Groupe : ***GP IA***
SCHMIDT--CASANOVA Théo
AROUNAGUIRY Jency

Rapport projet de reconnaissance d'image à partir d'unmodèle pré-entrainé sur le dataset coco

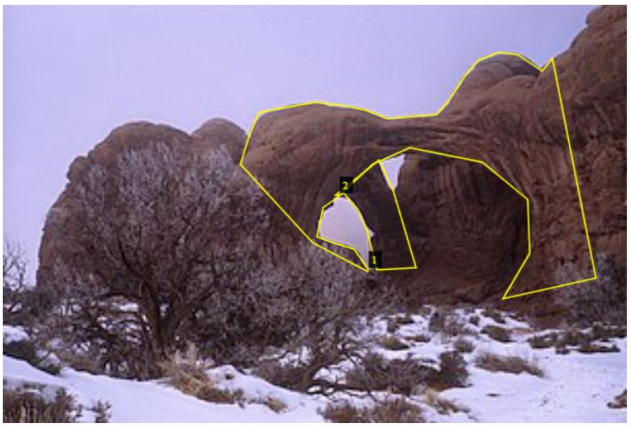
Table des matières

Groupe : ***GP IA***
SCHMIDT--CASANOVA Théo
AROUNAGUIRY Jency

Dans la réalisation de ce projet, où nous devons utiliser un réseau de neurone pré-entrainé sur le dataset coco afin d'entrainer la couche supérieur du réseau sur de nouvelles imagés labellisé afin de reconnaître le nouvel élément qu'on souhaite reconnaître.
Nous avons choisi de tenter de reconnaître les arches d'Utah où nous avons sélectionnés 148 images et labellisés ces dites images.
Quelques exemples ci-dessous montre nos images et la labellisation réaliser sur le site VGG Annotator :

Image sans labellisation

Image labellisé



Groupe : ***GP IA***
SCHMIDT--CASANOVA Théo
AROUNAGUIRY Jency

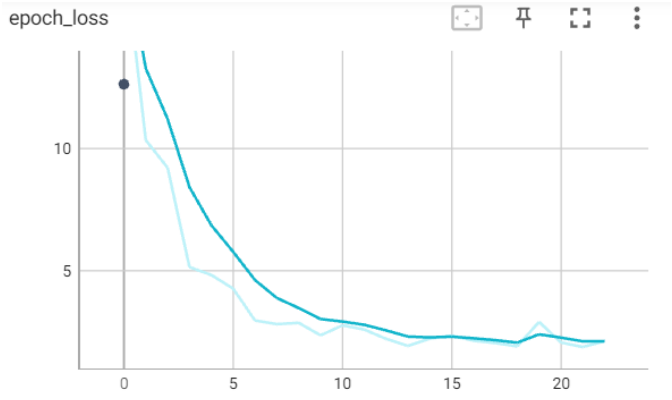
Cette annotation sera stockée dans un fichier json. Nous séparerons nos images dans deux dossier train/val tel que 80 % de nos images seront dans le fichier train et 20 % dans le fichier val. Il y a donc 118 images dans le fichier train et 29 images dans notre fichier val avec les fichiers json d'annotation associé aux images de train et val dans leur dossier respectif.

Nous avons tenté tout d'abord d'utiliser l'implémentation du mask RCNN provenant de « matterport » sur github. Avant de nous rendre compte qu'il y avait des incompatibilités concernant les divers versions des librairies tel que tensorflow, scikit-image. Ainsi après quelques recherches nous avons trouvé une version à jour des librairies python. La seule librairie dont il fallait descendre la version était scikit-image, nous devions baisser à la version 16.2 de la librairie pour éviter tout problème. Cette nouvelle version à jour de matterport provient sur github de « akTwelve » sur le lien suivant : https://github.com/akTwelve/Mask_RCNN.
Après avoir récupéré ces fichiers et un modèle pré-entrainé sur le dataset coco nous avons pu commencé à réalisé nos divers test. Avant d'expliquer le lancement de notre premier entrainement il faut précisé que nous étions sur google colab, qui a quelques limitations technique quant à l'accès d'un GPU notamment, et que certaines fois nos entrainement n'atteignais pas le nombre d'epoch que nous souhaitions. Ainsi nous analysons nos résultats avec des tests qui se sont arrêté en plein milieu certaine fois dû à des limitations techniques de Google Colab.

