Groupe 5: Détection automatique d'URLs malicieuses

Jeremi Levesque LEVJ1404
Timothée Blanchy TIMB1101
Chris Tchimmegne Tchassem TCHC1301

Abstract

Les URLs malicieuses sont nombreuses et peuvent causer des dommages financiers importants. Notre contribution cherche à les détecter automatiquement en utilisant les modèles à l'état de l'art, tout en offrant une interprétabilité aux classifications des modèles complexes. Nous avons donc cherché à accroître l'interprétabilité des modèles en utilisant LIME (Local Interpretable Model-Agnostic Explanations) pour pouvoir expliquer les classifications produites par notre modèle.

1 Introduction

En 2022, le FBI estime que les URLs malicieuses ont causé des pertes de 2.7 milliards $USD aux entreprises. Sur son site, le FBI (2022) recommande au public dans un rapport de porter une attention particulière aux URLs qui circulent sur le web. Ainsi, nous proposons d'utiliser des techniques de détection automatique d'URLs malicieuses afin de réduire la quantité d'URLs qui atteignent les utilisateurs de sorte à diminuer le risque que les sites frauduleux puissent causer des dommages. Nous attaquons le problème en faisant une classification binaire des URLs basée sur leur structure lexicale.

URLs en entrée	Type (sortie)
icloud.com	Bénin
br-icloud.com.br	Malicieux
mp3raid.com/music/krizz_kaliko.html	Bénin

Table 1: Exemples de classifications attendues d'URLs

Comme détaillé dans la table 1, certaines URLs malicieuses copient la structure d'autres URLs bénignes afin de duper l'utilisateur ou même un système qui serait basé sur quelques caractéristiques définies. Il existe toutes sortes d'URLs légitimes telles que mapple.com alors que dapple.com est malicieux.

2 Travaux connexes

Traditionnellement, l'utilisation de *blacklists*, notamment par Sun et al. (2015), Prakash et al. (2010) et Ma et al. (2009), était très répandue et efficace afin de filtrer les URLs malicieuses. Aujourd'hui, les URLs malicieuses sont créées en grand nombre et ce, de manière automatique, ce qui rend l'utilisation des *blacklist* difficile. Des méthodes d'apprentissage machine (SVM, Régression logistique, Arbres de décisions, etc.) ont étés utilisées afin d'analyser la structure syntaxique des URLs et celles-ci offrent des résultats compétitifs dans et Sahoo et al. (2017). Par contre, ces méthodes ont de la difficulté à gérer des *patterns* séquentiels et à gérer de toute nouvelle caractéristique puisque ces méthodes nécessitent des caractéristiques minutieusement annotées par des experts. Le et al. (2018) utilisent des modèles de réseaux de neurones à convolutions (CNN) (utilisés avec des réseaux temporels par Das et al. (2020)) permettent d'identifier automatiquement les caractéristiques discriminantes et ont une meilleure capacité de généralisation à de nouvelles caractéristiques. Finalement, très récemment, le papier de Vaswani et al. (2017) sur les *transformers* résulte en de performances qui deviennent rapidement l'état de l'art dans pratiquement tous les domaines du TALN. Ils ont donc été mis à l'essai par Maneriker et al. (2021), Wang and Chen (2022) et Su and Su (2023) où chacun utilise une version modifiée de BERT, de Devlin et al. (2018), ou RoBERTa, de Liu et al. (2019), pour analyser syntaxiquement les URLs. Les résultats de ces mises à l'essai sont impressionnants en font l'état de l'art en matière de performance dans ce domaine. Par contre, comme ceux-ci sont basés sur des réseaux de neurones, il est bien connu que ces derniers sont souvent considérés comme des *boîtes noires* puisqu'ils sont difficilement interprétables ce qui pose un problème de confiance dans plusieurs domaines.

