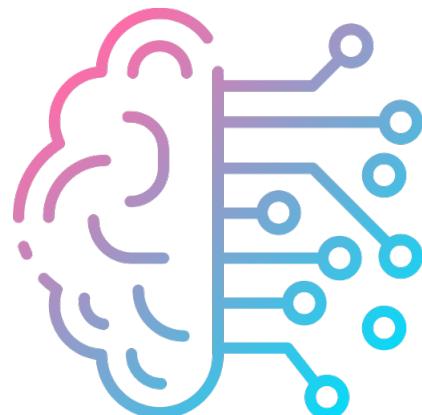

Compte-rendu de projet

RECONNAISSANCE DE SIGNES PAR INTELLIGENCE ARTIFICIELLE



Auteurs:

Thomas Tabuteau
Thomas Lépine
Ghassene Khecharem

Tuteur:

Mehdi Lhommeau

4eme année - SAGI

15 mai 2020

Sommaire

Introduction	2
Partie I : Introduction aux réseaux de neurones	3
Qu'est-ce qu'un réseau de neurones ?	3
Exemple de fonctionnement	4
Comment entraîner un réseau de neurones ?	6
La rétro-propagation	6
La différence entre les réseaux Feed Forward et Convolutional	7
Partie II : Conduite de projet	8
Planification et gestion du temps	8
Mise en place de l'environnement	8
Recherche et exploration	9
Exemple d'entraînement	9
Reconnaissance faciale	9
Reconnaissance de gestes	10
Partie III : Résultats et pistes d'amélioration	11
Résultats du projet d'entraînement	11
Résultats sur la reconnaissance faciale	13
Résultats de la reconnaissance de signes	16
Pistes d'améliorations	18
Partie IV : Travail de recherche	19
CONCLUSIONS	22
Conclusion générale	22
Conclusions personnelles	23
ANNEXES	24
Annexe 1	24
Annexe 2	25
Annexe 3	27
Annexe 4	28
Annexe 5	30
Sitographie	34
Résumé	36
Abstract	36

Introduction

Ce rapport fait suite à notre projet de fin de 4ème année proposé par monsieur Lhommeau. Il a consisté en la création d'un système de reconnaissance faciale basé sur un réseau de neurones. Ce projet permet donc de nous faire découvrir et comprendre le domaine de l'intelligence artificielle que l'on peut rencontrer absolument partout aujourd'hui. On peut par exemple retrouver des applications dans la reconnaissance visuelle (voitures autonomes, reconnaissance d'objets/visages, ...), la médecine (diagnostics, détection de cancers/tumeurs, ...), la sécurité, l'art, la traduction, etc.

Le but final de ce projet était de réguler l'accès à une salle grâce à la reconnaissance des personnes voulant entrer. Ce système devant être embarqué, nous avons eu comme contrainte de développer cet outil sur une Raspberry Pi, un ordinateur miniature à la puissance limitée. Pour compenser ce manque de capacité de calcul, une clé "Neural Network Stick" développée par Intel nous a été fournie, c'est elle qui s'occupe des calculs lourds liés au fonctionnement des réseaux de neurones.

Cependant, suite à plusieurs semaines de travail nous nous sommes rendus compte que le résultat demandé ne pourrait pas être atteint, la reconnaissance faciale demandant des connaissances, des données d'entraînement et une expertise relativement élevée dans le domaine de l'intelligence artificielle. Nous avons alors un peu simplifié le sujet, passant de la reconnaissance faciale à la reconnaissance de signe fait avec les mains formant un code pour entrer dans la pièce. Nous avons essayé lors du développement du projet de rendre le code le plus générique possible afin de pouvoir basculer à une reconnaissance faciale lorsqu'un réseau de neurones sera développé par exemple.



Figure 1: Image humoristique résumant l'intelligence artificielle

Partie I : Introduction aux réseaux de neurones

Avant toute chose nous devons expliquer ce qu'est un réseau de neurones et son fonctionnement, sans quoi la suite de ce rapport paraîtrait inutilement compliquée. Tout d'abord nous verrons ce qu'est un réseau de neurones, comment il fonctionne et comment il apprend, puis nous découvrirons les différents types de réseaux et leur utilité dans notre projet.

Qu'est-ce qu'un réseau de neurones ?

Un réseau de neurones peut être visualisé comme une sorte de "cerveau numérique", c'est une structure composée de neurones virtuels interconnectés afin de simuler le fonctionnement d'un cerveau biologique. Il faut visualiser l'organisation de ces neurones comme des couches, chaque couche est composée de plusieurs neurones et les neurones d'une couche seront reliés à ceux de la couche suivante comme on peut le voir sur la figure 2.

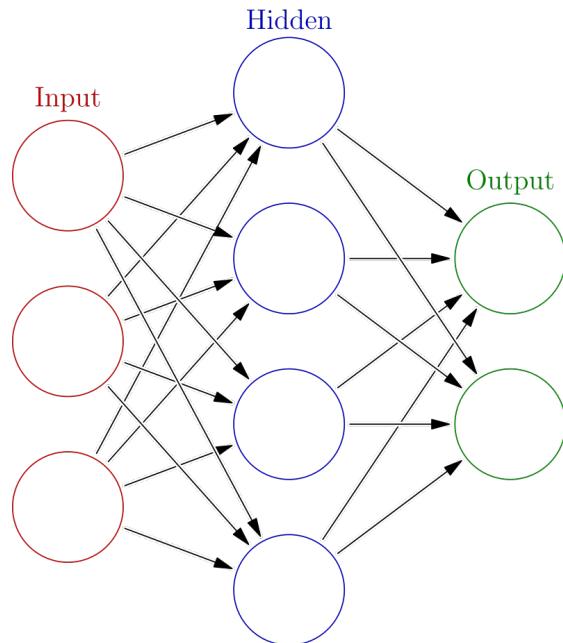


Figure 2: Réseau de neurone à 3 couches

Un neurone numérique fonctionne comme un neurone biologique, son activation sera déterminée par l'activation des neurones précédents. L'influence de ceux-ci sera fonction de la puissance du lien entre ces deux neurones, plus le lien sera fort, plus le neurone aura une influence sur l'activation de celui auquel il est connecté. Ainsi, les couches de neurones s'influencent les unes à la suite des autres pour au final arriver à l'activation de la dernière couche, celle-ci correspond à la sortie de notre système et c'est elle qui fourni le résultat calculé par le réseau de neurones. De façons plus précise et mathématique, l'activation a_1 d'un neurone dépendra des activations ou non des neurones précédents a_0 pondérés par le poids donné au lien qui les unis W , le tout est alors passé à une fonction d'activation σ qui donnera l'activation du neurone. Un biais b peut être appliqué afin de rendre le neurone plus facilement ou difficilement activable.

$$a_1 = \sigma(Wa_0 + b)$$

Exemple de fonctionnement

Maintenant que nous savons comment réagit chaque neurone à l'activation des neurones le précédent, nous pouvons voir le fonctionnement global d'un réseau de neurones et comment il arrive à "prendre une décision" par rapport à l'entrée qui lui est fournie. Pour cela nous allons prendre un réseau très simple composé d'une couche d'entrée à deux neurones, d'une couche cachée à trois neurones et d'une sortie à deux neurones. La fonction d'activation ReL (Rectified Linear) sera utilisée pour sa simplicité, nous pouvons voir sur la figure ci-dessous qu'elle sera égale à 0 en cas d'entrée négative et qu'elle la copiera sinon, sa fonction est $f(x) = \max(0, x)$.

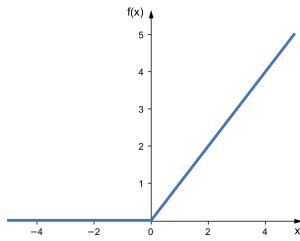


Figure 3: Courbe de la fonction d'activation ReL

Nous n'utiliseront pas de biais sur les neurones afin de simplifier cet exemple. Le poids des connexions sera noté W_{iNjk} où i est le numéro de la connexion du neurone N_{jk} , ici j sera le numéro de la couche et k le numéro du neurone dans cette couche. Le réseau est représenté graphiquement en figure 4, nous avons fixé ses entrées à $N_{11} = 0.5$ et $N_{12} = 0.2$ et nous allons voir comment cette entrée se propage couche par couche dans notre réseau et la sortie qu'elle génère.

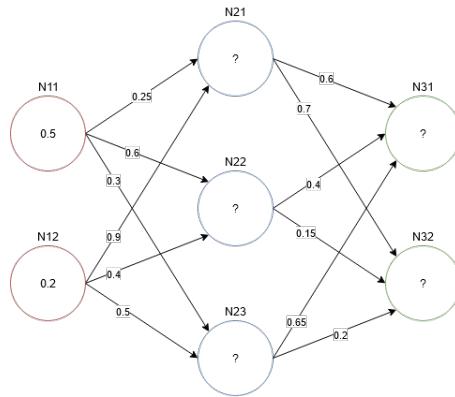


Figure 4: Réseau de neurones (exemple)

Prenons l'activation du neurone N_{21} comme exemple, celle-ci sera obtenue en faisant passer la somme pondérée des neurones N_{11} et N_{12} dans la fonction d'activation ReL, cela nous donne :

$$N_{21} = \text{ReLU}(N_{11} * W_{1N11} + N_{12} * W_{1N12})$$

$$N_{21} = \max(0, 0.5 * 0.25 + 0.2 * 0.9)$$

$$N_{21} = 0.305$$

La même opération est répétée pour les neurones N_{22} et N_{23} , nous obtenons alors:

$$N_{22} = 0.38$$

$$N_{23} = 0.25$$

Notre réseau de neurones ressemble maintenant à la figure ci-dessous. La couche cachée a été activée en fonction des données de la couche d'entrée et elle va à son tour propager ces données vers la couche de sortie.

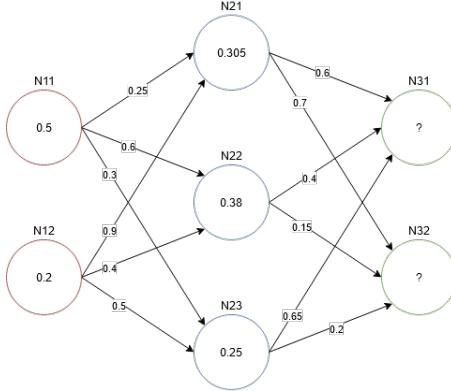


Figure 5: Réseau de neurones

La même opération que pour la couche cachée est répétée pour la couche de sortie. La fonction d'activation utilisée pour cette couche est extrêmement importante, car c'est elle qui va mettre en forme la "réponse" de notre réseau. Une fonction sigmoïde ou Softmax sera plutôt utilisée en cas de multiples réponses vrai possible (par exemple chaque neurone de sortie représente la présence d'un objet dans une scène, il peut y avoir plusieurs objets différents) alors qu'une fonction linéaire comme la ReL sera utilisée lorsque le réseau sert à faire de la régression. Pour cet exemple nous allons garder la fonction ReL comme fonction d'activation pour notre sortie.

$$N_{31} = \text{ReLU}(N_{21} * W_{1N21} + N_{22} * W_{1N22} + N_{23} * W_{1N23})$$

$$N_{31} = \max(0, 0.305 * 0.6 + 0.38 * 0.4 + 0.25 * 0.65)$$

$$N_{31} = 0.4975$$

$$N_{32} = \text{ReLU}(N_{21} * W_{2N21} + N_{22} * W_{2N22} + N_{23} * W_{2N23})$$

$$N_{32} = \max(0, 0.305 * 0.7 + 0.38 * 0.15 + 0.25 * 0.2)$$

$$N_{32} = 0.3205$$

Nous obtenons alors la sortie de notre réseau de neurones, $N_{31} = 0.4975$ et $N_{32} = 0.3205$, ces valeurs sont alors interprétées par le reste du programme utilisant ce réseau, elles n'ont pas vraiment de sens tant qu'elles ne sont pas interprétées.

Comment entraîner un réseau de neurones ?

Maintenant que nous savons comment fonctionne un réseau de neurones, nous pouvons voir plus en détail comment celui-ci est entraîné à faire les tâches qui lui sont destinées. Avant toutes choses, il faut bien comprendre qu'un réseau de neurones ne sera entraîné à faire qu'une seule chose bien précise, que ce soit, reconnaître des visages, générer des recommandations ou même restaurer des vidéos. Bien qu'un réseau puisse être adapté à faire des tâches proches de son entraînement initial (par exemple un réseau entraîné à reconnaître des objets pourrait être adapté pour reconnaître le langage des signes), il sera impossible de lui faire faire quelque chose trop éloigné de son entraînement.