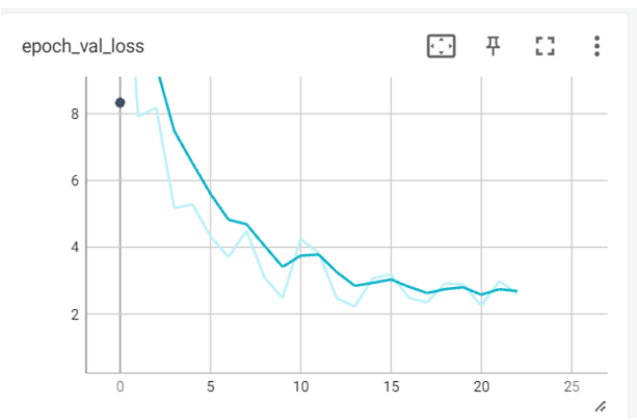
Le premier test

Nous avons lancé deux tests dès le départ. Un premier pour observer le fonctionnement les step per epoch et des epoch. En effet, nous avons directement mit 30 step per epoch et 1 epoch pour le premier test afin de voir ce que cela nous donnait. Ainsi avec 1 epoch, n'avons eu aucun changement au niveau des divers loss obtenu car nous n'avons aucune évolution (le point bleu marine dans les graphique 1 et 2).
Ainsi le second test lancé aura 25 step per epoch, et 22 epoch.
On remarque tout d'abord un loss élevé à 2.118 (Graphique 1), qui doit normalement se rapprocher de 0 pour être efficient.

Graphique 1 – Loss



Graphique 2 – Val_loss



Nous observons aussi que notre val_loss stagne autour des 2.604. Pour l'instant nous allons nous concentrer sur la réduction de notre loss. Cependant notre val_loss nous fait supposer qu'il est possible que nous ayons de l'underfitting. Avant de se prononcer sur la possible présence d'underfitting, nous allons nous concentrer sur la réduction du loss.

Resnet-50

Afin de diminuer le loss, nous connaissons la grande utilité que peut avoir un resnet. Lorsque le signal des poids provenant d'une couche de neurone est faible, le resnet considère que l'information n'était pas suffisamment important pour être considéré par les couches suivantes.

Dans notre cas il est possible d'utiliser deux types de resnet : Le Resnet-101 et le Resnet-50. La principale différence entre ces deux Resnet se situe dans la profondeurs des layers qu'ils utilisent. Le Resnet-101 serait adapté à un grand nombre d'image, mais comme nous n'avons que 148 images au total nous allons utiliser le Resnet-50.

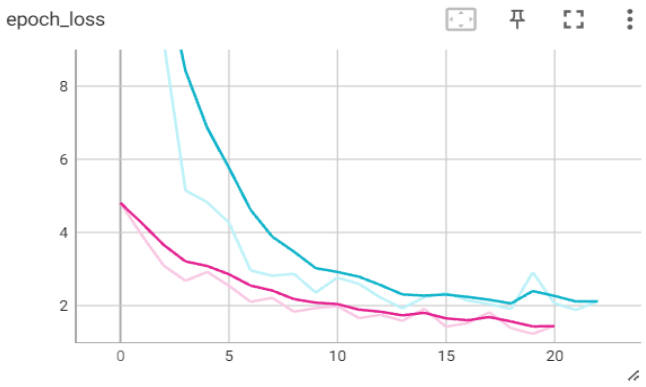
Nous allons tout d'abord appliquer un resnet50, nous observons que notre loss va diminuer mais rester proche de 1 mais pas en-dessous, plus particulièrement autour des 1.24 (Tableau 1).