Notamment, en cybersécurité, les analystes privilégient des solutions interprétables et compréhensibles, telles que la détection basée sur des règles ou des signatures comme détaillé par Brigugilio (2020). Cela est dû à la nécessité de régler et d'optimiser ces solutions pour atténuer et contrôler l'effet des faux positifs et des faux négatifs. Une difficulté avec l'interprétabilité de l'apprentissage automatique dans ce domaine est que bon nombre des caractéristiques (valeurs des neurones) ne sont pas significatives pour les humains sans le contexte dans lequel elles apparaissent. Certes, les modèles de *transformers* sont munis de couches

d'auto-attention qui peuvent nous donner quelques indications sur le comportement du modèle. Des outils comme BertViz, développé par Vig (2019), nous permettent de facilement voir ces couches d'attention sur chaque jeton ce qui améliore donc l'interprétabilité de ces modèles. Par contre, la visualisation de l'attention permet de mettre en lumière un type d'architecture, mais ne donne pas nécessairement une explication directe des prédictions faites par le modèle. De plus, les modèles sans couches d'attention (e.g. les CNNs, LSTM, etc.) ne peuvent pas bénéficier d'un tel outil de visualisation. Nous cherchons donc à permettre une meilleure interprétabilité de tout type de modèle qui peut être utilisé en appliquant la méthode de Ribeiro et al. (2016), nommée LIME (détails dans la section 3) pour une première fois dans un contexte de classification d'URLs.

3 Méthodologie

3.1 Architecture des modèles

Les arbres de décisions sont des modèles simples qui sont reconnus pour offrir une bonne interprétabilité des résultats. Nous avons donc implémenté un arbre de décision pour nous donner une référence d'un modèle plus simple à interpréter. De la sorte, nous pouvons comparer les interprétations des modèles plus complexes à ceux des modèles plus simples. Peut-être que les interprétations seront les mêmes, ou encore différentes ce qui explique leur différences de performances et de généralisation? Les arbres de décisions permettent de diviser les données selon des caractéristiques prédéterminées jugées discriminantes par le modèle ce qui signifie que notre jeu de données doit être modifié afin d'inclure des annotations de caractéristiques en plus d'avoir les URLs.

Nous avons aussi décider d'utiliser un Multi-Layer Perceptron (MLP), un modèle légèrement plus performant. Nous avons décidé de nous baser sur l'encodage de Zhang et al. (2016), basé sur un dictionnaire de 70 caractères, qui transforme une URL en un one-hot vector de dimension $(longueur\_max\_url, nb\_caract\_dic)$. Les URLs pouvant être très longues, après avoir constaté que la longueur moyenne d'URL était d'environ 60 caractères, nous avons arbitrairement choisi 256 caractères comme longueur maximum. Le one-hot vector obtenu est éparse: il y a bien plus de 0 que de 1 dedans. Le MLP est détaillé schématiquement dans la figure 2 de l'annexe pour plus de détails.

Pour le réseau à convolution (CNN), nous avons gardé le même encodage que pour le MLP. La figure 3 de l'annexe détaille l'architecture du réseau. Nous avons décidé d'intégrer une couche pleinement connectée avant les convolutions, car cela nous semblait important, avec en entrée un one-hot vector avec de l'infomrmation éparse.

Nous avons aussi implémenté les *BertModel* et *RobertaModel* de PyTorch pré-entraîné qui nous donnent en sortie le dernier état caché du modèle. Nous avons ainsi pu rajouter notre propre tête de classification avec une couche pleinement connectée en sortie de façon équivalente pour les deux modèles. Cette dernière couche résulte en 1 seul neurone de sortie auquel une sigmoïde est appliquée afin que la sortie de nos réseaux **RobertaUrl** et **BertUrl** soit $p(malicious|url, w)$. RoBERTa et BERT ont été pré-entraînés avec leurs *tokenizers* respectifs: *Byte-Pair Encoding* et le *WordPiece*. Nous avons donc utiliser ces mêmes méthodes de *tokenization* qui proviennent de ces modèles pré-entraînés afin de pouvoir jetoniser facilement les URLs en entrée de ces modèles.

3.2 LIME

Pour résoudre le problème de l'interprétabilité des réseaux profonds tels que les MLPs, CNNs et les *transformers*, nous avons décidé d'utiliser **LIME**[1], pour « Local Interpretable Model-agnostic Explanations » par Ribeiro et al. (2016), qui propose d'expliquer les origines des prédictions de façon agnostique au modèle. Tout d'abord, cette méthode génère plusieurs points dans l'espace des caractéristiques de sorte à avoir 5 000 petites variations d'une URL d'intérêt que nous tentons d'interpréter et on vient donner toutes ces variations à notre modèle profond pour voir son comportement. Avec ces points, on a alors une bonne idée du comportement du modèle localement pour cette URL d'intérêt. L'objectif est donc de venir entraîner une régression logistique (qui est interprétable puisque les poids correspondent directement à l'importance des jetons) sur notre ensemble de données locales afin que ce modèle puisse approximer le comportement de notre modèle profond pour la prédiction de cette donnée et ainsi nous donner une meilleure interprétation pour sa prédiction. L'algorithme se résume donc de la façon suivante:

1. Choisir une donnée pour interpréter le com-

[1] LIME est d'ailleurs disponible en open-source.

portement du modèle.
2. Générer des points un peu partout dans l'espace des caractéristiques.
3. Prédire la sortie du modèle pour ces points.
4. Pondérer ces points basés sur la proximité avec le point de référence.
5. Entraîner un modèle de substitution simple et explicable (e.g. régression logistique) sur les points pondérés pour approximer les prédictions du modèle original localement.

Cette méthode a déjà utilisé dans contextes similaires par Aslam et al. (2022) pour expliquer le raisonnement de certains modèles dits *boîtes noires*, mais la combinaison de la performance des *transformers* et de la puissance l'interprétabilité de LIME n'a jamais été réalisée à notre connaissance pour des URLs.

4 Expériences

Comme nous n'utilisons pas nécessairement un ensemble de données comparable aux résultats qui sont obtenus dans la littérature, il est important de d'abord évaluer les performances des architectures classiques décrites dans la section 3. De cette façon, nous pouvons confirmer que les résultats obtenus dans la littérature concordent avec notre ensemble de données et que les tests d'interprétabilités seront adéquats par rapport à la performance des modèles. Par la suite, comme notre objectif est d'améliorer l'interprétabilité des modèles profonds dans la détection d'URLs, nous avons fait une analyse qualitative des résultats de l'application de LIME sur nos principaux modèles.

4.1 Données

Les données utilisées pour notre entraînement proviennent des deux ensembles de données suivants qui sont publiques qui ont été mis en place notamment pour la détection d'URLs malicieuses.

1. "Malicious URLs dataset" sur kaggle, avec 641 119 données classéss en 4 catégories (*benign*, *defacement*, *phishing*, *malware*)
2. "Detect Malicious URL using ML" sur kaggle, avec 450 176 données classées de façon binaire (*benign* ou *malicious*)

Cependant, nous avons tout d'abord transformé l'ensemble de données 1 pour qu'il soit composé uniquement de classes binaires. Nous avons associé toutes les URLs non-bénignes à des URLs malicieuses. Donc, nous avons deux ensembles de données qui sont composés de paires ("url", x_c ∈{bénin (0), malicieux (1)}). Afin d'obtenir un plus grand ensemble de données, nous combinons ces deux ensembles de données en un seul en retirant les doublons de sorte à obtenir un ensemble de données étiquettées final de 1 090 236 URLs étiquettées qui est décrit par la table 2. Nous avons ensuite séparé aléatoirement les données entre le jeu d'entraînement (90% des données) et de test (10% des données).

Statistique	Valeur
Nombre total d'URLs	1090221
Nombre d'URLs bénignes	773818 (70.98 %)
Nombre d'URLs malicieuses	316403 (29.02 %)
Longueur moyenne	59.957
Longueur maximum	2314

Table 2: Statistiques du dataset

Comme notre arbre de décision nécessite d'avoir des caractéristiques définies afin de séparer les données d'entraînement et de tests en conséquence, nous avons mis en place un système qui permet d'extraire 99 caractéristiques (e.g. taille de l'URL, nombre de "&", nombre de "-", etc.) sur nos URLs à la volée durant l'entraînement et l'inférence de façon transparente aux utilisateurs du modèle. Les caractéristiques identifiées sur les URLs sont inspirées d'un autre ensemble de données publique nommé "Phishing Dataset" qui avait construit un grand jeu de données avec les caractéristiques déjà pré extraites.