L'entraînement d'un réseau de neurones va se faire grâce à des banques de données déjà triées et catégorisées. Ces données vont alors être analysées par le réseau de neurones. Les réponses obtenues vont alors être comparées à la réalité, si celles-ci diffèrent, les poids des liens entre les neurones vont être modifiés afin d'avoir une réponse plus proche de ce que l'on souhaite. On appelle cela la "rétro-propagation". Cela est répété avec une grande quantité de données jusqu'à ce que le réseau puisse fournir les réponses attendues, même sur des données qu'il n'avait jamais analysées. On appelle cela la phase d'apprentissage.

La rétro-propagation

Cette phase d'apprentissage va utiliser l'algorithme de rétro-propagation afin d'entraîner le réseau et de le faire converger vers le fonctionnement souhaité. Cet algorithme va utiliser la différence entre la sortie du réseau et la sortie voulue (aussi appelé le coût) pour modifier les poids et biais des neurones composants le réseau. Le nom de rétro-propagation qui a été donné à cet algorithme est représentatif de son fonctionnement, nous allons propager l'erreur de la dernière couche du réseau à la première en ajustant celui-ci au passage.

Prenons comme exemple le simple réseau à 2 couches ci-dessous, il possède 3 neurones d'entrées et 2 neurones de sortie. Nous allons voir comment fonctionne la rétro-propagation sur ce réseau.

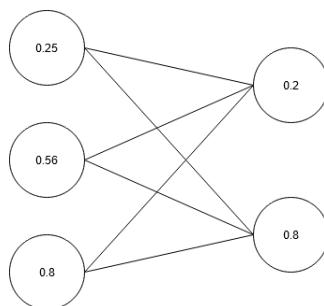


Figure 6: Simple réseau à 2 couches

Imaginons que l'entrée du réseau [0.25, 0.56, 0.8] est sensée résulter en cette sortie : [0.5, 0.5]. Nous allons tout d'abord nous concentrer sur le premier neurone de la couche de sortie, sa valeur est différente de la valeur attendue. Il y a une différence de +0.3 entre les deux. La rétro-propagation va chercher à faire tendre cette différence vers 0 en modifiant les valeurs des neurones de la couche précédente. Un neurone connecté avec un poids faible aura alors une faible influence sur la valeur du neurone actuel et sera moins modifié qu'un neurone ayant une forte connexion (un poids plus important). En répétant cette opération avec les autres neurones de cette couche nous obtenons les valeurs souhaitées des neurones de la couche précédente. Tout cela est alors répété jusqu'à atteindre la première couche du réseau, l'erreur est propagée jusqu'à l'entrée pour que le réseau ait ainsi un comportement plus proche de celui attendu.

La différence entre les réseaux Feed Forward et Convolutional

Lors de ce projet nous avons été amené à utiliser deux types de réseaux de neurones :

- les réseaux Deep Feed Forward (DFF)
- les réseaux Deep Convolutional (DCN)

En effet il existe beaucoup de types de réseaux de neurones et chacun sont plus ou moins adaptés à certaines tâches.

> Les réseaux Deep Feed Forward

Les réseaux de types DFF sont parmi les plus basiques, ils sont composés d'une couche d'entrée, d'une couche de sortie et de plusieurs couches cachées complètement connectées. Nous avons utilisé ce genre de réseaux pendant notre phase de recherche et d'apprentissage. Ils sont relativement simples à créer et à comprendre, mais ils ont une utilité très limitée en traitement d'images, nous les avons donc seulement utilisés comme outils pédagogiques afin d'apprendre à utiliser les librairies telles que Keras ou la clé Intel qui allait gérer les calculs lourds.

> Les réseaux Convolutionnels

Les réseaux convolutionnels profonds sont composés d'une couche d'entrée, suivie de couches convolutionnelles, puis ils se finissent par des couches cachées et une couche de sortie. C'est ce type de réseaux que nous avons utilisé dans notre projet afin d'automatiser la reconnaissance d'images. Une des limites de ces réseaux est qu'ils demandent énormément de ressources et de données afin d'entraîner convenablement les couches convolutionnelles. Heureusement, celles-ci peuvent être entraînées séparément des couches cachées, nous avons donc pu utiliser des couches convolutionnelles pré-entraînées et les ajouter à notre propre réseau. Ces couches convolutionnelles vont avoir comme tâche d'extraire les caractéristiques de l'image analysée, permettant de ne fournir les informations importantes à nos couches cachées, réduisant ainsi le temps d'analyse de chaque image.

Partie II : Conduite de projet

Planification et gestion du temps

Avant de commencer ce projet, on a discuté entre nous pour estimer la répartition du temps pour ses différentes parties, et dans un premier temps on a fait les estimations suivantes :

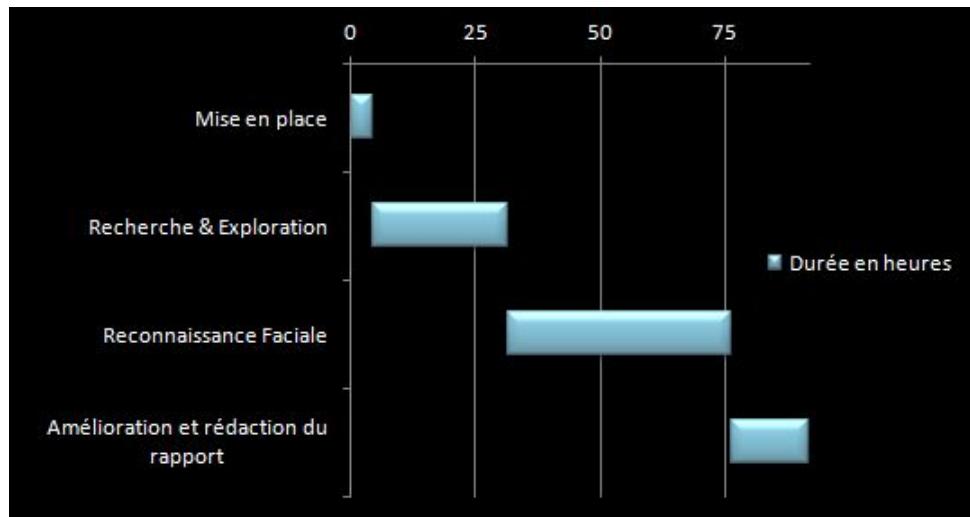


Figure 7: Diagramme de Gantt initial

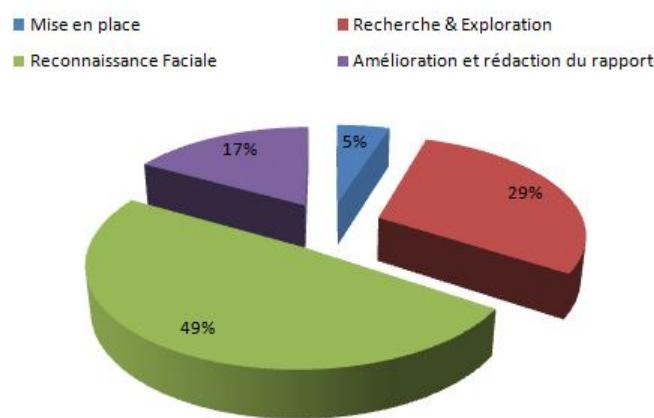


Figure 8: Répartition du temps initiale

Mise en place de l'environnement

Partant du matériel fourni on a commencé par :

L'installation du système d'exploitation de la carte Raspberry Pi 4.

L'installation des packages essentiels pour le fonctionnement de la caméra et de la clé "Neural Network Stick" de Intel.

Recherche et exploration

En nous basant sur les initiations qu'on a vues au cours de la conférence de M. Stephane Canu, nous avons entamé des recherches pour élaborer une idée sur leurs structures et leur fonctionnement où on a vu des modèles d'abstraction, des explications et des exemples. Les recherches ont été faites sur github, Intel AI, Google AI et YouTube principalement.

Au fur et à mesure que l'on avance sur le projet, il s'est avéré qu'on a largement sous-estimé la partie de la recherche qu'on a estimé initialement à 27 heures de travail (l'équivalent de 3 journées) mais on a dû y passer une journée de plus, ce qui a augmenté sa durée à 35,5 heures.

Exemple d'entraînement

Après la recherche et d'après ce qu'on a vu, on a estimé qu'il serait bénéfique de nous entraîner sur un exemple plus simple que la reconnaissance faciale pour mettre en pratique les informations acquises par la recherche et l'exploration et bien manipuler les techniques des réseaux de neurones afin de gagner du temps sur la partie suivante et opérer d'une manière plus efficace. Enfin, on a choisi de prendre un réseau de neurones qui aide à la distinction des chiffres à partir de photos en entrée comme exemple d'entraînement.

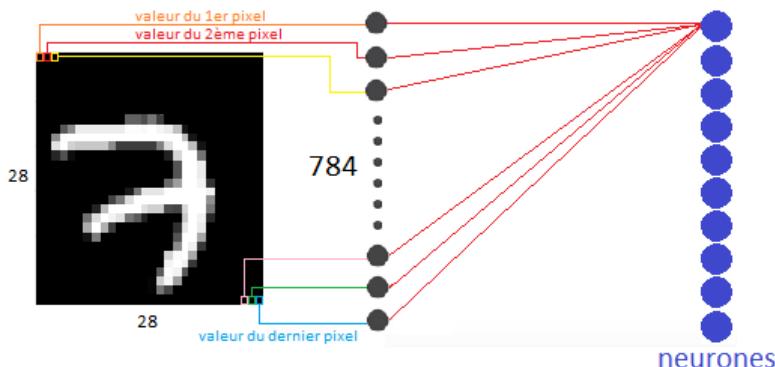


Figure 9: Reconnaissance de chiffres

Grâce à cet exemple, on a su d'une part, pratiquer de près le savoir qu'on a acquis durant la recherche faite auparavant, et d'autre part, manipuler la clé "Neural network stick" et assurer une exécution plus rapide du programme par cette clé.

De la conception du modèle du réseau au testing passant par le développement et l'exécution, on a pu élaborer une idée générale sur le travail qu'on aura à faire pour la reconnaissance faciale. Cet exemple d'entraînement nous a coûté une journée de travail ce qui équivaut à environ 9 heures.

Reconnaissance faciale

Après avoir fini l'exemple de reconnaissance de chiffres on a finalement entamé la partie de la reconnaissance faciale. Nous y avons passé en cumulé environ 25 heures. Nous avons néanmoins dû prendre la décision avec notre tuteur de prendre le sujet vers une autre direction qui est la reconnaissance de gestes.

Reconnaissance de gestes

Cette partie nous a pris finalement deux semaines de travail ce qui équivaut à environ 18 heures entre développement et testing. Le temps passé peut paraître court, ceci grâce au fait que nous avons pu réutiliser des connaissances et des lignes de codes et développés durant les phases précédentes.

Tous ces changements et imprévus nous ont conduit à une répartition différentes du temps de celle estimée au début et d'où aussi à un diagramme de Gantt différent. Ci-dessous les résultats finaux :

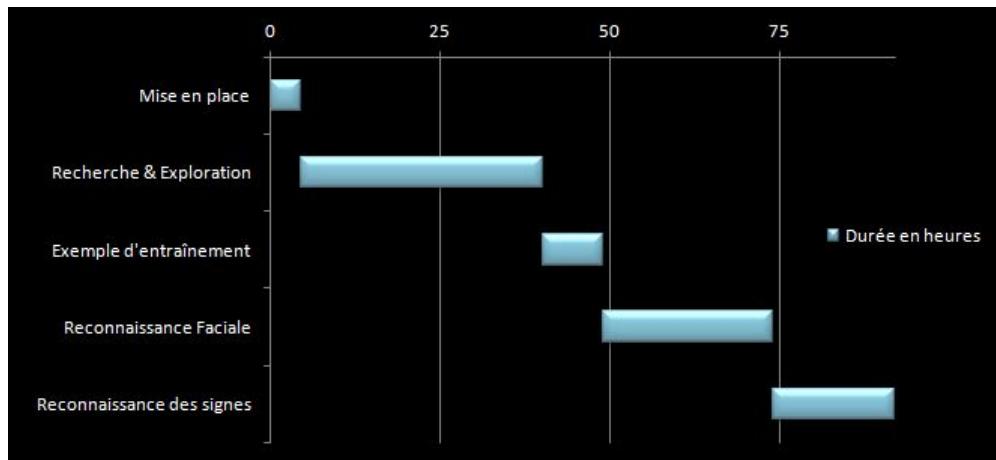


Figure 10: Diagramme de Gantt final

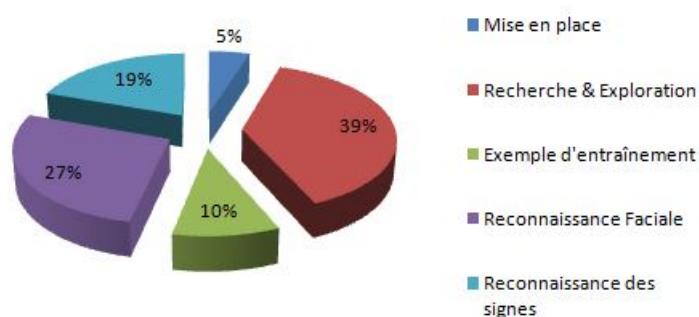


Figure 11: Répartition du temps finale

Partie III : Résultats et pistes d'amélioration

Résultats du projet d'entraînement

Avant de commencer à s'intéresser aux résultats que nous avons obtenues au final, nous allons tout d'abord regarder ceux obtenus lors de notre appropriation du sujet.

