Projet Deep Learning : Salimans 16

1 Présentation du problème

Dans la plupart des domaines, la collection des données est relativement facile tandis que l'étiquetage de celles-ci par les humains peut être coûteux et surtout prend du temps. Une solution à cette problématique consiste en l'utilisation de modèles dit semi-supervisés. Ces modèles apprennent les patterns présents dans les données non étiquetées et combinent ces connaissances avec l'échantillon d'apprentissage étiqueté (souvent de petite taille) afin de réaliser une tâche d'apprentissage supervisé – en occurrence, classification des images. Dans ce rapport, nous représentons la méthode de Generative Adversarial Network GAN qui est couramment utilisée quand on ne dispose pas de suffisamment de données étiquetées ainsi que son architecture pour la classification des images. Notre travail s'appuie sur une méthode traitée dans un article scientifique pour la construction d'un modèle de classification semi-supervisée sur les données MNIST.

La database MNIST contient 60000 chiffres écrits à la main de 0 à 9, représentés par 28x28 pixels.

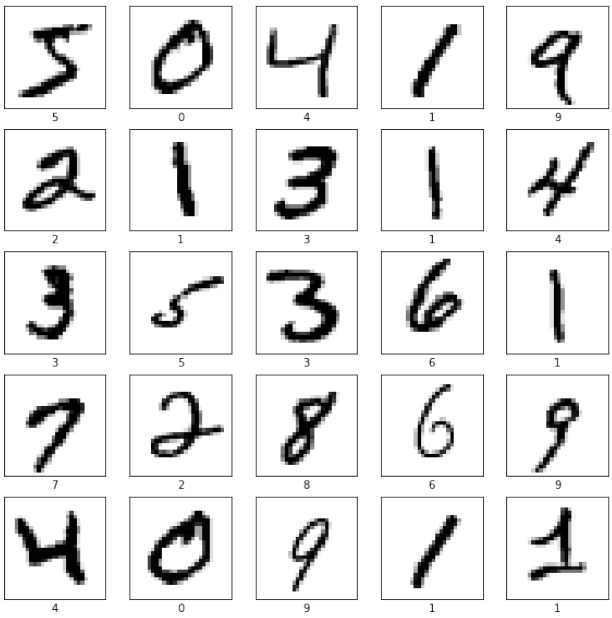

Figure 1: Exemple du contenu de la base de données MNIST

Pour entraîner notre algorithme, nous allons utiliser les 60000 chiffres écrits à la main mais seulement 100 labels (10 de chaque chiffres). Nous allons donc comparer des modèles baselines (CNN et DNN) basiques et ce qu'apporte en gain de performance le modèle présenté dans notre article de littérature qui est l'article **Improved techniques for Training GANs** par *Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen.*

Dans un premier temps, nous allons présenter la théorie des méthodes présentées dans l'article. Ensuite, nous présenterons comment sont construits nos modèles puis nous comparerons les résultats.

2 Techniques de l'article Salimans 16

2.1 Méthode GANs

Les **Generative adversial networks** sont des méthodes d'apprentissage génératifs. Leur but est d'entraîner un réseau générateur $G(z;\theta^{(G)})$ qui produit des images de chiffres manuscrits à partir de la data $p_{data}(x)$ en transformant des vecteurs de bruits z en des vecteurs $x = G(z;\theta^{(G)})$. Le modèle génératif est entraîné par un réseau classificateur $D(x)$ qui est entraîné pour distinguer la data réelle et la data générée par le modèle génératif $p_{model}(x)$. Ensuite, le générateur est à son tour entraîné pour déjouer le classificateur.

2.2 Convergence du GAN

Le problème de l'entraînement du GAN est que chaque modèle veut minimiser sa fonction de coût $J^{(D)}(\theta^{(D)}, \theta^{(G)})$ pour le modèle classificateur et $J^{(G)}(\theta^{(D)}, \theta^{(G)})$ pour le modèle générateur. Trouver cet équilibre est un problème compliqué.

2.2.1 Feature Matching

La technique du Feature Matching répond au problème d'instabilité en specifiant un nouvel objectif sur le générateur pour éviter le surapprentissage sur le classificateur. Dans ce cas, on ne maximise pas directement l'output du classificateur. Le générateur doit générer de la data qui "match" les caractéristiques de la data réelle à la sortie d'une sous couche du classificateur.