Tableau 1 – Resultat d'un resnet-50 pour 21 epoch

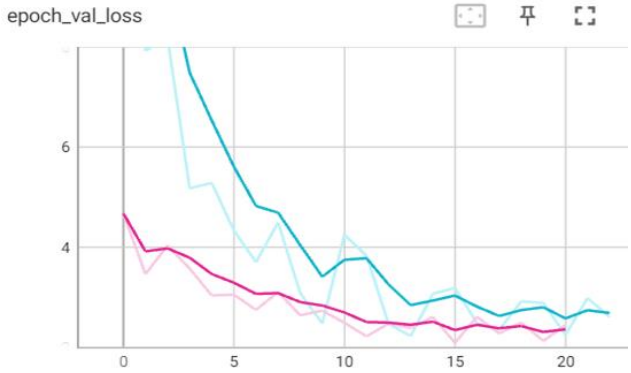
```
Epoch 1/30
30/30 [==============================] - 624s 21s/step - batch: 14.5000 - size: 2.0000 - loss: 5.3793 - rpn_c
Epoch 2/30
30/30 [==============================] - 939s 32s/step - batch: 14.5000 - size: 2.0000 - loss: 3.3000 - rpn_c
Epoch 3/30
30/30 [==============================] - 212s 7s/step - batch: 14.5000 - size: 2.0000 - loss: 3.1921 - rpn_cl
Epoch 4/30
30/30 [==============================] - 913s 31s/step - batch: 14.5000 - size: 2.0000 - loss: 2.9719 - rpn_c
Epoch 5/30
30/30 [==============================] - 205s 7s/step - batch: 14.5000 - size: 2.0000 - loss: 2.1481 - rpn_cl
Epoch 6/30
30/30 [==============================] - 911s 31s/step - batch: 14.5000 - size: 2.0000 - loss: 2.2465 - rpn_c
Epoch 7/30
30/30 [==============================] - 207s 7s/step - batch: 14.5000 - size: 2.0000 - loss: 2.0407 - rpn_cl
Epoch 8/30
30/30 [==============================] - 838s 29s/step - batch: 14.5000 - size: 2.0000 - loss: 2.1143 - rpn_c
Epoch 9/30
30/30 [==============================] - 212s 7s/step - batch: 14.5000 - size: 2.0000 - loss: 1.7403 - rpn_cl
Epoch 10/30
30/30 [==============================] - 917s 32s/step - batch: 14.5000 - size: 2.0000 - loss: 1.8256 - rpn_c
Epoch 11/30
30/30 [==============================] - 200s 7s/step - batch: 14.5000 - size: 2.0000 - loss: 1.6953 - rpn_cl
Epoch 12/30
30/30 [==============================] - 887s 31s/step - batch: 14.5000 - size: 2.0000 - loss: 2.1996 - rpn_c
Epoch 13/30
30/30 [==============================] - 208s 7s/step - batch: 14.5000 - size: 2.0000 - loss: 1.7931 - rpn_cl
Epoch 14/30
30/30 [==============================] - 847s 29s/step - batch: 14.5000 - size: 2.0000 - loss: 1.4229 - rpn_c
Epoch 15/30
30/30 [==============================] - 215s 7s/step - batch: 14.5000 - size: 2.0000 - loss: 1.4372 - rpn_cl
Epoch 16/30
30/30 [==============================] - 855s 29s/step - batch: 14.5000 - size: 2.0000 - loss: 1.5451 - rpn_c
Epoch 17/30
30/30 [==============================] - 206s 7s/step - batch: 14.5000 - size: 2.0000 - loss: 1.5535 - rpn_cl
Epoch 18/30
30/30 [==============================] - 821s 28s/step - batch: 14.5000 - size: 2.0000 - loss: 1.3462 - rpn_c
Epoch 19/30
30/30 [==============================] - 213s 7s/step - batch: 14.5000 - size: 2.0000 - loss: 1.4173 - rpn_cl
Epoch 20/30
30/30 [==============================] - 785s 27s/step - batch: 14.5000 - size: 2.0000 - loss: 1.3388 - rpn_c
Epoch 21/30
30/30 [==============================] - ETA: 0s - batch: 14.5000 - size: 2.0000 - loss: 1.2481 - rpn class l
```

En comparaison de notre test précédent (courbe en bleu clair du graphique 3 et 4) le Resnet nous permet de diminuer le loss de notre nouveau test (courbe en rose du Graphique 3 et 4). De même pour notre val_loss dans le graphique 4.

Graphique 3 – Loss



Graphique 4 – Val_loss



Ainsi nous conserverons le Resnet-50 dans tous les prochains tests que nous réaliserons car il nous permettra constamment de gagner du temps mais aussi de gagner en information pertinente de nos poids dans notre fonction de coût.

2 types de tests : méthode d'optimisation du gradient descent et modification de la régularisation du L2 et divers paramètre

Nous réaliserons 2 tests en parallèle afin de trouver divers moyens pour diminuer notre loss et val_loss. Des tests concernant la modification de la régularisation L2 et donc de la suppression de nombreux poids dans notre fonction de loss et de la modification de divers paramètres . Et secondement des tests concernant la modification de la méthode d'optimisation du gradient descent et de son learning rate.