Dans un premier temps, nous nous sommes entraînés, comme indiqué dans la partie précédente, sur la reconnaissance de chiffre manuscrit.



Figure 12: Exemple de chiffres à reconnaître

Le but était de réussir à s'approprier le matériel, mais également d'apprendre à réaliser notre premier réseau de neurones. Pour ce faire, nous avons récupérer une banque de données de chiffres (mnist), puis nous avons créé l'architecture de notre réseau de neurones. L'entraînement de l'arbre se fait avec 60000 images et les tests avec 10000 images pour vérifier la performance de notre réseau. Pour établir la "meilleure" architecture de l'arbre, nous avons réaliser plusieurs tests avec différents paramètres (tel que les modes d'activation, le nombre de couches, le nombre de neurones profonds par couche, ...), afin d'obtenir les meilleurs résultats possibles. Notre réseau se constitue ainsi d'une couche cachée de 300 neurones, de 784 neurones pour l'entrée (784 pixels constituants l'image) et de 10 sorties (les 10 résultats possibles). Le taux de précision de ce modèle avoisine les 98%. Chaque neurone est connecté à tous les autres neurones de la couche suivante avec un poids différent (ajusté lors des multiples entraînements).

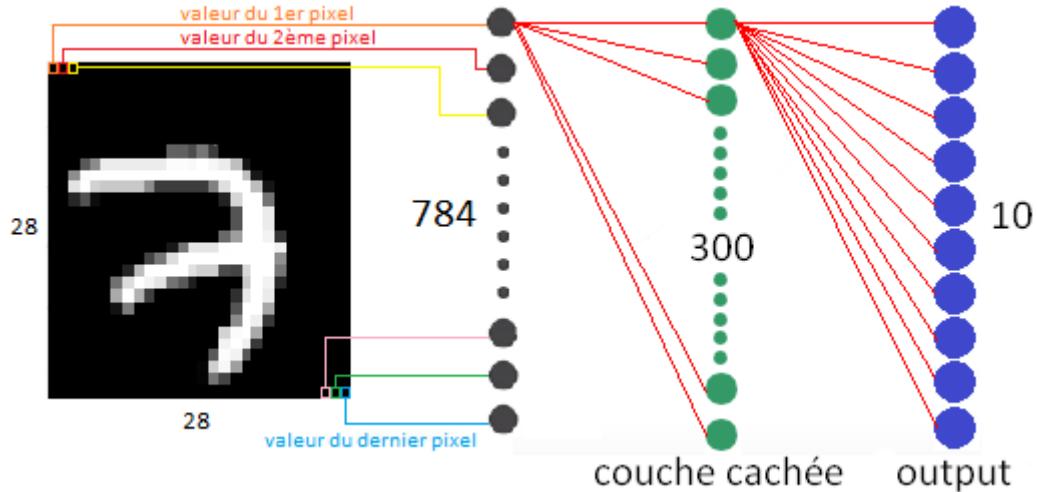


Figure 13: Dessin de l'architecture pour l'exemple de reconnaissance de chiffres

Cependant, nous avons testé notre intelligence artificielle sur des chiffres que nous avons créés nous-même, nous remarquons un problème d'identification sur le chiffre 3 par exemple, qui est reconnu comme un 8. En effet, un réseau de neurones comme celui-ci récupère chaque pixel sur la première couche, et selon les différentes valeurs, le réseau essaie de déterminer de quel chiffre il s'agit. La forme du chiffre n'est pas regardé dans sa globalité, ce type de réseau n'est pas suffisamment robuste pour la reconnaissance faciale par exemple.

En résumé, ce premier réseau de neurones nous a permis d'apprendre les bases de l'intelligence artificielle, mais également de voir les difficultés que nous allons rencontrer sur des modèles plus complexes. L'utilisation d'un réseau de neurones profonds et non d'un réseaux neuronal convolutif (*convolutional deep neural networks*) nous limite à certaines applications. Pour la reconnaissance faciale, il semble donc évident que l'utilisation d'un réseau neuronal convolutif est indispensable, car ce type de réseau sera plus robuste et plus optimal qu'un réseau de neurones profonds.

Résultats sur la reconnaissance faciale

La plus grande partie du projet a porté sur la reconnaissance faciale. Après de nombreuses recherches, nous avons décomposé le projet en plusieurs parties. Tout d'abord, pour entraîner notre futur réseau de neurones, nous devions récupérer nos visages, puis choisir le modèle convolutif que nous allions utiliser, enfin, nous devions réussir à labelliser les photos afin de pouvoir reconnaître par la suite. (La labellisation sera expliquée dans la partie suivante avec la reconnaissance de gestes.)

> Reconnaissance de visages

Le premier réseau de neurones élaboré pour cette partie est la reconnaissance de visage. Notre première piste pour la reconnaissance faciale en temps réel était d'extraire les visages présents d'une vidéo pour ensuite les reconnaître grâce à un réseau de neurones. Nous sommes donc partie sur la piste d'une intelligence artificielle permettant d'extraire les visages. Le but de cette intelligence n'est pas de reconnaître une personne, mais de réussir à reconnaître un visage grâce aux caractéristiques qui le différencient d'un autre objet d'une photo. Après du recul, cela n'était pas indispensable, car les réseaux de neurones convolutifs sont suffisamment robustes pour permettre l'extraction des visages directement lors de la phase de caractérisation des objets composants l'image. Cependant, nous n'avions trouvés aucun modèle déjà entraîné le permettant à ce stade du projet, nous pensions donc qu'il est tout de même important de parler de cette partie vu qu'elle a représenté plusieurs heures de travail. Le code et le lien de téléchargement des archives sont présents en annexe 1 (*page 24*). Pour illustrer le résultat, voici deux exemples d'application :

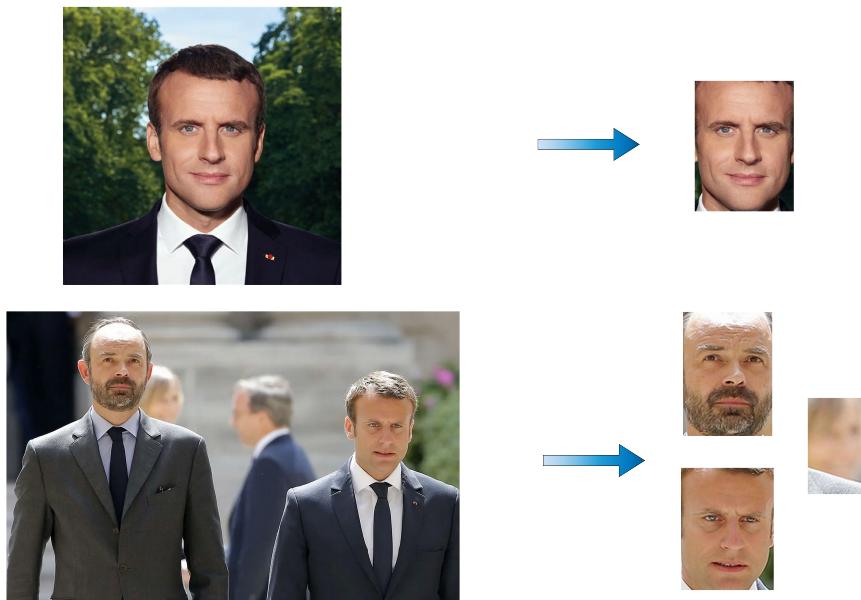


Figure 14: Exemples de la reconnaissance et extraction de visages

Par la suite, après avoir entraîné notre propre réseau de neurones avec nos visages extraits d'une vidéo où nous avons pris soin de le montrer sous différents angles, nous nous sommes rendus compte que cela n'était pas une bonne approche du problème. Le réseau de neurones n'était pas suffisamment flexible sur les changements de fonds sur les divers tests réalisés. Nous avons donc abandonné cette piste pour nous tourner ensuite vers la recherche de réseaux de neurones convolutifs.

> Modèles de réseaux de neurones convolutifs pour la reconnaissance faciale

Les réseaux de neurones convolutifs permettent de faire sortir certaines caractéristiques de l'image (featuring learning). Il vient ensuite une couche de neurones profonds permettant la classification finale. Ce fonctionnement est plus lourd que ce que nous avons vu précédemment, cependant, l'intelligence artificielle qui en résulte est bien plus robuste et plus fiable.

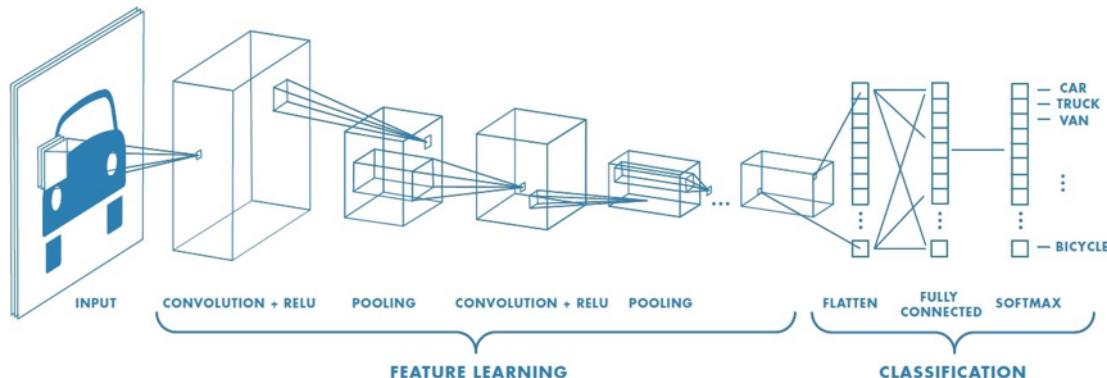


Figure 15: Schéma du fonctionnement d'un réseau de neurones convolutifs

La convolution consiste à appliquer un filtre sur l'image. Pour cela un kernel se déplace sur la totalité de l'image d'entrée et agit comme un filtre afin de produire une image en sortie. Cela permet ainsi de détecter des formes particulières (contours, trous, formes, ...)

Pour le projet, nous nous sommes intéressé à deux types de réseaux de neurones convolutifs : InceptionV3 et MobilNet. Les deux réseaux ont un fonctionnement et des performances différentes. MobilNet est aujourd'hui très présent dans les systèmes embarqués.

La principale différence entre ces deux types de réseaux est que MobilNet utilise la convolution séparable en profondeur alors que InceptionV3 utilise la convolution standard. Il en résulte un nombre de paramètres moins importants dans MobilNet par rapport à InceptionV3. Nous pouvons également noté que la convolution séparable en profondeur a été proposée par Google depuis avril 2017.

Dans une convolution standard, le filtre fonctionne sur les M canaux de l'image d'entrée dans son ensemble et produit N "cartes" de caractéristiques, c'est-à-dire que la multiplication matricielle entre l'entrée et le filtre est multidimensionnelle. Dans une convolution standard, chaque élément d'entrée se multipliera avec un filtre et enfin, après la multiplication, les "cartes" de caractéristiques seront ajoutées aux N "cartes" de caractéristiques de sortie. Prenons par exemple, un cube de taille $D_k \times D_k \times M$, puis, dans une convolution standard, chaque élément du cube se multipliera avec l'élément correspondant dans la matrice de caractéristiques d'entrée.