Si on note $f(x)$ les activations sur une couche intermédiaire du classificateur, notre nouvel objectif se définit comme ceci :

$$||\mathbb{E}_{x\sim p_{data}} f(x) - \mathbb{E}_{z\sim p_z(z)} f(G(z))||_2^2 \tag{1}$$

2.2.2 Minibatch discrimination

Le but de la "minibatch discrimination" est d'éviter que le générateur converge vers un seul et même point et qu'il emette toujours le même résultat. Le classificateur va alors apprendre que ce point unique vient du générateur et le modèle général ne va pas converger.

Le concept de cette méthode est général : le fait que n'importe quel classificateur qui regarde plusieurs exemples de combinaison plutôt que chaque cas isolé va éviter la "mort" du générateur.

Une manière de modéliser les minibatch :

- $f(x_i) \in \mathbb{R}^A$ est un vecteur de features pour un input x_i produit par une couche intermédiaire du classificateur.
- on multiplie le vecteur $f(x_i)$ par un tensor $T \in \mathbb{R}^{AxBxC}$ qui donne une matrice $M_i \in \mathbb{R}^{BxC}$
- on calcule ensuite la L_1-distance entre les lignes de M_i et on applique un exponentielle négative $c_b(x_i, x_j) = exp(-||M_{i,b} - M_{j,b}||_{L_1}) \in \mathbb{R}$

La sortie de ce minibatch $o(x_i)$ pour un vecteur x_i est défini comme la somme des $c_b(x_i, x_j)$:

$$o(x_i)_b = \sum_{j=1}^{n} c_b(xi, xj) \in \mathbb{R} \tag{2}$$

Enfin il faut concaténer $o(xi)$ avec les features $f(x_i)$ qui étaient dans l'input et donner ce résultat à la prochaine couche du classificateur. Le classificateur doit toujours renvoyer si la source vient de la data ou du générateur mais il peut se servir des minibatch comme de l'information supplémentaire.

Les minibatch sont très utiles pour la génération de données. Cependant, le feature matching est largement meilleur pour construire un classificateur fort avec de l'apprentissage semi-supervisé, nous n'utiliserons donc pas cette méthode.

2.2.3 One-sided label smoothing

Au lieu d'avoir des 0 et des 1 comme résultat du classificateur, on les remplace par des valeurs comme 0.9 ou 0.1.

En remplacant les 1 par α et les 0 par β, le classificateur optimal devient $D(x) = \frac{\alpha p_{data}(x)+\beta p_{model}(x)}{p_{data}(x)+p_{model}(x)}$.

La présence de p_{model} peut poser problème puisque p_{data} peut être proche de 0 et p_{model} très grand. On peut alors remplacé 1 par α et garder les 0 en 0.

On a alors

$$D(x) = \frac{\alpha p_{model}}{p_{data}(x) + p_{model}(x)}$$

2.3 Semi-supervised learning

On considère un classificateur standard qui classifie un point de data x dans une des K classes possibles (10 dans notre cas). Ce modèle prend en input le point x et donne en outputs un vecteur de logits de taille K $\{l_1, \ldots, l_K\}$ qui peut être transformé en probabilités de classes en appliquant le softmax :

$$p_{model}(y = j|x) = \frac{exp(l_j)}{\sum_{k=1}^{K} exp(l_k)}$$

Dans le cas d'un modèle supervisé, un tel modèle est entraîné en minimisant la cross-entropy entre les datas observées et le modèle prédictif de distribution $p_{model}(y|x)$.

Dans le cas semi-supervisé, il suffit d'ajouter des datas du générateur GAN et de rajouter une classe K+1. $p_{model}(y = K + 1|x)$ est alors la probabilité que x soit faux (venant du générateur) et correspond à $1 - D(x)$.

Dans le cas du non-supervisé, on peut aussi classifier, en fixant arbitrairement que la moitié de la data vient de la vraie data et l'autre moitié d'une data générée.

On a alors une fonction de perte :

$$\begin{aligned} L &= -\mathbb{E}_{x,y\sim p_{data}(x,y)}[\log p_{model}(y|x)] - \mathbb{E}_{x\sim G}[\log p_{model}(y = K + 1|x)] \\ &= L_{supervised} + L_{unsupervised} \end{aligned}$$

Avec :

$$L_{supervised} = -\mathbb{E}_{x,y\sim p_{data}(x,y)}[\log p_{model}(y|x, y < K + 1)]$$

$$L_{unsupervised} = -\{\mathbb{E}_{x\sim p_{data}(x)}[\log[1 - p_{model}(y = K + 1|x)]] + \mathbb{E}_{z\sim G}[\log p_{model}(y = K + 1|z)]\}$$

Nous avons décomposé la cross-entropy totale en faisant apparaître la loss fonction basique $L_{supervised}$ et la loss non-supervisée $L_{unsupervised}$.