Parlons tout d'abord des modifications menées sur la méthode d'optimisation du gradient descent.

Modification de la méthode du gradient descent

La méthode qui est utilisé de base est le SGD. Nous savons pertinemment que la méthode Adam est meilleur car elle mêle le SGD et Adagram. Mais avant réalisons le test avec Adagram.

Groupe :
SCHMIDT--CASANOVA Théo
AROUNAGUIRY Jency

Adagram

Nos paramètre de tests sont :

Step per epoch	Epoch	Resnet-50	Méthode du gradient descent	Learning rate
25	20	Oui - Utilisé	Adagrad	0.001

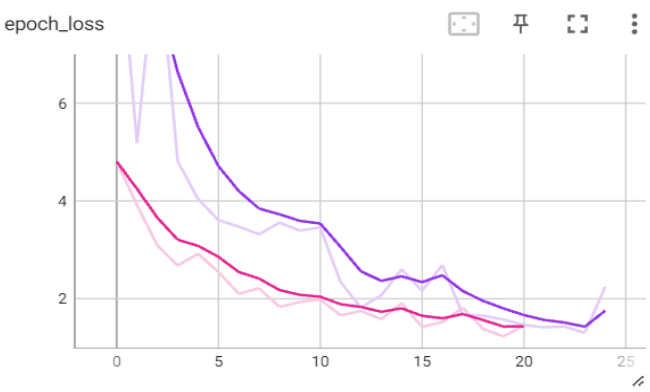
En réalisant le test avec Adagrad on obtien un loss pire que tout ceux qu'on a pu avoir pour l'instant. Même du premier test. En effet, le loss atteint à peu près 20 dans les premières epoch et descend de la même manière que la méthode SGD. Ainsi nous passons directement à l'utilisation de la méthode Adam sans tenter de modifié le learning rate pour Adagrad.

Adam

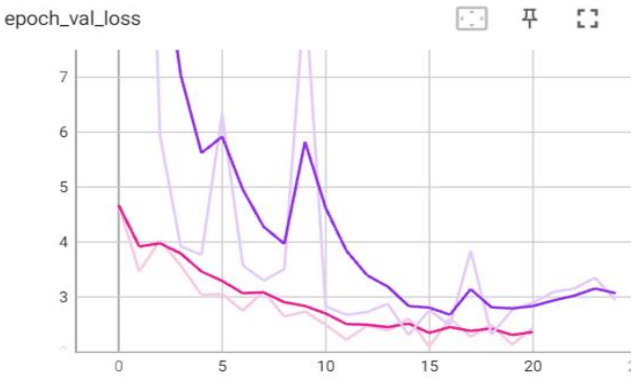
Step per epoch	Epoch	Resnet-50	Méthode du gradient descent	Learning rate	beta_1	beta_2
25	24	Oui - Utilisé	Adagrad	0.001	0.9	0.99

Un premier test avec le learning_rate de base nous montreras qu'il n'y avait pas tant de différence que ça avec le loss et le val_loss du SGD (Graphique 5 et 6). Ce qui nous pousse donc à modifier notre learning rate et à le baisser énormément afin de vérifier s'il existe des résultats différents.

Graphique 5 – Loss



Graphique 6 – Val_loss



Groupe : ***GP IA***
SCHMIDT--CASANOVA Théo
AROUNAGUIRY Jency

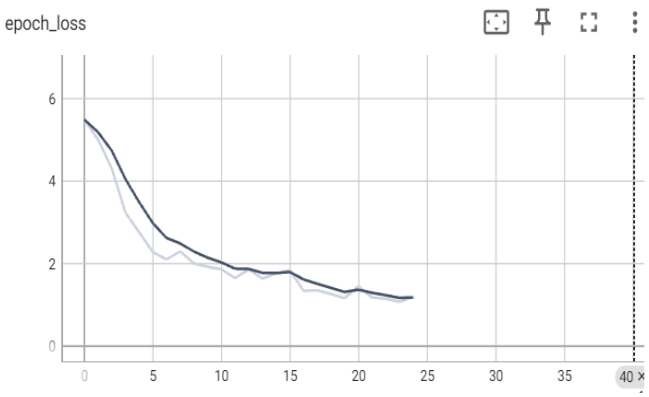
On voit sur nos graphique 5 et 6 que notre nouveau test avec notre méthode Adam (courbe en violet) que notre loss se rapproche de notre test avec le SGD (courbe en rose). Modifions le learning rate et beta_1 et observons si cela change notre loss et val_loss.