Cependant, dans une convolution séparable en profondeur, les M filtres à canal unique fonctionneront sur un seul cube dans la caractéristique d'entrée et une fois que les M filtres de sortie sont obtenus, un filtre ponctuel de taille $1 \times 1 \times M$ fonctionnera sur celui-ci pour donner N cartes de caractéristiques de sortie. Ceci peut être compris à partir de la figure ci-dessous, tirée du document MobileNet :

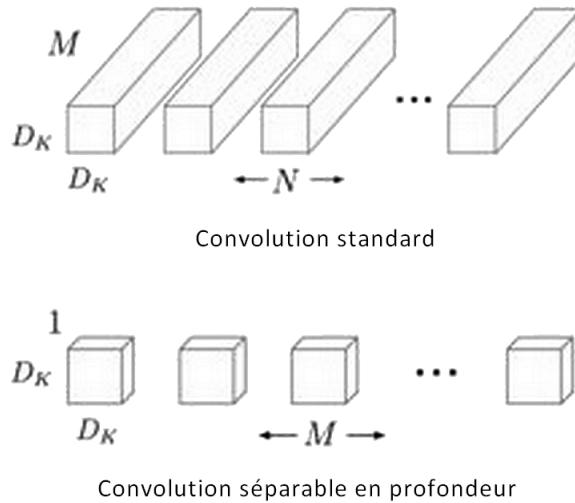


Figure 16: Schéma du fonctionnement des deux réseaux de convolutions InceptionV3 et MobilNet

Après de nombreux tests avec ces deux types de réseaux et avec divers visages, nous pensons qu'ici aussi, ce type de fonctionnement ne répond pas au problème dû à la grande imprécision de l'intelligence artificielle. Nous avons réalisé les tests avec nos visages, mais également avec le visage de plusieurs stars hollywoodiennes grâce à des banques d'images trouvables sur internet. Notre intelligence artificielle n'est pas suffisamment robuste, il y a trop de tests qui concluent à un résultat erroné (mauvaise reconnaissance). Cela vient de deux facteurs : l'entraînement et le type du réseau.

Pour les entraînements réalisés, nous étions limités par le nombre d'images, mais également par la puissance des ordinateurs. Un entraînement avec une cinquantaine d'images pour un visage prenait environ 10 minutes. De plus, nous pensons qu'il fallait plus d'images que cela et d'une plus grande variété pour avoir un résultat satisfaisant. L'autre principale raison était le type de réseau. Les deux modèles de réseaux de neurones convolutifs sont plus performants pour les reconnaissances d'objets sur une image que pour une reconnaissance faciale. Les visages sont reconnaissables uniquement grâce à un très grand nombre de caractéristiques trop précises pour ces types de réseaux.

Après ces nombreux tests et de nombreuses recherches, nous avons donc conclu que la reconnaissance faciale serait trop difficile à mettre en place dû aux nombreuses contraintes expliquées. C'est donc à la suite d'une discussion avec notre tuteur de projet Mr Mehdi Lhommeau, que nous avons pris la décision de réaliser une intelligence artificielle sur la reconnaissance de gestes/signes. En effet, cela permet de répondre à la problématique de réaliser une IA pour un système embarqué ayant le but de restreindre l'accès à une pièce grâce à une caméra. Les utilisateurs devront donc réaliser une suite de signes avec leur main pour y déverrouiller l'accès.

Résultats de la reconnaissance de signes

Pour la reconnaissance des gestes/signes, nous nous sommes tout d'abord intéressés à une intelligence artificielle proposée par Google : Hand Tracking. Le réseau associé est disponible en open source. Cependant, après quelques tests, ce réseau est peu maniable, cela demanderait trop de travail par rapport au temps restant pour pouvoir l'ajuster à notre projet.

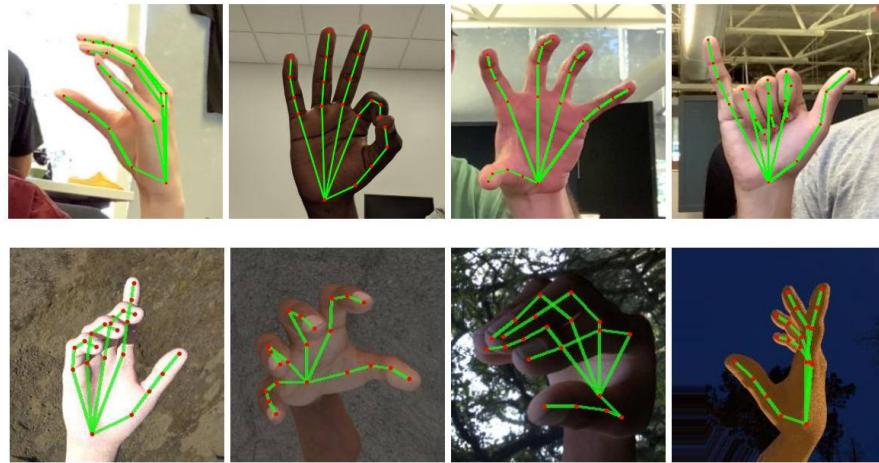


Figure 17: Hand Tracking développé par Google

Nous avons ensuite poursuivi nos recherches, et nous avons trouvé un réseau de neurones convolutifs pré-entraîné avec 1 millions d'images pour la reconnaissance de signes. Après plusieurs tests réalisés sur celui-ci, il nous a semblé satisfaisant. Nous avons donc commencé le développement de l'intégration de cette intelligence artificielle pour un programme utilisant la caméra Pi v2.1 et faisant la vérification du mot de passe.

Le main.py du programme développé est disponible en annexe 2 ([page 25](#)).

Les signes reconnus par le réseau de neurones sont disponibles en annexe 3 ([page 27](#)).

Le code se décompose en plusieurs parties :

- l'initialisation de la caméra et sa mise en route, pour avoir un affichage en "temps réel";
- récupération de l'image capturée par la caméra pour la transférer vers notre intelligence artificielle;
- analyse de l'image par le réseau de neurones;
- récupération du résultat et affichage de celui-ci;
- mise à jour du code proposé par l'utilisateur.

La caméra utilisée est la caméra Pi v2.1 de Raspberry. Grâce à la librairie picamera, nous pouvons paramétriser facilement l'image capturée. Nous avons essayé plusieurs paramétrages différents de manière à ce que l'image permette ensuite une identification plus simple. En modifiant les iso et la luminosité, nous pouvons rendre plus facile la reconnaissance des signes réalisés. Par la suite, nous récupérons l'image de la caméra que nous sauvegardons de manière temporaire grâce à une variable de type stream. L'image est redimensionnée de façon à pouvoir correspondre aux attentes du réseau de neurones (224 * 224). Cette image est ensuite classifiée si le taux de

ressemblance à un signe appris par l'IA est supérieur à 50%, sinon, l'image est soit non-reconnue (ne correspond à aucune correspondance), soit elle ressemble à un signe mais avec un taux de fiabilité trop faible. Dans ce cas, le signe reconnu est informé à l'utilisateur avec le pourcentage de correspondance pour que celui-ci ajuste sa main en conséquence. Lorsque qu'une image est reconnue par le réseau de neurones, le programme donne un label au résultat. Ce processus permet non seulement de permettre de faire des tests, mais également d'apporter une meilleure ergonomie auprès des utilisateurs en les informant sur le geste réalisé. La labellisation est réalisée car à un programme python contenant un tableau regroupant les signes qui peuvent être reconnus. Ce programme est disponible en annexe 4 (*page 28*). Le code est composé par l'utilisateur, une vérification sera effectuée une fois le nombre de gestes composants le code réalisé.

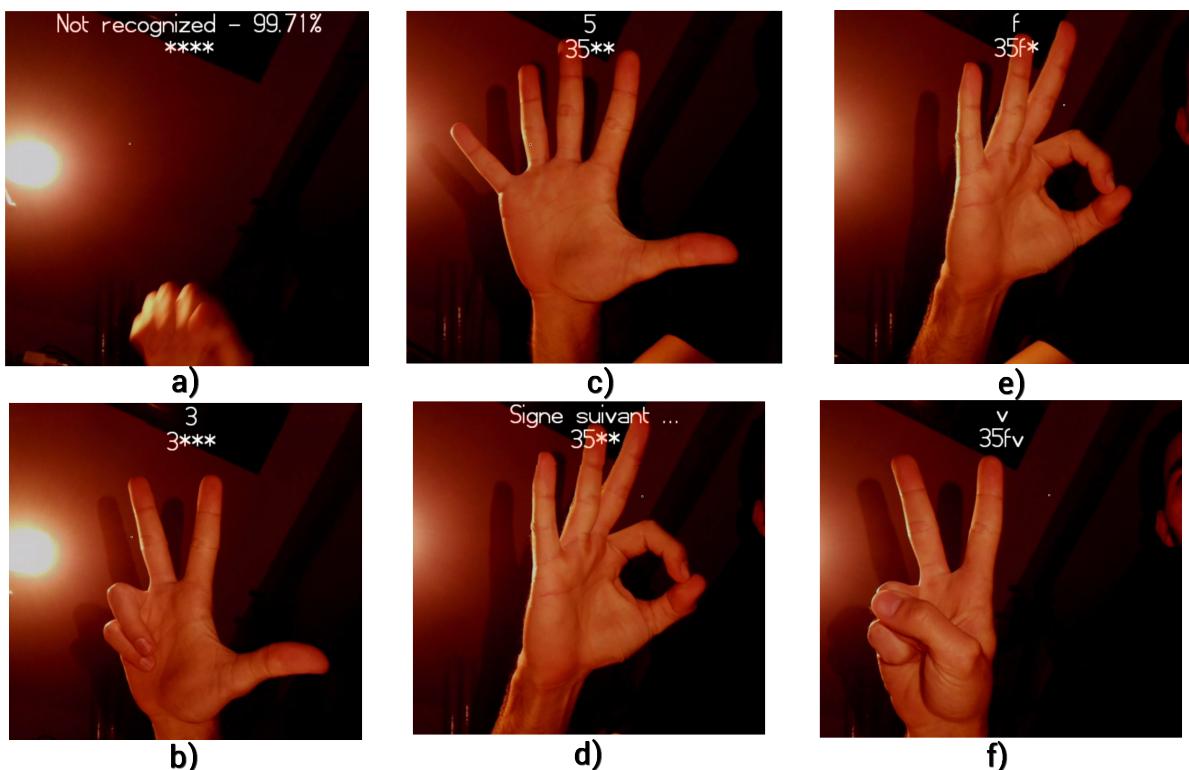


Figure 18: Exemple d'un résultat obtenu

La vidéo entière présentant un exemple de résultat est téléchargeable [ici](#).

Légende des photos :

- a) Aucun signe n'est reconnu,
- b) Le chiffre 3 est reconnu,
- c) Le chiffre 5 est reconnu,
- d) Après la reconnaissance d'un signe, il y a une phase où on ne cherche pas à reconnaître ce qui est visible sur la caméra, afin de laisser le temps à l'utilisateur de se mettre en position sans avoir des résultats erronés,
- e) Le signe f est reconnu,
- f) Le signe v est reconnu.

Le mot passe ici est alors : 35fv.

Pistes d'améliorations

Plusieurs points sur le projet auraient pu être améliorés. Tout d'abord au niveau de la caméra, le modèle utilisé pourrait être remplacé par un modèle plus performant, nous aurions pu peut-être avoir de meilleurs résultats lors de l'identification des signes. Le fait d'être limité à l'utilisation d'une carte Raspberry Pi est une contrainte à cause de ses performances plus faibles que certaines cartes programmables pour réaliser des systèmes embarqués. Néanmoins, on note tout de même que d'avoir la clé Neural Compute Stick en notre possession pour le projet, nous a grandement aidé et permis de réaliser des calculs beaucoup plus rapidement. Ensuite, il a été observé avec l'équipe, que la détection des signes était bien meilleure lorsque la main était exposée à une lumière avec un fond relativement foncé. Il pourrait donc être envisageable de réaliser un système avec une sorte de caisson sombre où des spots de lumière éclairent la main. La fabrication de cet objet permettrait également à l'utilisateur de réaliser le code à l'abri des regards.