La solution optimale pour minimiser les deux est d'avoir $exp(l_j(x)) = c(x)p(y = j, x) \forall j > K + 1$ et $exp(l_{K+1}(x)) = c(x)p_G(x)$ avec c une fonction non déterminée.

Mais notre classificateur avec $K + 1$ outputs est surparamétré et supprimer une fonction $f(x)$ de chaque élément de la sortie ne modifie pas la sortie du softmax.

Dans notre cas, on fixe $l_{K+1}(x) = 0 \quad \forall x$. Ainsi, $L_{supervised}$ devient la loss supervisée standard avec K classes et le classificateur devient $D(x) = \frac{Z(x)}{Z(x)+1}$ avec $Z(x) = \sum_{k=1}^{K} exp[l_k(x)]$

$$L_{unsupervised} = -\{\mathbb{E}_{x \sim p_{data}(x)}[\log[D(x)] + \mathbb{E}_{z \sim G}[\log(1 - D(G(z))\}$$

Cela correspond à la loss d'un discriminateur d'un GAN de base.

3 Baselines

Dans un premier temps, il nous était demandé de réaliser une baseline sur 100 exemples. C'est une méthode d'apprentissage supervisé. Nous avons fait 2 réseaux de neurones : un CNN (adapté pour les images) et un DNN (moins adapté pour les images). Le but d'avoir 2 baselines différentes est de pouvoir comparer les résultats et les améliorations de chaque modèle.

