissage, nous utilisons la *backpropagation* : le réseau calcul un résultat et si celui-ci est erroné, nous lui indiquons les résultats qu'il aurait dû avoir, puis nous répétons cette action jusqu'à avoir un taux d'erreur satisfaisant. Ce taux d'erreur est calculé via le carré de la différence entre le résultat attendu et le résultat obtenu. La version présentée à la soutenance comporte une erreur, l'initialisation des sommes n'est pas à 0 ce qui cause une erreur dans les résultats. Il est également envisagé de rajouter des `structs` qui représentent les neurones pour simplifier le code. Afin de tester le code, nous avons utilisé les différentes possibilités de la fonction XOR. Ci-dessous, une représentation des neurones, chaque trait correspondant au poids du neurone, c'est-à-dire la probabilité qu'il active le neurone suivant. Chaque neurone d'une couche est relié à celui de la suivante. C'est ainsi que modifier un poids en particulier peut avoir un impact plus ou moins important.



3.2.3 Sauvegarde des poids




Cette partie, partiellement implémentée, est cependant complètement claire dans nos esprits et seul la lecture du fichier manque à l'appel. Les poids, sauvegardés dans des matrices, sont écrits dans un fichier sous la forme suivante :

$$\{\{value_a, value_b\}, \{value_c, value_d\}\}$$

3.2.4 Progression

Le réseau de neurones va subir une importante révision d'ici la soutenance finale : En premier lieu, afin de respecter les conventions du langage C, même pourquoi pas le reprendre à zéro : nous allons l'entraîner à reconnaître des lettres et des chiffres afin qu'il fonctionne parfaitement.

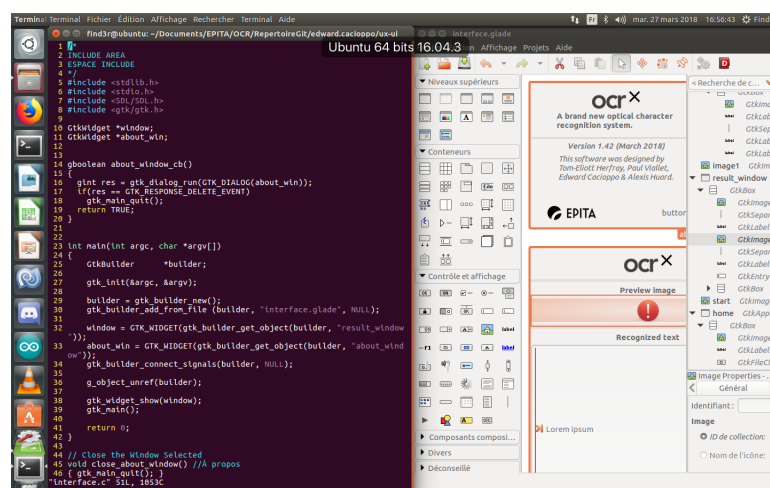
3.3 Expérience & interface utilisateur (UX/UI)

Pour cette première soutenance, le cahier des charges n'indiquait pas de présence explicite d'une interface graphique (GUI) dans notre projet. Cependant, nous avons déjà commencé à travailler sur un premier prototype utilisant la bibliothèque C *The GIMP Toolkit*¹ et l'outil de prototypage *Gnome Glade*, qui permet la compilation, via un programme conçu en C, d'une interface propre .XML et multi-plateforme (Linux , macOS , et bien sûr, Windows .

Pour compiler l'interface, nous avons dû fournir au compilateur GCC un fichier .glade (pour lister les éléments graphiques de l'interface, comme les boutons, les menus, images, etc.), un fichier .c qui exécute et affiche les différentes window de l'interface (dans notre cas : `about_window`, `main_window` et `result_window`), un fichier header .h (pour déclarer les fonctions) et un fichier Makefile (qui "mergera" avec les autres parties à la soutenance finale) ainsi que les images utilisées sur l'interface (logo, icônes, etc.).

L'interface actuelle d'OCR^x est fonctionnelle, cependant, nous avons souhaité concevoir nos parties séparément pour cette soutenance intermédiaire : Ainsi, l'interface n'appelle pas encore les fonctions de segmentation et du réseau de neurones. Il suffit d'implémenter la fonction de segmentation dans `interface.c` après avoir récupéré l'image via la fonction `load_img_cb()`. De même pour le résultat, que l'on obtient après avoir récupéré une *string* du réseau de neurones.

Pour la version finale de la GUI d'OCR^x, nous proposerons également, comme vous le verrez dans la section *Conclusion*, un correcteur orthographique implémenté directement dans l'interface, mais ça, c'est une autre histoire ! comme dirait Christophe Boulay



Environnement de travail *Gnome Glade* sur Ubuntu 16.04.