Un autre point que nous n'avons absolument pas abordé dans le projet par manque de temps principalement est la cybersécurité. En effet, dans le cadre où l'utilisateur renseigne un mot de passe, il est important que la vérification et le stockage de celui-ci soit crypté et accessible difficilement. Pour des raisons de simplicité et parce que ce n'était pas l'objectif principal, le mot de passe est vérifié à l'heure actuelle en "dur" (directement sur le programme). Avec plus de temps, nous pourrions envisager de stocker ce mot de passe en crypté (SHA 256 par exemple), sur un serveur ou dans un dossier protégé de la carte Raspberry. Il faudrait également permettre au protocole de vérification d'envoyer les données en cryptées avec également un système empêchant le brut force dans le cas où la carte serait dérobée.



Partie IV : Travail de recherche

Dans le cadre de l'initiation à la recherche mise en place par Polytech Angers, il nous a été demandé d'analyser un article scientifique en rapport avec notre projet. Celui choisi par monsieur Lhommeau est l'article "Learning representations by back-propagating errors" paru dans le journal "Letters to Nature" le 9 octobre 1986. Celui-ci s'inscrit dans notre projet car il décrit une méthode d'apprentissage par rétro-propagation des erreurs qui est aujourd'hui utilisé par presque tous les réseaux de neurones modernes. L'article en question est disponible en annexe 5 ([page 30](#)).

Cet article porte sur l'amélioration de la phase d'apprentissage des réseaux de neurones en introduisant le concept de rétro-propagation. Ce concept vise à créer une règle permettant de modifier un réseau de neurones arbitrairement connecté afin de développer une structure résolvant une tâche spécifique. La rétro-propagation proposée ici va utiliser la différence entre la sortie souhaitée du réseau et la sortie obtenue pour une entrée donnée. Cette différence (aussi nommée erreur) est définie comme :

$$E = \frac{1}{2} * \sum_c \sum_j (y_{j,c} - d_{j,c})$$

c représente l'index d'une paire entrée/sortie, j l'index des neurones de sortie, y l'état du neurone de sortie et d l'état désiré. Cette erreur globale va être propagée de la couche de sortie à la couche d'entrée en utilisant l'algorithme du gradient afin de la faire converger vers un minimum local et ainsi amener le réseau au fonctionnement souhaité. L'auteur conclu alors par une comparaison à la méthode d'apprentissage des cerveaux biologiques. Il admet que la rétro-propagation n'est pas un modèle viable pour celle-ci, mais qu'il serait intéressant de chercher des méthodes plus biologiquement proches afin d'appliquer l'algorithme du gradient dans un réseau de neurones.

Extreme Learning Machine (ELM)

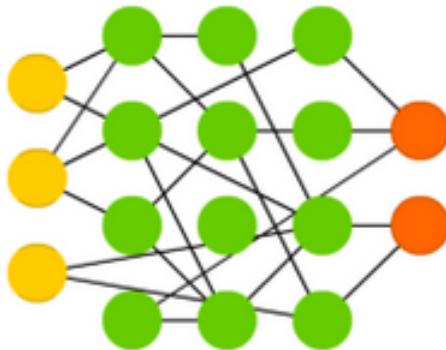


Figure 19: Réseau de neurones offrant l'apprentissage le plus simple d'après l'article

En résumé, la grande majorité des réseaux de neurones possède un algorithme d'entraînement qui consiste à modifier les poids synaptiques (Les paramètres de la fonction d'activation (*ou fonction de seuillage*) qui décrit le fonctionnement du neurone.) en fonction d'un jeu de données présenté en entrée du réseau. Un calcul est effectué, le neurone propage alors son nouvel état interne sur son axone (*Les liens reliés à celui-ci*) comme nous l'explique l'article. En d'autres termes, le but de cet entraînement est de permettre au réseau de neurones d'apprendre à partir d'exemples. L'entraînement réalisé est "efficace" si les pourcentages de reconnaissances à la fin sont proches des 100% sur les exemples d'entraînement. Mais tout l'intérêt des réseaux

de neurones réside dans leur capacité à généraliser à partir du jeu de test. Une fois la phase d'entraînement réalisé, le réseau de neurones peut être utilisé. Selon la somme pondérée arrivant à l'entrée du neurone, la sortie est calculée (*souvent compris dans l'intervalle [0,1] ou [-1,1]*).

Si on s'intéresse à un point de vue de logique mathématique, un réseau de neurones n'est alors pas associable au cadre de la logique déductive (à partir de connaissances établies on déduit des résultats), puisque la rétro-propagation réalise l'inverse : on part des résultats bien connus pour réussir à trouver le chemin de raisonnement, cela s'appelle un procédé par induction. La notion d'apprentissage recouvre deux réalités souvent traitées de façon successive :

- mémorisation : le fait d'assimiler sous une forme dense des exemples nombreux,
- généralisation : le fait d'être capable, grâce aux exemples appris, de traiter des exemples distincts, dont certains non rencontrés, mais relativement similaires.

Un élément que nous pensons manquant dans l'article est la notion de sur-entraînement ou surapprentissage. En effet, il arrive souvent que les exemples de la base d'apprentissage comportent des valeurs approximatives ou bruitées par exemple. Si on oblige le réseau à répondre de façon quasi parfaite à ces exemples, on peut obtenir un réseau qui est biaisé par des valeurs erronées. Cela peut aussi s'interpréter comme un apprentissage "par cœur" des données, une sorte de mémorisation point par point. Si l'on autorise un ensemble de fonctions d'apprentissage plus grand, par exemple l'ensemble des fonctions polynomiales à coefficients réels, il est possible de trouver un modèle décrivant parfaitement les données d'apprentissage (erreur d'apprentissage nulle). C'est le cas du polynôme d'interpolation de Lagrange ou les méthodes de régularisation comme le weight decay qui permettent également de limiter la spécialisation / sur-entraînement. De plus, pour éviter et détecter un surapprentissage, il existe une méthode simple, on sépare les données en deux sous-ensembles : l'ensemble d'apprentissage et l'ensemble de validation. Le premier sert à l'apprentissage pour faire évoluer le poids du réseau de neurones, et le second sert à l'évaluation de l'apprentissage. Tant que l'erreur obtenue sur le deuxième ensemble diminue, on peut continuer l'apprentissage, sinon on arrête.

En se re-focalisant sur l'article, dans les premiers paragraphes de celui-ci, nous pouvons lire une référence sur les procédés d'apprentissages effectués avant la rétro-propagation. Cependant, il est dommage de ne rentrer plus dans les détails ... En effet, une comparaison entre ce nouveau type d'algorithme et ses prédecesseurs aurait été la bienvenue, principalement sur la comparaison de performance en matière de fiabilité et de rapidité. De plus, il est évoqué plusieurs fois dans l'article un choix dans l'algorithme de rétro-propagation, celui de changer les poids après chaque cas d'entrée/sortie pour éviter d'avoir à stocker les dérivées. La mémoire est très limitée sur les ordinateurs lors de la parution de l'article en 1986. Qu'en est-il avec les ordinateurs d'aujourd'hui ? Nous pensons qu'il serait intéressant d'avoir les comparaisons sur les différentes méthodes avec les contraintes contemporaines moins réductrices. Ce point est plus soulevé par curiosité, car sur tout le reste, la note de recherche reste novatrice et très révolutionnaire dans le domaine du deep-learning et qui a bien vieilli avec le temps.

Dans le dernier paragraphe de l'article, un parallèle avec le cerveau humain est fait, il est écrit que ce modèle n'est pas viable/décalable pour un cerveau humain. Nous trouvons le parallèle intéressant et pas dénué de sens, cependant, nous pensons qu'en regardant avec plus de recul cela n'est pas forcément juste. En effet, nous nous sommes posés la question de savoir si ce n'était pas déjà le cas. C'est-à-dire, est-ce que finalement, un cerveau biologique n'apprend pas avec les erreurs commises et apprises pour ensuite ne pas les répéter, tout comme le fonctionnement d'un réseau de neurones par retro-propagation. Néanmoins, comme le disent très bien les auteurs, l'algorithme est simplifié pour des raisons de performances, ils suggèrent qu'il ne serait pas dénier de sens de chercher des méthodes plus proches d'un cerveau biologique pour faire l'algorithme du gradient dans les réseaux de neurones.

Cet article est le premier article scientifique posant la méthode de rétro-propagation. D'ailleurs les plus anciens articles universitaires disponibles sur le moteur de recherche Google Scholar concernant cette méthode datent au plus tôt de 1989, c'est-à-dire trois ans après la publication de l'article entre nos mains. Ceci dit, la nouveauté révélée ici aurait pu être qualifiée comme révolutionnaire en ce qui concerne les réseaux de neurones, car elle rapporte une vision du problème d'une perspective différente. Grâce aux démarches et aux exemples expliqués, les auteurs ont su ramener la méthode à un niveau d'abstraction assez convenable pour être comprise par les novices des réseaux de neurones.

CONCLUSIONS

Conclusion générale

En conclusion, ce projet a permis au groupe de travailler sur une notion plus qu'ancrée dans les problématiques de digitalisation actuelles : l'intelligence artificielle. En effet, cette notion et la notion de deep-learning présentent un enjeu majeur. De plus en plus d'applications les sollicitent.

Notre mission initiale était de réussir à développer une intelligence artificielle présente dans un système embarqué afin de réaliser une reconnaissance faciale. Cependant, à cause des nombreuses contraintes décrites dans le rapport, nous n'avons pas pu réaliser cette mission. Après plusieurs recherches effectuées, nous pensons que l'utilisation d'une intelligence artificielle pour une reconnaissance faciale via une caméra n'est pas ce qu'il y a de plus optimale. En effet, nous pensons qu'un système comme le "Face ID" développé par Apple répond plus à la problématique. La caméra TrueDepth capture des données faciales précises en projetant plus de 30 000 points invisibles, afin de créer une carte de profondeur et de capturer une image infrarouge d'un visage. Un réseau de neurones est utilisé pour permettre d'adapter la vérification de visage après un changement de l'apparence, comme le maquillage ou la pilosité faciale. Un système comme celui-ci est moins lourd en matière de ressources derrière et est plus fiable pour de la vérification de visage. Il a également l'avantage de pouvoir détecter un véritable visage d'une photo ou d'un masque en silicone. Cependant, la caméra utilisée est bien plus coûteuse.

La technologie de la reconnaissance faciale avec une "simple" caméra est néanmoins très largement utilisée pour des raisons de sécurité dans les lieux publics dans certains pays, afin de retrouver des criminels par exemple.

Une fois la reconnaissance faciale abandonnée, nous nous sommes tournés vers la reconnaissance de gestes/signes. Toutes les problématiques ne sont pas encore résolues aujourd'hui, puisqu'il y a un véritable enjeu de pouvoir par exemple, traduire en temps réel une personne muette parlant avec le langage des signes. Google a développé la technologie du Hand Tracking permettant, à l'aide d'un réseau de neurones, de modéliser les mains avec des points sur les articulations. La suite de ce projet, est de réaliser une intelligence artificielle permettant de reconnaître n'importe quel signe selon la position de ces points (positionnement de la main). Notre projet s'articule autour d'un réseau de neurones convolutifs pré-entraîné. Celui-ci est intégré à un programme développé en Python sur la Raspberry afin de détecter les gestes réalisés par un utilisateur devant la caméra en temps réel.

Nous sommes relativement satisfaits du résultat final, car il répond au cahier des charges. Le contrôle de l'accès à une salle par reconnaissances de signes permet de répondre à une problématique actuelle, portée par la crise sanitaire, de sécuriser l'accès à des lieux tout en évitant d'avoir un contact avec un objet (clavier numérique par exemple). Nous restons néanmoins déçus de ne pas avoir pu réussir à mener au bout le projet initial de reconnaissance faciale. Cet "échec" nous a tout même permis d'apprendre de nombreuses choses et de se rendre compte que l'intelligence artificielle a encore de belles années devant elle.

Conclusions personnelles

> Thomas Tabuteau

Ce projet m'a beaucoup appris en ce qui concerne les réseaux de neurones et l'intelligence artificielle. Ce sujet m'intéressait avant le projet et j'avais déjà expérimenté avec de la vision par ordinateur lors de mon stage l'année dernière, mais la complexité de ce sujet m'avait rebuté et sans introduction réelle je n'avais pas réussi à créer un réseau fonctionnel. La conférence organisée par Polytech et animée par Stéphane Canu a fait office d'introduction à ce concept et m'a permis de me sentir à l'aise lors de ce projet, même si beaucoup de recherches ont été requises tout au long de celui-ci.