Architecture des réseaux

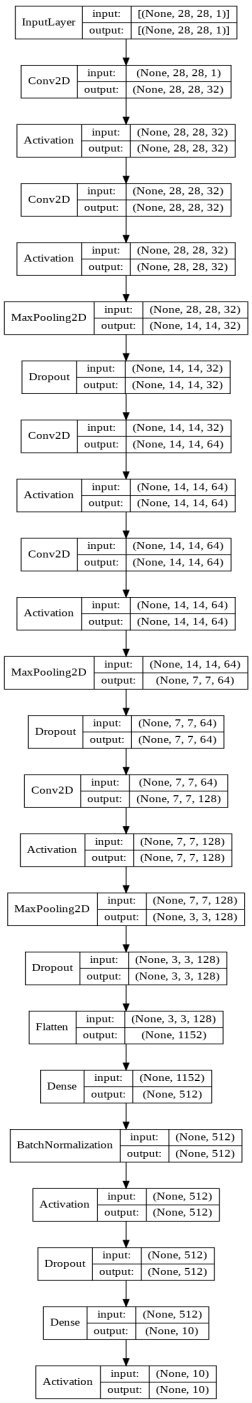

Figure 2: CNN

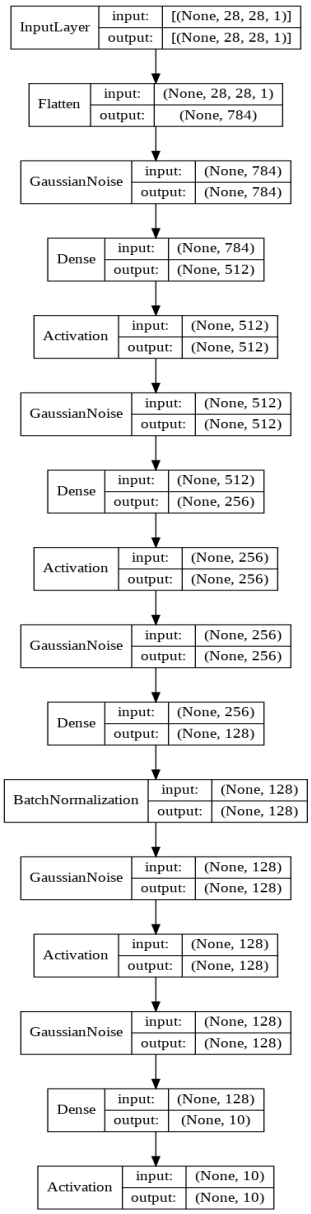

Figure 3: DNN

4 Modèle GAN

4.1 Les Discriminateurs

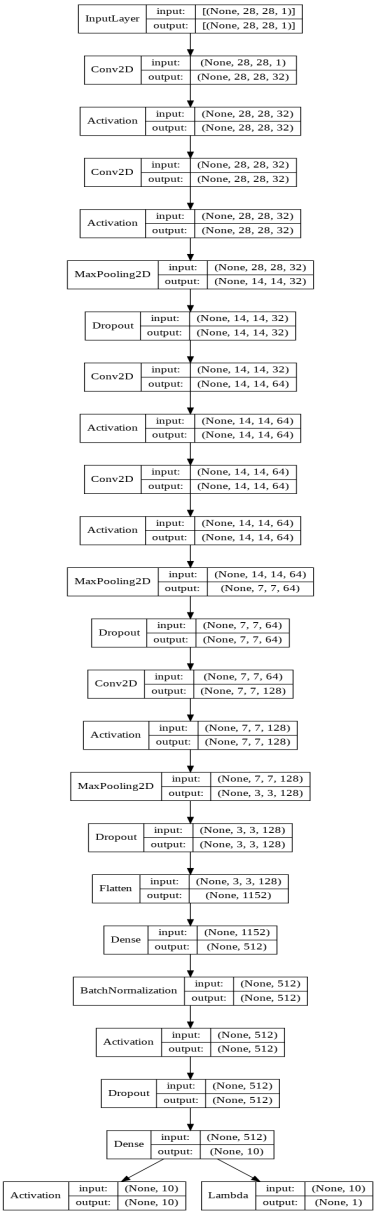

Figure 4: Discriminateur CNN

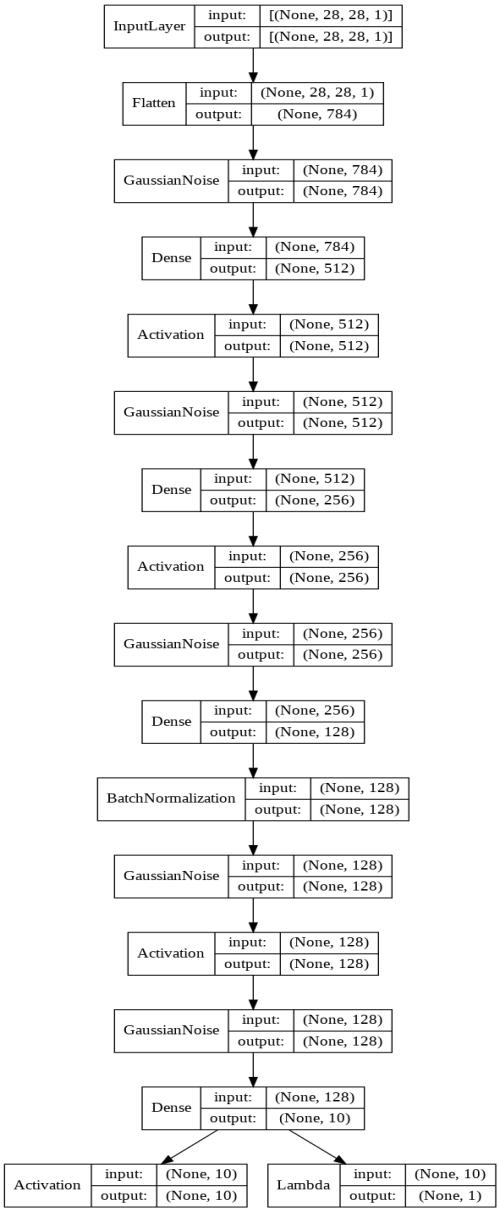

Figure 5: Discriminateur DNN

Maintenant, nous utiliserons toutes les données pour générer notre sortie. On peut remarquer que nos discriminateurs possèdent plusieurs sorties.

La première sortie avec l'activation Softmax va s'occuper de la partie supervisée de notre problème. On ne l'utilisera que pour les 100 données labélisées.

La seconde sortie avec l'activation Lambda correspond à la partie non supervisée du modèle. Notre activation Lambda est définie par $D(x) = \frac{Z(x)}{1+Z(x)}$. Cette sortie sera utile pour toutes les données. (partie non supervisée)

Enfin, notre discriminateur renvoie aussi la valeur d'une couche intermédiaire afin de pouvoir réaliser le feature matching.

Nous avons choisi de prendre comme couche de sortie : celle avant BatchNormalization pour CNN (6e couche avant les sorties) et celle à l'activation avant le GaussianNoise pour DNN (4e couche avant les sorties).