4.2 Performances des modèles

Avec notre grand ensemble de données, entraîner nos modèles sur une seule nécessitait un temps considérable de temps de calcul pour la plupart de nos modèles (∼3h pour BERT & RoBERTa). Par contre, après ce seul époque, la majorité des modèles atteignaient des performances similaires à l'état de l'art qui sont indiquées dans la table 4. De toute façon, dans la majorité des cas, la perte à la fin de l'époque était quasi-nulle ce qui résulte en très peu d'apprentissage si nous étions pour faire plus d'époques. Comme attendu dans la littérature, les modèles d'attention performent mieux que les autres modèles. Lors de l'entraînement de notre arbre de décision, nous n'avons pas contraint la profondeur de l'arbre de décision ce qui résulte en un arbre de profondeur 60, ce qui permettrait de bien séparer toutes les données d'entraînement d'où une accuracy d'entraînement plus élevée que les autres. De plus, le CNN est notre méthode la moins performante, mais comme nous n'avons

pas utilisé les mêmes méthodes de *tokenization* que dans la littérature (qui combinait caractères et mots), peut-être que la généralisation de notre modèle en était impactée dans le cadre de ce projet.

Modèle	Train Acc.	Valid Acc.	Test Acc.
Arbre de décision	**99.44%**	96.95%	96.92%
MLP	96.07%	96.61%	96.54%
CNN	94.97%	96.01%	96.00%
BERT	98.89%	99.19%	99.14%
RoBERTa	98.98%	**99.24%**	**99.24%**

Table 3: Justesse moyenne des différents modèles sur le dataset combiné

Modèle	Precision	Recall	F1
Arbre de décision	0.9507	0.9425	0.9466
MLP	0.9594	0.9194	0.9390
CNN	0.9503	0.9094	0.9294
BERT	0.9870	0.9830	0.9850
RoBERTa	**0.9902**	**0.9836**	**0.9869**

Table 4: Métriques de performance moyens des différents modèles sur le dataset combiné

4.3 Interprétation des prédictions

Une fois les modèles entraînés et testés, nous avons utilisé LIME expliquer certaines prédictions de nos modèles de RoBERTa et de l'arbre de décision. Les prédictions sur les autres modèles avaient des comportements assez similaires à ce qui est présenté ici, c'est pourquoi on met uniquement l'accent sur ceux-ci. La figure 1A illustre que l'URL malicieuse est bien classée, surtout grâce au termes "*www*" et "*svtool*". Par contre, comme le montre la figure 1B, l'arbre de décision se base plutôt sur le "*index*" et "*html*" pour une même décision. La figure 1C donne un exemple pour une URL bénigne bien détectée, même si les raisons sont inattendues notamment que la présence du "*www*" influence pour beaucoup la prédiction dans la majorité des exemples. Si on donne une URL simplifiée sans "*https*" et sans "*www*", comme fait dans la figure 1D, on remarque que la prédiction est totalement erronée. Cette URL est clairement bénigne, alors que tous nos modèles la prédise comme étant malicieuse. C'est un scénario qui se reproduit avec plusieurs autres URLs et qui est un comportement inattendu et non désiré: il est généralement optionnel de spécifier les composantes initiales d'une URL.

D'une part, notre utilisation de LIME est bien limitée par la technique de *tokenization* supportée par LIME. Comme LIME a été adapté pour analyser des textes continus originalement, les jetons sont créés à partir des mots. Cela implique que les caractères spéciaux ne sont pas considérés pour

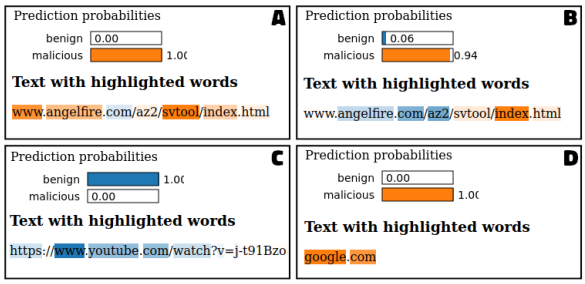

Figure 1: Résultats de LIME sur quelques URLs. (A) Prédiction de RoBERTa qui classifie correctement une URL malicieuse. (B) L'arbre de décision qui classifie correctement une URL malicieuse. (C) RoBERTa qui classifie correctement une URL bénigne. (D) RoBERTa qui classifie de façon erronée l'URL ne provenant pas de l'ensemble d'entraînement comme malicieuse.

l'interprétation des prédictions alors que la plupart de nos modèles tiennent compte de ceux-ci pour leurs prédictions. D'autre part, le fait que les "*www*" et une absence de "*https*" dans une URL influence énormément les résultats des prédictions révèle peut-être une faiblesse de nos jeux de données. Il n'est souvent pas nécessaire de spécifier ces composantes de l'URL pour accéder à un site web. Elles ne devraient donc normalement pas trop être prises en compte lors de la prédiction. Il faudrait alors peut-être assurer une représentation des URLs avec et sans ses composantes dans notre jeu de données afin d'éviter qu'autant de poids soit mis sur celles-ci.