> Thomas Lépine

Je n'ai vraiment pas été déçu par le projet, il a répondu à mes attentes en me permettant de découvrir et de mieux appréhender l'intelligence artificielle. En effet, les notions d'IA et de réseaux de neurones m'étaient peu connues auparavant. Ces notions peuvent paraître très complexes et difficilement accessibles dans un premier temps, cependant, à l'aide du projet j'ai pu m'apercevoir qu'au contraire, elles sont abordables. J'ai ressenti un véritable développement personnel et de mes compétences, puisque de nombreuses notions sont abordées telles que le développement, l'utilisation d'une Raspberry, l'utilisation de nombreuses bibliothèques et la recherche. Cette dernière notion était au cœur du projet. Nous avons été "obligés" de passer une grande partie de notre temps à réaliser des recherches sur l'intelligence artificielle et les différents types de réseaux de neurones avant de pouvoir les manipuler.

J'ai apprécié ce projet de groupe, ma seule déception a été de devoir changer le sujet initial. La reconnaissance faciale était trop complexe à mettre en place à cause de plusieurs contraintes tel que le matériel (puissance des ordinateurs, nombre de modèles d'entraînement, ...). La reconnaissance des signes a tout de même réussi à remplir la mission principale du projet : nous faire apprendre à utiliser une intelligence artificielle.

> Ghassene Khecharem

Cette expérience m'a tant apporté sur les deux plans, technique et personnel. Techniquement, j'ai eu la chance pour la première fois de travailler avec la carte Raspberry Pi 4 et la clé "Neural Network Stick" d'Intel comme j'ai su manipuler l'une des plus importantes techniques de l'intelligence artificielle et de l'informatique en général qui est les réseaux de neurones.

Sur le plan personnel, et en tant qu'étudiant étranger, c'est mon premier travail en groupe en France, et c'était vraiment une très belle expérience, de collaborer et de s'entraider avec Thomas et Thomas ça m'a fait énormément plaisir.

Certes, notre travail n'a pas atteint la totalité des objectifs techniques, mais les objectifs pédagogiques et pratiques ont été largement atteints.

ANNEXES

Annexe 1

> Code Python reconnaissance et extraction de visages

Toutes les archives sont téléchargeables [ici](#).

```

1 import os
2 from os import path as os_path
3 import shutil
4 import argparse
5
6 var_path = os_path.abspath(os_path.split(__file__)[0]) #Récupère le ré
      pertoire courant
7 path_ia = var_path + '/van/face_detection_retail_intel'
8 shutil.rmtree(var_path + '/result_cropped_faces') #Supprime le contenu du
      repertoire et celui-ci
9 os.makedirs(var_path + '/result_cropped_faces') #Recreer le repertoire
10 ACroper_path = '/ACroper'
11
12 def parse_args():
13     parser = argparse.ArgumentParser(description = 'Image face croper using
      Intel Neural Compute Stick 2.' + ' || Script developed by Thomas
      LEPINE')
14
15     parser.add_argument( '--INPUT_DIR', metavar = 'Directory use for the
      INPUT',
16                         type=str, default = var_path + ACroper_path,
17                         help = 'Choose your INPUT path (By default is : ' +
      var_path + ACroper_path + ')')
18
19     return parser
20
21 ARGS = parse_args().parse_args()
22 prnt = ARGS.prnt
23 Input_path = ARGS.INPUT_DIR
24
25 for Personn in os.listdir(Input_path):
26     final_path = var_path + '/result_cropped_faces/' + Personn
27     variable_de_nommage = 1
28     print(prnt)
29     if not os.path.exists(final_path):
30         os.makedirs(final_path) #Créer le dossier pour stocker les faces
      cropées s'il n'existe pas déjà
31
32     for image in os.listdir(var_path + ACroper_path + '/' + Personn):
33         result_temp_path = var_path + '/van/temp'
34
35         os.system('python ' + path_ia + '/face_detection_retail_0004.py' +
36                   ' --face_ir ' + path_ia + '/face-detection-adas-0001-fp16.
      xml -i ',
37                   + var_path + ACroper_path + '/' + Personn + '/' +
      image + ' --crop ' + result_temp_path + ' --show no')
38
39
40     for faceI in os.listdir(result_temp_path): #Toutes les faces
      extraites
41         os.system('mv ' + result_temp_path + '/' + faceI + ' ' +
      final_path + '/' + Personn + '_' + str(variable_de_nommage) + '.png')
50         variable_de_nommage += 1

```

Annexe 2

> Code Python (main.py) de la reconnaissance de signes

Toutes les archives sont téléchargeables [ici](#).

```

1 import cv2
2 import io
3 import numpy as np
4 import os
5 import time
6 from time import sleep
7 from picamera import PiCamera
8 import sys
9 import math
10 sys.path.append("libs/classification/")
11 sys.path.append("libs/")
12 from classification_sample import processImg
13 from dictionary import dict
14
15 #PARAMETRAGE DE LA CAMÉRA _____
16 camera = PiCamera() #utilisation de la librairie PiCamera()
17 camera.resolution = (480, 480) #Vidéo de 1024*1024 pixels
18 camera.framerate = 30 #Nb d'images/s
19 camera.iso = 100
20 camera.image_effect = 'saturation' #Effet d'image #non = default value
21 camera.brightness = 40
22 camera.awb_mode = "shade" #Mise au point de la luminosité
23 #
24 i = 0
25 lock = 0 #Permet de forcer une attente entre 2 symboles pour que l'utilisateur puisse préparer son geste
26 texte_screen = ""
27 indx_code = 0
28 code = ['*', '*', '*', '*']
29 camera.start_preview() #Affiche le flux vidéo en direct
30 while(1):
31     start_time = time.time()
32     stream = io.BytesIO()
33     camera.capture(stream, format="jpeg", resize=(224, 224)) #Prend une "photo"
34     stream.seek(0)
35     verif_symbol_img = cv2.imdecode(np.fromstring(stream.read(), np.uint8), 1)
36     stream.flush()
37     t_prie = time.time() - start_time
38     start_time = time.time()
39     resultat = processImg('libs/network/1miohands-v2.xml', verif_symbol_img)
    #Appel fonction
40     t_trait = time.time() - start_time #Pour mesurer les performances
41     print("____ %s s pour prendre l'image ____"%(t_prie))
42     print("____ %s s pour traiter l'image ____"%(t_trait))
43     if lock == 0 :
44         if(resultat[1][resultat[0][0]]>0.5 and resultat[0][0] != 0): #On vérifie qu'un signe a été reconnu
45             lock = 1
46             texte_screen = str(dict[resultat[0][0]])
47             code[indx_code] = texte_screen
48             indx_code += 1
49         else :
50             texte_screen = str(dict[resultat[0][0]]) + ' - ' + "{:.2f}".
format(resultat[1][resultat[0][0]]*100) + '%'

```

```
51     else :
52         texte_screen = 'Signe suivant ... '
53 #Display result
54     texte_screen += "\n"
55     for codeI in code :
56         texte_screen += str(codeI) #Montre le code enregistré pour le moment
57     camera.annotate_text = texte_screen #Affiche le code sur la caméra
58     if lock == 1 :
59         i += 1
60         if i == 3 :
61             lock = 0
62             i = 0
63
64     cv2.waitKey(1)
65     print("i =", i, " | lock =", lock)
66     print('\n')
67     if indx_code == 4 : #Le code est enregistré
68         texte_fin = ""
69         for codeI in code :
70             texte_fin += str(codeI)
71         print(texte_fin)
72         input("Appuyer sur une entrée")
73         break; #On arrête le programme
74
75 camera.stop_preview()
```

Annexe 3

> Signes reconnus par l'intelligence artificielle

Nr	Shape	Name	Nr	Shape	Name
1		1	31		l_hook
2		2	32		middle
3		3	33		m
4		3_hook	34		n
5		4	35		o
6		5	36		index
7		6	37		index flex
8		7	38		index_hook
9		8	39		pincer
10		a	40		ital
11		b	41		ital_thumb
12		b_nothumb	42		ital_nothumb
13		b_thumb	43		ital_open
14		cbaby	44		r
15		obaby	45		s
16		by	46		write
17		c	47		spoon
18		d	48		t
19		e	49		v
20		f	50		v flex
21		f_open	51		v_hook
22		fly	52		v_thumb
23		fly_nothumb	53		w
24		g	54		y
25		h	55		ae
26		h_hook	56		ae_thumb
27		h_thumb	57		pincer_double
28		i	58		obaby_double
29		jesus	59		m2
30		k	60		jesus_thumb

Figure 20: Signes reconnus par l'intelligence artificielle

Annexe 4

> Dictionnaire pour la labellisation des signes reconnus

Voici le code python permettant la labellisation des signes reconnus

```

1  dict = [
2      "Not recognized",
3      "1",
4      "2",
5      "3",
6      "3 hook",
7      "4",
8      "5",
9      "6",
10     "7",
11     "8",
12     "a",
13     "b",
14     "b not thumb",
15     "b thumb",
16     "c baby",
17     "o baby",
18     "by",
19     "c",
20     "d",
21     "e",
22     "f",
23     "f open",
24     "fly",
25     "fly no thumb",
26     "g",
27     "h",
28     "h hook",
29     "h thumb",
30     "i",
31     "jesus",
32     "k",
33     "l hook",
34     "middle",
35     "m",
36     "n",
37     "o",
38     "index",
39     "index flex",
40     "index hook",
41     "pincet",
42     "ital",
43     "ital thumb",
44     "ital no thumb",
45     "ital open",
46     "r",
47     "s",
48     "write",
49     "spoon",
50     "t",
51     "v",
52     "v flex",
53     "v hook",
54     "v thumb",
55     "w",
56     "y",

```

```
57     "ae" ,  
58     "ae thumb" ,  
59     "pincet double" ,  
60     "o baby double" ,  
61     "m2" ,  
62     "jesus thumb"  
63 ]
```

Annexe 5

> Article de recherche sur la retro-propagation

Article complet au format pdf téléchargeable [ici](#).

NATURE VOL. 323 9 OCTOBER 1986

LETTERS TO NATURE

533

delineating the absolute indigeneity of amino acids in fossils. As AMS techniques are refined to handle smaller samples, it may also become possible to date individual amino acid enantiomers by the ^{14}C method. If one enantiomer is entirely derived from the other by racemization during diagenesis, the individual D- and L-enantiomers for a given amino acid should have identical ^{14}C ages.

Older, more poorly preserved fossils may not always prove amenable to the determination of amino acid indigeneity by the stable isotope method, as the prospects for complete replacement of indigenous amino acids with non-indigenous amino acids increases with time. As non-indigenous amino acids undergo racemization, the enantiomers may have identical isotopic compositions and still not be related to the original organisms. Such a circumstance may, however, become easier to recognize as more information becomes available concerning the distribution and stable isotopic composition of the amino acid constituents of modern representatives of fossil organisms. Also, AMS dates on individual amino acid enantiomers may, in some cases, help to clarify indigeneity problems, in particular when stratigraphic controls can be used to estimate a general age range for the fossil in question.

Finally, the development of techniques for determining the stable isotopic composition of amino acid enantiomers may enable us to establish whether non-racemic amino acids in some carbonaceous meteorites²⁷ are indigenous, or result in part from terrestrial contamination.

M.H.E. thanks the NSF, Division of Earth Sciences (grant EAR-8352055) and the following contributors to his Presidential Young Investigator Award for partial support of this research:

Arco, Exxon, Phillips Petroleum, Texaco Inc., The Upjohn Co. We also acknowledge the donors of the Petroleum Research Fund, administered by the American Chemical Society (grant 16144-AC2 to M.H.E., grant 14805-AC2 to S.A.M.) for support. S.A.M. acknowledges NSERC (grant A2644) for partial support.

Received 19 May; accepted 15 July 1986.