4.2 Generateur

Pour les 2 modèles, nous utiliserons le même générateur. (on garde le même générateur pour pouvoir avoir une meilleure comparaison entre les modèles)

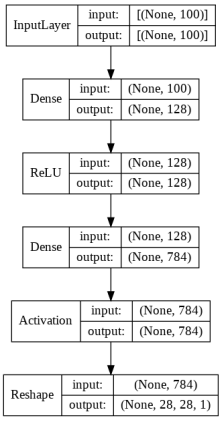

Figure 6: Architecture du générateur

L'entrée du générateur que nous avons choisi est un tenseur de taille 100 généré par $Z \sim \mathcal{U}[0, 1]$

5 Résultats

5.1 Baseline

Pour les baselines, nous avons pris les paramètres suivants :

	optimizer	metrics	epochs	batch size	validation split
CNN	Adam(lr = 0.001)	Accuracy	100	32	0.3
DNN	Adam(lr = 0.001)	Accuracy	50	32	0.3

Table 1: paramètres baseline

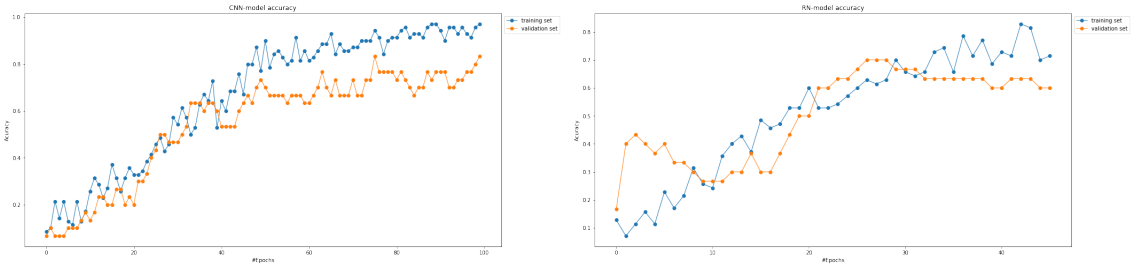

Figure 7: Accuracy des modèles pendant l'entrainement (CNN gauche et DNN droite)

Voici les résultats obtenus sur notre base de test :

On remarque que le CNN est plus performant que le DNN (confirme le fait que le CNN est plus adapté au problème).

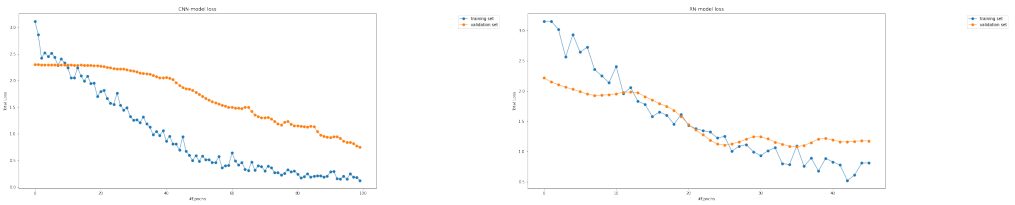

Figure 8: Loss des modèles pendant l'entrainement (CNN gauche et DNN droite)

	Accuracy
CNN	0.8339
DNN	0.6946

Table 2: Résultats Baseline

5.2 Semi-supervised Learning

Pour l'apprentissage semi supervisé, voici les paramètres que nous avons choisi :

	données supervisées	données non supervisées	optimizer	epochs	batch size
CNN	100	60 000	Adam(lr =0.001, $\text{beta}_1 = 0.5$)	10	512
DNN	100	60 000	Adam(lr =0.001, $\text{beta}_1 = 0.5$)	10	512

Table 3: paramètres SSL

Les valeurs d'accuracy ont été obtenues en utilisant directement la base de test

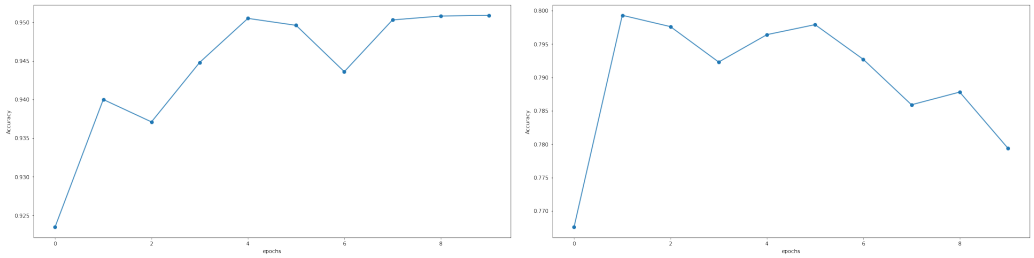

Figure 9: Accuracy des modèles sur l'ensemble test (CNN gauche et DNN droite)