1. Version utilisée : GTK+ 3.22

3.3.1 Fenêtre *Main*

La fenêtre principale (celle que nous voyons lorsque nous compilons le programme) se compose d'un champ pour rechercher une image (png, jpeg, bitmap), d'un bouton "About" qui lance la fenêtre *About* (voir section ci-dessous) ainsi que du bouton "Start the OCR". Ce dernier appelle la fonction de segmentation qui traite l'image avant de la renvoyer au réseau de neurones.



Fenêtre principale du projet.

3.3.2 Fenêtre *About*

La fenêtre *About* résume en quelques mots le projet et crédite les auteurs, on y voit les logos d'EPITA et notre groupe de projet, Focal Studio, ainsi qu'un lien sur le bouton de droite vers notre site internet du projet : **ocrx.focalstudio.me**. On y remarque également la version actuelle du logiciel².

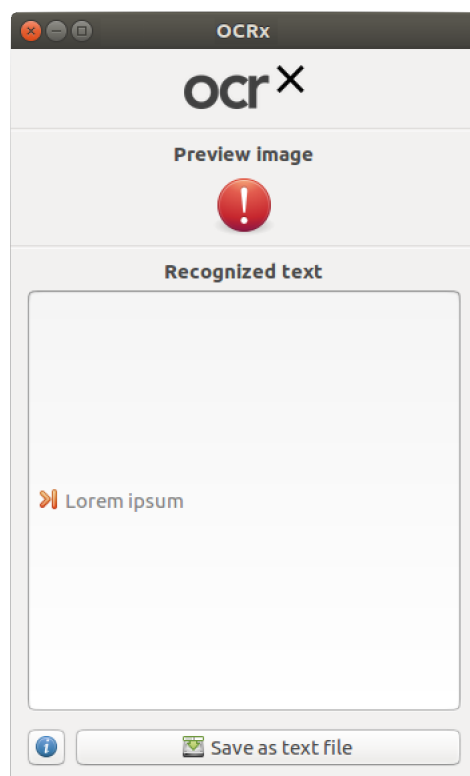


Fenêtre À propos du projet, avec les auteurs et un lien vers le site web.

2. Version actuelle : Beta 1.42 (oui, ce n'est pas très original, je sais...)

3.3.3 Fenêtre *Result*

La fenêtre `Result` est la plus importante (et la plus intéressante du projet, car c'est elle qui affiche le résultat de l'OCR en sortie dans une `text-box`. Le programme `interface.c` récupère la *string* en sortie du réseau de neurones, soit le texte détecté par OCR^x. Pour vérifier si le résultat est le bon, la fenêtre affiche au dessus un champ *Preview image* qui affiche l'image donnée en entrée du programme, dans la fenêtre `Main`. Le texte est modifiable en sortie si l'utilisateur constate des irrégularités sur le texte détectée.



Fenêtre du résultat retourné par le réseau de neurones d'OCR^x³.

Nota Bene : Les screenshots proviennent d'Ubuntu 16.04 mais le module Glade/GTK+ est déjà implémenté sur les machines à EPITA, donc pas de problème de compatibilité.

3. Lorsqu'une image n'est pas reconnue (mauvais format, pas de texte), la fenêtre renvoie un point d'exclamation en *Preview image* ainsi qu'une erreur dans le champ texte.

Conclusion

À l'issue de cette soutenance intermédiaire, l'essentiel de l'OCR a été réalisé. En effet, nous avons réalisé un réseau de neurones capable d'apprendre la fonction logique XOR, il est d'ailleurs plus avancé que cela, comme vous pouvez le voir dans la section *Réseau de neurones*. La segmentation permet également un bon traitement de l'image, la découpe sera opérationnelle à la soutenance finale.

L'interface est également bien avancée, il ne manque plus qu'à *linker* (relier) les programmes de segmentation et de réseau de neurones. L'interface finale permettra également d'exporter le texte détecté, ainsi que les différentes phases de segmentation de l'image, dans, respectivement, un fichier texte et un fichier image (selon la source fournie).

Nous incluons également, dans la version finale, quelques *easter-eggs*¹, ainsi qu'un correcteur orthographique et syntaxique au sein de la *text-box* générée par l'interface et le réseau de neurones.

Nous travaillons actuellement sur un site web qui sera disponible à l'adresse suivante : ocrx.focalstudio.me, et qui rassemblera les différents éléments de notre projet (source GitHub, auteurs, README, mode d'emploi fr/en, rapport de soutenance intermédiaire, rapport de soutenance finale, ...).

Merci de votre lecture. **Paix et prospérité** 🙌

1. Vous comprendrez que nous ne souhaitons pas *spoiler* ces fameux *easter-eggs* pour l'instant !



Proudly powered by \LaTeX