1. Bada, J. L. & Protsch, R. *Proc. natn. Acad. Sci. U.S.A.* **70**, 1331-1334 (1973).
2. Bada, J. L., Schroeder, R. A. & Carter, G. F. *Science* **184**, 791-793 (1974).
3. Boulton, G. S. *et al. Nature* **298**, 437-441 (1982).
4. Wehmiller, J. F. in *Quaternary Dating Methods* (ed. Mahaney, W. C.) 171-193 (Elsevier, Amsterdam, 1984).
5. Engel, M. H., Zumberge, J. E. & Nagy, B. *Analyt. Biochem.* **82**, 415-422 (1977).
6. Bada, J. L. *A Rev. Earth planet. Sci.* **13**, 241-268 (1985).
7. Chisholm, B. S., Nelson, D. E. & Schwarz, H. P. *Science* **216**, 1131-1132 (1982).
8. Ambrose, S. H. & DeNiro, M. J. *Nature* **319**, 321-324 (1986).
9. Macko, S. A., Estep, M. L. F., Hare, P. E. & Hoering, T. C. *Yb. Carnegie Instn Wash.* **82**, 404-410 (1983).
10. Hare, P. E. & Estep, M. L. F. *Yb. Carnegie Instn Wash.* **82**, 410-414 (1983).
11. Engel, M. H. & Hare, P. E. in *Chemistry and Biochemistry of the Amino Acids* (ed. Barrett, G. C.) 462-479 (Chapman and Hall, London, 1985).
12. Johnstone, R. A. W., Rose, M. E. in *Chemistry and Biochemistry of the Amino Acids* (ed. Barrett, G. C.) 480-524 (Chapman and Hall, London, 1985).
13. Weinstein, S., Engel, M. H. & Hare, P. E. in *Practical Protein Chemistry—A Handbook* (ed. Darbe, A.) 337-344 (Wiley, New York, 1986).
14. Bada, J. L., Gillespie, R., Gowlett, J. A. J. & Hedges, R. E. M. *Nature* **312**, 442-444 (1984).
15. Mitterer, R. M. & Kriauskul, N. *Org. Geochem.* **7**, 91-98 (1984).
16. Williams, K. M. & Smith, G. G. *Origins Life* **8**, 91-144 (1977).
17. Engel, M. H. & Hare, P. E. *Yb. Carnegie Instn Wash.* **81**, 425-430 (1982).
18. Hare, P. E. *Yb. Carnegie Instn Wash.* **73**, 576-581 (1974).
19. Pillinger, T. C. *Nature* **296**, 802 (1982).
20. Neuberger, A. *Adv. Protein Chem.* **4**, 298-383 (1948).
21. Engel, M. H. & Macko, S. A. *Analyst Chem.* **56**, 2598-2600 (1984).
22. Dungworth, G. *Chem. Geol.* **17**, 135-153 (1976).
23. Weinstein, S., Engel, M. H. & Hare, P. E. *Analyt. Biochem.* **121**, 370-377 (1982).
24. Macko, S. A., Lee, W. Y. & Parker, P. L. *J. exp. mar. Biol. Ecol.* **63**, 145-149 (1982).
25. Macko, S. A., Estep, M. L. F. & Hoering, T. C. *Yb. Carnegie Instn Wash.* **81**, 413-417 (1982).
26. Valentyne, J. R. *Geochim. cosmochim. Acta* **28**, 157-188 (1964).
27. Engel, M. H. & Nagy, B. *Nature* **296**, 837-840 (1982).

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure¹.

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of the input units. If the input units are directly connected to the output units it is relatively easy to find learning rules that iteratively adjust the relative strengths of the connections so as to progressively reduce the difference between the actual and desired output vectors². Learning becomes more interesting but

more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input-output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j , to unit j is a linear function of the outputs, y_i , of the units that are connected to j and of the weights, w_{ji} , on these connections

$$x_j = \sum_i y_i w_{ji} \quad (1)$$

Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

A unit has a real-valued output, y_j , which is a non-linear function of its total input

$$y_j = \frac{1}{1 + e^{-x_j}} \quad (2)$$

* To whom correspondence should be addressed.

534

LETTERS TO NATURE

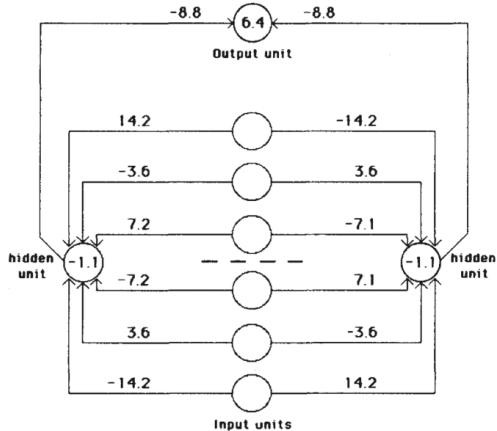


Fig. 1 A network that has learned to detect mirror symmetry in the input vector. The numbers on the arcs are weights and the numbers inside the nodes are biases. The learning required 1,425 sweeps through the set of 64 possible input vectors, with the weights being adjusted on the basis of the accumulated gradient after each sweep. The values of the parameters in equation (9) were $\epsilon = 0.1$ and $\alpha = 0.9$. The initial weights were random and were uniformly distributed between -0.3 and 0.3. The key property of this solution is that for a given hidden unit, weights that are symmetric about the middle of the input vector are equal in magnitude and opposite in sign. So if a symmetrical pattern is presented, both hidden units will receive a net input of 0 from the input units, and, because the hidden units have a negative bias, both will be off. In this case the output unit, having a positive bias, will be on. Note that the weights on each side of the midpoint are in the ratio 1:2:4. This ensures that each of the eight patterns that can occur above the midpoint sends a unique activation sum to each hidden unit, so the only pattern below the midpoint that can exactly balance this sum is the symmetrical one. For all non-symmetrical patterns, both hidden units will receive non-zero activations from the input units. The two hidden units have identical patterns of weights but with opposite signs, so for every non-symmetric pattern one hidden unit will come on and suppress the output unit.

It is not necessary to use exactly the functions given in equations (1) and (2). Any input-output function which has a bounded derivative will do. However, the use of a linear function for combining the inputs to a unit before applying the nonlinearity greatly simplifies the learning procedure.

The aim is to find a set of weights that ensure that for each input vector the output vector produced by the network is the same as (or sufficiently close to) the desired output vector. If there is a fixed, finite set of input-output cases, the total error in the performance of the network with a particular set of weights can be computed by comparing the actual and desired output vectors for every case. The total error, E , is defined as

$$E = \frac{1}{2} \sum_j (y_{j,c} - d_{j,c})^2 \quad (3)$$

where c is an index over cases (input-output pairs), j is an index over output units, y is the actual state of an output unit and d is its desired state. To minimize E by gradient descent it is necessary to compute the partial derivative of E with respect to each weight in the network. This is simply the sum of the partial derivatives for each of the input-output cases. For a given case, the partial derivatives of the error with respect to each weight are computed in two passes. We have already described the forward pass in which the units in each layer have their states determined by the input they receive from units in lower layers using equations (1) and (2). The backward pass which propagates derivatives from the top layer back to the bottom one is more complicated.

©1986 Nature Publishing Group

NATURE VOL. 323 9 OCTOBER 1986

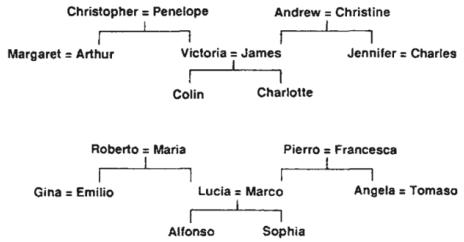


Fig. 2 Two isomorphic family trees. The information can be expressed as a set of triples of the form $\langle \text{person } 1 \rangle \langle \text{relationship} \rangle \langle \text{person } 2 \rangle$, where the possible relationships are {father, mother, husband, wife, son, daughter, uncle, aunt, brother, sister, nephew, niece}. A layered net can be said to 'know' these triples if it can produce the third term of each triple when given the first two. The first two terms are encoded by activating two of the input units, and the network must then complete the proposition by activating the output unit that represents the third term.

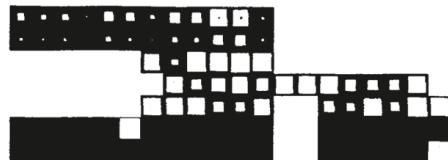


Fig. 3 Activity levels in a five-layer network after it has learned. The bottom layer has 24 input units on the left for representing $\langle \text{person } 1 \rangle$ and 12 input units on the right for representing the relationship. The white squares inside these two groups show the activity levels of the units. There is one active unit in the first group representing Colin and one in the second group representing the relationship 'has-aunt'. Each of the two input groups is totally connected to its own group of 6 units in the second layer. These groups learn to encode people and relationships as distributed patterns of activity. The second layer is totally connected to the central layer of 12 units, and these are connected to the penultimate layer of 6 units. The activity in the penultimate layer must activate the correct output units, each of which stands for a particular $\langle \text{person } 2 \rangle$. In this case, there are two correct answers (marked by black dots) because Colin has two aunts. Both the input units and the output units are laid out spatially with the English people in one row and the isomorphic Italians immediately below.

The backward pass starts by computing $\partial E / \partial y_j$ for each of the output units. Differentiating equation (3) for a particular case, c , and suppressing the index c gives

$$\partial E / \partial y_j = y_j - d_j \quad (4)$$

We can then apply the chain rule to compute $\partial E / \partial x_i$

$$\partial E / \partial x_i = \partial E / \partial y_j \cdot \partial y_j / \partial x_i$$

Differentiating equation (2) to get the value of $\partial y_j / \partial x_i$ and substituting gives

$$\partial E / \partial x_i = \partial E / \partial y_j \cdot y_j (1 - y_j) \quad (5)$$

This means that we know how a change in the total input x to an output unit will affect the error. But this total input is just a linear function of the states of the lower level units and it is also a linear function of the weights on the connections, so it is easy to compute how the error will be affected by changing these states and weights. For a weight w_{ji} , from i to j the derivative is

$$\begin{aligned} \partial E / \partial w_{ji} &= \partial E / \partial x_i \cdot \partial x_i / \partial w_{ji} \\ &= \partial E / \partial x_i \cdot y_i \end{aligned} \quad (6)$$

and for the output of the i^{th} unit the contribution to $\partial E / \partial y_i$

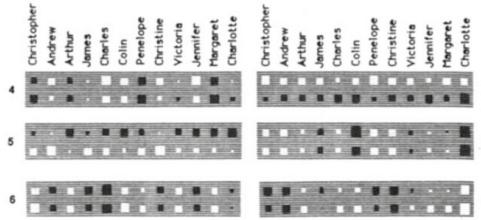


Fig. 4 The weights from the 24 input units that represent people to the 6 units in the second layer that learn distributed representations of people. White rectangles, excitatory weights; black rectangles, inhibitory weights; area of the rectangle encodes the magnitude of the weight. The weights from the 12 English people are in the top row of each unit. Unit 1 is primarily concerned with the distinction between English and Italian and most of the other units ignore this distinction. This means that the representation of an English person is very similar to the representation of their Italian equivalent. The network is making use of the isomorphism between the two family trees to allow it to share structure and it will therefore tend to generalize sensibly from one tree to the other. Unit 2 encodes which generation a person belongs to, and unit 6 encodes which branch of the family they come from. The features captured by the hidden units are not at all explicit in the input and output encodings, since these use a separate unit for each person. Because the hidden features capture the underlying structure of the task domain, the network generalizes correctly to the four triples on which it was not trained. We trained the network for 1500 sweeps, using $\epsilon = 0.005$ and $\alpha = 0.5$ for the first 20 sweeps and $\epsilon = 0.01$ and $\alpha = 0.9$ for the remaining sweeps. To make it easier to interpret the weights we introduced 'weight-decay' by decrementing every weight by 0.2% after each weight change. After prolonged learning, the decay was balanced by $\partial E / \partial w$, so the final magnitude of each weight indicates its usefulness in reducing the error. To prevent the network needing large weights to drive the outputs to 1 or 0, the error was considered to be zero if output units that should be on had activities above 0.8 and output units that should be off had activities below 0.2.

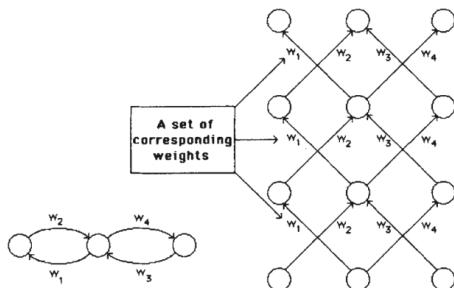


Fig. 5 A synchronous iterative net that is run for three iterations and the equivalent layered net. Each time-step in the recurrent net corresponds to a layer in the layered net. The learning procedure for layered nets can be mapped into a learning procedure for iterative nets. Two complications arise in performing this mapping: first, in a layered net the output levels of the units in the intermediate layers during the forward pass are required for performing the backward pass (see equations (5) and (6)). So in an iterative net it is necessary to store the history of output states of each unit. Second, for a layered net to be equivalent to an iterative net, corresponding weights between different layers must have the same value. To preserve this property, we average $\partial E / \partial w$ for all the weights in each set of corresponding weights and then change each weight in the set by an amount proportional to this average gradient. With these two provisos, the learning procedure can be applied directly to iterative nets. These nets can then either learn to perform iterative searches or learn sequential structures⁴.

resulting from the effect of i on j is simply

$$\partial E / \partial x_j \cdot \partial x_j / \partial y_i = \partial E / \partial x_j \cdot w_{ji}$$

so taking into account all the connections emanating from unit i we have

$$\partial E / \partial y_i = \sum_j \partial E / \partial x_j \cdot w_{ji} \quad (7)$$

We have now seen how to compute $\partial E / \partial y$ for any unit in the penultimate layer when given $\partial E / \partial y$ for all units in the last layer. We can therefore repeat this procedure to compute this term for successively earlier layers, computing $\partial E / \partial w$ for the weights as we go.