5 Conclusion

En conclusion, il est clair qu'accroître l'interprétabilité des réseaux de neurones profonds reste un sujet de recherche actif. Notre expérimentation permet de mettre la lumière sur certaines caractéristiques identifiées par quelques modèles de l'état de l'art en la détection d'URLs malicieuses. Par contre, celle-ci est limitée par la *tokenization* qui, non seulement varie entre les modèles, mais la *tokenization* de LIME n'est probablement pas adaptée à celle des modèles testés. Peut-être qu'une utilisation de LIME avec une *tokenization* personnalisée selon chaque modèle pourrait offrir plus d'indications sur des caractéristiques plus précises de chaque URL. Il serait peut-être aussi pertinent de voir si d'autres modèles de la famille BERT pourraient offrir des résultats comparables et peut-être même de voir ce qu'un LLM (e.g. ChatGPT) serait capable d'offrir comme interprétabilité pour faire de la classification d'URLs.

References

Nida Aslam, Irfan Ullah Khan, Samiha Mirza, Alanoud AlOwayed, Fatima M Anis, Reef M Aljuaid, and Reham Baageel. 2022. Interpretable machine learning models for malicious domains detection using explainable artificial intelligence (xai). *Sustainability*, 14(12):7375.

William R Brigugilio. 2020. Machine learning interpretability in malware detection. Master's thesis, University of Windsor (Canada).

Arijit Das, Ankita Das, Anisha Datta, Shukrity Si, and Subhas Barman. 2020. Deep approaches on malicious url classification. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–6. IEEE.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Upendra Shetty DR, Anusha Patil, et al. 2023. Malicious url detection and classification analysis using machine learning models. In *2023 International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*, pages 470–476. IEEE.

FBI. 2022. Internet crime complaint center (ic3) 2022. *Internet Crime Complaint Center (IC3)*.

Hung Le, Quang Pham, Doyen Sahoo, and Steven CH Hoi. 2018. Urlnet: Learning a url representation with deep learning for malicious url detection. *arXiv preprint arXiv:1802.03162*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2009. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1245–1254.

Pranav Maneriker, Jack W Stokes, Edir Garcia Lazo, Diana Carutasu, Farid Tajaddodianfar, and Arun Gururajan. 2021. Urltran: Improving phishing url detection using transformers. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*, pages 197–204. IEEE.

Pawan Prakash, Manish Kumar, Ramana Rao Kompella, and Minaxi Gupta. 2010. Phishnet: predictive blacklisting to detect phishing attacks. In *2010 Proceedings IEEE INFOCOM*, pages 1–5. IEEE.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "why should i trust you?": Explaining the predictions of any classifier.

Doyen Sahoo, Chenghao Liu, and Steven CH Hoi. 2017. Malicious url detection using machine learning: A survey. *arXiv preprint arXiv:1701.07179*.

Ming-Yang Su and Kuan-Lin Su. 2023. Bert-based approaches to identifying malicious urls. *Sensors*, 23(20):8499.

Bo Sun, Mitsuaki Akiyama, Takeshi Yagi, Mitsuhiro Hatada, and Tatsuya Mori. 2015. Autoblg: Automatic url blacklist generator using search space expansion and filters. In *2015 IEEE Symposium on Computers and Communication (ISCC)*, pages 625–631. IEEE.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Jesse Vig. 2019. A multiscale visualization of attention in the transformer model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 37–42, Florence, Italy. Association for Computational Linguistics.

Chenguang Wang and Yuanyuan Chen. 2022. Tcurl: Exploring hybrid transformer and convolutional neural network on phishing url detection. *Knowledge-Based Systems*, 258:109955.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2016. Character-level convolutional networks for text classification.