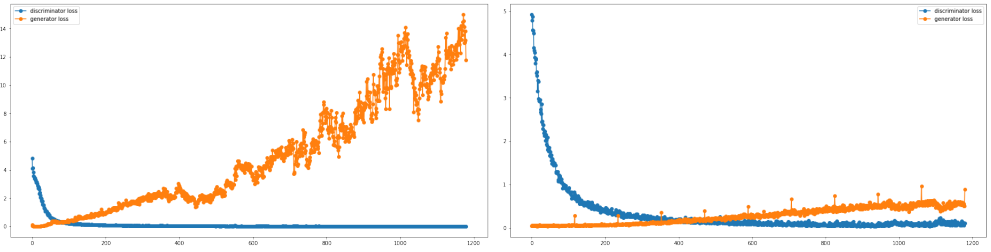

Figure 10: Loss des modèles pendant l'entrainment (CNN gauche et DNN droite)

Voici les résultats obtenus sur notre base de test :
On remarque que le CNN reste plus performant que le DNN.

	Accuracy
CNN	0.9509
DNN	0.7794

Table 4: Résultats SSL

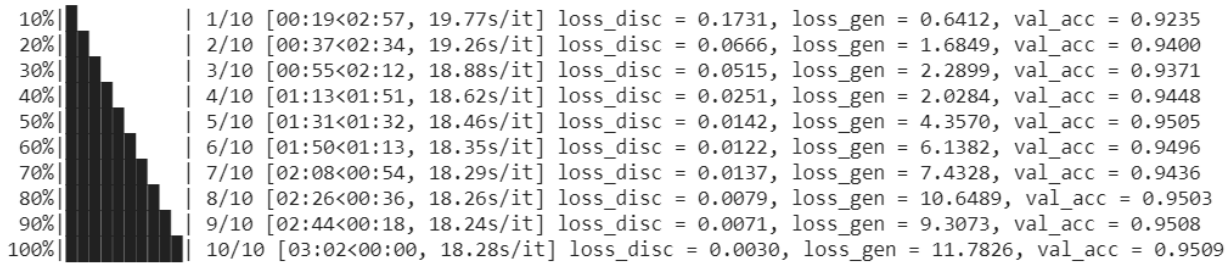

Figure 11: Epochs du modèle avec discriminateur CNN

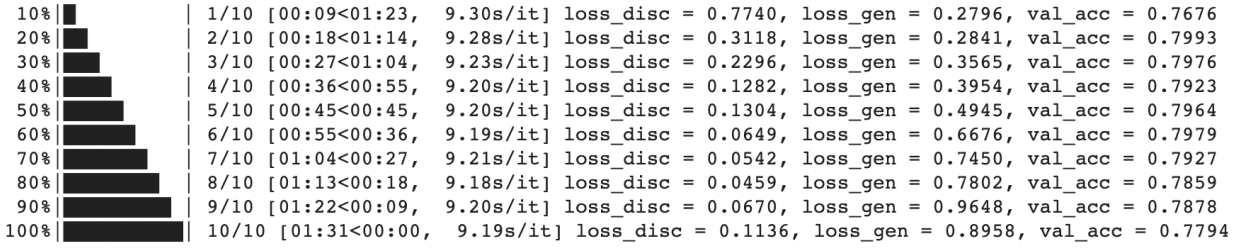

Figure 12: Epochs du modèle avec discriminateur DNN

5.3 Remarques et Limites

Grâce aux résultats, nous pouvons conclure que notre implémentation semi supervisée fonctionne pour les 2 types de réseaux de neurones.

Néanmoins, on remarque que les résultats sont liés aux réseaux utilisés : le CNN reste meilleur que le DNN.

On a vu que pour un même générateur, la méthode d'apprentissage semi-supervisée améliore les résultats peu importe le réseau utilisé initialement. Néanmoins cette amélioration semble limitée : le DNN reste toujours faible.

On peut donc se poser les questions suivantes pour améliorer un apprentissage semi-supervisé :

- Doit-on améliorer le discriminateur ?
- Doit-on améliorer le générateur ?
- Doit-on améliorer les 2 ?

6 Annexe

Veuillez trouver en annexe, le code du projet.