One way of using $\partial E / \partial w$ is to change the weights after every input-output case. This has the advantage that no separate memory is required for the derivatives. An alternative scheme, which we used in the research reported here, is to accumulate $\partial E / \partial w$ over all the input-output cases before changing the weights. The simplest version of gradient descent is to change each weight by an amount proportional to the accumulated $\partial E / \partial w$

$$\Delta w = -\epsilon \partial E / \partial w \quad (8)$$

This method does not converge as rapidly as methods which make use of the second derivatives, but it is much simpler and can easily be implemented by local computations in parallel hardware. It can be significantly improved, without sacrificing the simplicity and locality, by using an acceleration method in which the current gradient is used to modify the velocity of the point in weight space instead of its position

$$\Delta w(t) = -\epsilon \partial E / \partial w(t) + \alpha \Delta w(t-1) \quad (9)$$

where t is incremented by 1 for each sweep through the whole set of input-output cases, and α is an exponential decay factor between 0 and 1 that determines the relative contribution of the current gradient and earlier gradients to the weight change.

To break symmetry we start with small random weights. Variants on the learning procedure have been discovered independently by David Parker (personal communication) and by Yann Le Cun³.

One simple task that cannot be done by just connecting the input units to the output units is the detection of symmetry. To detect whether the binary activity levels of a one-dimensional array of input units are symmetrical about the centre point, it is essential to use an intermediate layer because the activity in an individual input unit, considered alone, provides no evidence about the symmetry or non-symmetry of the whole input vector, so simply adding up the evidence from the individual input units is insufficient. (A more formal proof that intermediate units are required is given in ref. 2.) The learning procedure discovered an elegant solution using just two intermediate units, as shown in Fig. 1.

Another interesting task is to store the information in the two family trees (Fig. 2). Figure 3 shows the network we used, and Fig. 4 shows the 'receptive fields' of some of the hidden units after the network was trained on 100 of the 104 possible triples.

So far, we have only dealt with layered, feed-forward networks. The equivalence between layered networks and recurrent networks that are run iteratively is shown in Fig. 5.

The most obvious drawback of the learning procedure is that the error-surface may contain local minima so that gradient descent is not guaranteed to find a global minimum. However, experience with many tasks shows that the network very rarely gets stuck in poor local minima that are significantly worse than the global minimum. We have only encountered this undesirable behaviour in networks that have just enough connections to perform the task. Adding a few more connections creates extra dimensions in weight-space and these dimensions provide paths around the barriers that create poor local minima in the lower dimensional subspaces.

The learning procedure, in its current form, is not a plausible model of learning in brains. However, applying the procedure to various tasks shows that interesting internal representations can be constructed by gradient descent in weight-space, and this suggests that it is worth looking for more biologically plausible ways of doing gradient descent in neural networks.

We thank the System Development Foundation and the Office of Naval Research for financial support.

Received 1 May; accepted 31 July 1986.

1. Rosenblatt, F. *Principles of Neurodynamics* (Spartan, Washington, DC, 1961).
2. Minsky, M. L. & Papert, S. *Perceptrons* (MIT, Cambridge, 1969).
3. Le Cun, Y. Proc. *Cognitiva* 85, 599–604 (1985).
4. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: *Foundations* (eds Rumelhart, D. E. & McClelland, J. L.) 318–362 (MIT, Cambridge, 1986).

Bilateral amblyopia after a short period of reverse occlusion in kittens

Kathryn M. Murphy* & Donald E. Mitchell

Department of Psychology, Dalhousie University,
Halifax Nova Scotia, Canada B3H 4J1

The majority of neurones in the visual cortex of both adult cats and kittens can be excited by visual stimulation of either eye. Nevertheless, if one eye is deprived of patterned vision early in life, most cortical cells can only be activated by visual stimuli presented to the nondeprived eye and behaviourally the deprived eye is apparently useless^{1,2}. Although the consequences of monocular deprivation can be severe, they can in many circumstances be rapidly reversed with the early implementation of reverse occlusion which forces the use of the initially deprived eye^{3,4}. However, by itself reverse occlusion does not restore a normal distribution of cortical ocular dominance³ and only promotes visual recovery in one eye^{5,6}. In an effort to find a procedure that might restore good binocular vision, we have examined the effects on acuity and cortical ocular dominance of a short, but physiologically optimal period of reverse occlusion, followed by a period of binocular vision beginning at 7.5 weeks of age. Surprisingly, despite the early introduction of binocular vision, both eyes attained acuities that were only approximately 1/3 of normal acuity levels. Despite the severe bilateral amblyopia, cortical ocular dominance appeared similar to that of normal cats. This is the first demonstration of severe bilateral amblyopia following consecutive periods of monocular occlusion.

Nine kittens were used, of which eight were monocularly deprived by eyelid suture from about the time of natural eye opening (6 to 11 days) until 5 weeks of age, at which time the initially deprived eye was opened and the other eye was sutured closed for 18 days. Physiological recordings from area 17 were made from one normal control and from five monocularly-deprived kittens, one immediately after reverse occlusion (as a control); the remaining four after a further 4 weeks at least (range 4–8 weeks) of normal binocular vision. Grating acuity thresholds were determined for both eyes of a further three kittens (subjected to the same regime—monocular deprivation, 18 days reverse suturing, followed by normal binocular vision) by use of a jumping stand^{5,7}. None of the kittens tested behaviourally were examined physiologically. Single unit recordings were made in area 17 of the anaesthetized, paralysed kittens (one normal, five experimental) with glass coated platinum-iridium electrodes. Anaesthesia was induced by

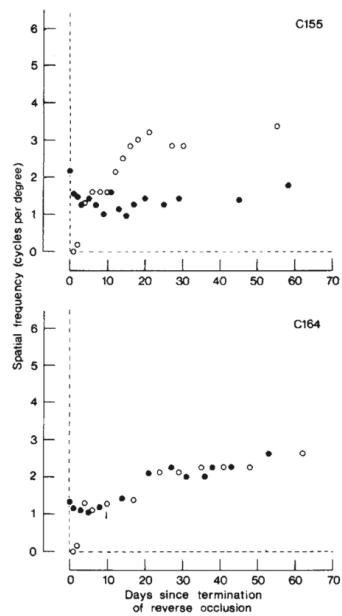


Fig. 1. Changes in visual acuity during the period of binocular vision for two kittens (C155 and C164) that were previously monocularly deprived until 5 weeks of age, and then reverse occluded for 18 days. ●, Acuity of the initially deprived eye; ○, acuity of the initially nondeprived eye.

intravenous pentothal and maintained by artificial respiration with 70% N₂O and 30% O₂ supplemented with intravenous Nembutal; EEG, EKG, body temperature, and expired CO₂ levels were monitored. The eyes were brought to focus on a tangent screen 137 cm distant from the kitten using contact lenses with 3 mm artificial pupils. Single units were recorded along one long penetration in area 17 down the medial bank of the postlateral gyrus in each hemisphere, always beginning in the hemisphere contralateral to the initially open eye. Receptive fields were sampled according to established procedures⁸, every 100 µm along the penetration in a cortical region corresponding to the horizontal meridian of visual space. All units were located within 15° of the area centralis, with the majority within 5°.

The longitudinal changes in visual acuity of both eyes following introduction of binocular vision are shown in Fig. 1 for two representative kittens. At the end of 18 days of reverse occlusion the vision of the initially deprived eye had recovered to only rudimentary levels (1–2.5 cycles per degree) while at the same time the initially nondeprived eye had been rendered blind. During the subsequent period of binocular visual exposure the vision of both eyes improved slightly, but only to a very limited extent (to between 1.7 and 3.4 cycles per degree). The results from the third animal were very similar. After more than 2 months of binocular exposure the acuities of the initially deprived and nondeprived eyes were respectively, 2.54 and 3.35 cycles per degree. Surprisingly, after 2 months of binocular vision, the acuity of both eyes of these animals remained at about one-third to one-half of normal levels⁶. Although the initially deprived eye was opened at the peak of the sensitive period (5 weeks of age) and the initially nondeprived eye was closed for a relatively brief period of time (18 days), this deprivation regimen had a devastating and permanent effect upon the visual acuity of both eyes.

* Present address: School of Optometry, University of California, Berkeley, California 94720, USA.

Sitographie

Oleksii Kharkovyna. *An intro to deep learning for face recognition.* Toward data science, 26 Juin 2019.

<https://towardsdatascience.com/an-intro-to-deep-learning-for-face-recognition-aa8dfbbc51fb>

Saidakbar. *Real-time face recognition: training and deploying on Android using Tensorflow lite.* Medium, 25 Février 2019.

<https://medium.com/@saidakbarp/real-time-face-recognition-tflite-3fb818ac039a>

Jason Brownlee. *How to Develop a Face Recognition System Using FaceNet in Keras.* Machine learning mastery, 7 Juin 2019.

<https://machinelearningmastery.com/how-to-develop-a-face-recognition-system-using-facenet-in-keras-and-an-svm-classifier>

Filipe Borba. *Using Deep Learning and CNNs to make a Hand Gesture recognition model.* Toward data science, 7 Mai 2019.

<https://towardsdatascience.com/tutorial-using-deep-learning-and-cnns-to-make-a-hand-gesture-recognition-model-371770b63a51>

Lucien Vieillard-Baron. *Une IA de Google pour traduire le langage des signes .* Rotek, Août 2019.

<https://rotek.fr/ia-google-traduire-langage-des-signes>

Yazid Benazzouz. *Gesture recognition using Movidius.* Medium, 26 Novembre 2018.

<https://medium.com/@yazid.benazzouz/gesture-recognition-using-movidius-b800c99c1d33>

Wikipedia. *Fonctionnement des réseaux de neurones artificiels.*

<https://fr.wikipedia.org/wiki/Réseau-de-neurones-artificiels>

Intel. *Documentation OpenVino.*

<https://software.intel.com/content/www/us/en/develop/tools/opencv-toolkit.html>

CNRS. *PLM latex - développement de document LaTeX.*

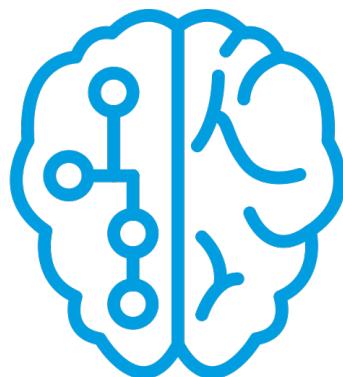
<https://plmlatex.math.cnrs.fr>

Résumé

Ce dossier rapporte le travail effectué lors du projet de 4eme année du cycle ingénieur (spécialité SAGI). Celui-ci portait initialement sur la reconnaissance faciale à l'aide de réseaux de neurones sur un appareil embarqué. Le sujet a été changé au profit de la reconnaissance de signes par réseau de neurones, au vu de la complexité du sujet initial. Les scripts Python et le réseau ainsi développé permettent la reconnaissance de certains gestes du langage des signes par une Raspberry Pi muni d'une caméra. Le finalité de ce projet est de contrôler l'accès à une pièce grâce à un code composé de signes que l'utilisateur doit réaliser devant la caméra, permettant de vérifier les autorisations sans requérir un contact physique par l'utilisateur.

Abstract

This document report the work accomplished for our project during the 4th year of engineering. While initially aiming to implement facial recognition using neural networks in an embedded system, the subject was changed to recognize some signs from the sign language as it was less complex to implement. The resulting Python scripts and neural network will allow for hand signs recognition from a raspberry pi. In the end, this project will allow the restriction of access to a room using a "sign password" which the user can enter without any physical interaction with the device.



Mots clés :

- intelligence artificielle
- réseau de neurones
- deep-learning
- rétro-propagation
- python
- reconnaissance de signes
- reconnaissance faciale