A Principales bibliothèques

1. PyTorch
2. huggingface-hub
3. transformers (de HuggingFace)
4. json
5. matplotlib
6. pandas
7. scikit-learn
8. tqdm

B Contributions pour tout le projet

Code: Voir le code complet du projet sur notre **GitHub**

JEREMI LEVESQUE

Contribution:

- Rédaction du rapport final, progression & proposition.
- Revue de littérature.
- Implémentation BERT & RoBERTa.
- Combinaison et pré-traitement des ensembles de données.
- Balancement des ensemble de données (non-inclus dans le rapport).
- Implémentation de LIME et de la structure pour pouvoir interpréter plusieurs URLs facilement.
- Intégration du arbre de décision dans le projet.
- Implémentation du MLP.
- Génération et compilation des interprétations de LIME pour tous les modèles.
- Entraînement et compilation des résultats de tous les modèles.
- Préparation des slides de la présentation.
- Rédaction du README.md.

TIMOTHÉE BLANCHY

Contribution:

- Implémentation d'une partie des métriques.
- Ajout de plusieurs statistiques sur les datasets.
- Implémentation de l'extracteur de caractéristiques du arbre de décision.
- Préparation des slides revue de littérature.
- Implémentation du CNN et du BasicTokenizer.
- Rédaction du README.MD
- Rédaction du rapport final, & progression & proposition

CHRIS TCHIMMEGNE TCHASSEM

Contribution:

- Début d'implémentation du MLP
- Début d'implémentation de l'arbre de décision.
- Aide aux travaux connexes pour le rapport de progression.

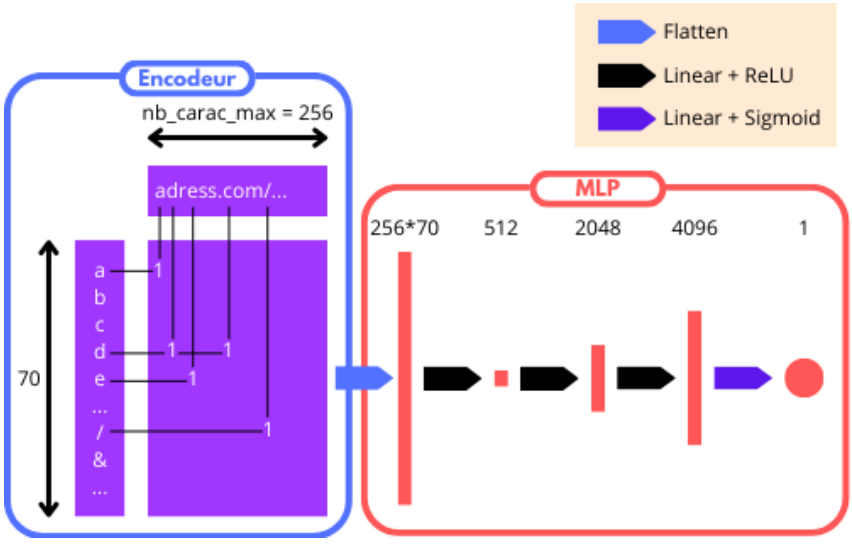

Figure 2: Réseau MLP implémenté

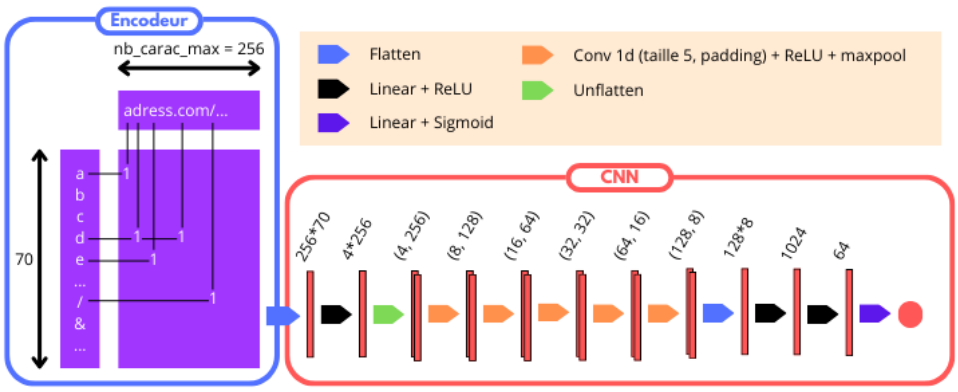

Figure 3: Réseau CNN implémenté