Step per epoch	Epoch	Resnet-50	Méthode du gradient descent	Learning rate	beta_1	beta_2
100	24	Oui - Utilisé	Adagrad	$1*10^{-4}$	0.99	0.99

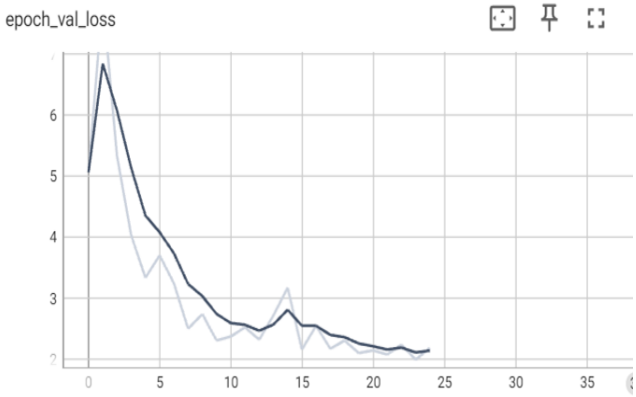
En modifiant le learning rate de 0.001 à $5*10^{-4}$ et beta_1 de 0.9 à 0.99, nous modifions aussi le nombre de step per epoch de 25 à 100 afin de permettre à notre modèle de trouver le point le plus faible dans notre fonction de cout.

Au sein des graphiques 7 et 8 nous observons que nous obtenons le loss le plus faible possible. Nous avions tenté d'atteindre les 40 epoch mais le procédé à pris trop de temps et nous avons eu un problème technique qui nous a fait arrêter à 24 epoch. Mais cela reste une plus forte diminution avec un loss à 1.194 à l'epoch 24 et un val_loss à 2.186.

Graphique 7 – Loss



Graphique 8 – Val_loss



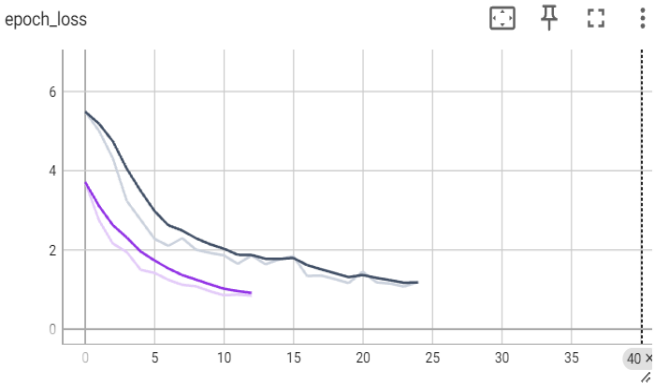
Groupe :
SCHMIDT--CASANOVA Théo
AROUNAGUIRY Jency

Nous tentons de réaliser un dernier test avec la modification du learning rate afin de le diminuer une nouvelle fois. De plus pendant la réalisation de l'entrainement de notre modèle le loss pouvait atteindre un minimum et augmenter à cause du fait qu'on avait trop de step per epoch.

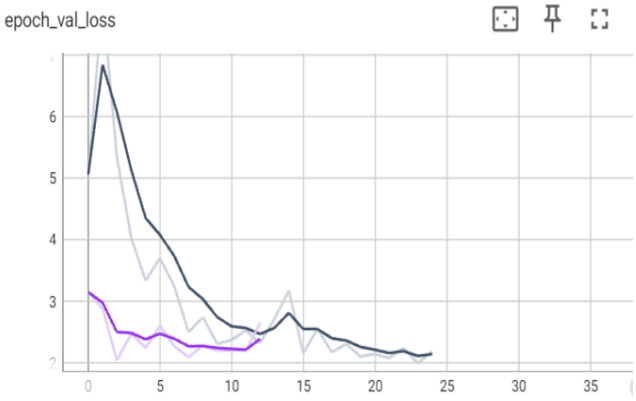
Step per epoch	Epoch	Resnet-50	Méthode du gradient descent	Learning rate	beta_1	beta_2
75	12	Oui - Utilisé	Adagrad	$5*10^{-5}$	0.99	0.99

Après modification du nombre de step per epoch de 100 à 75, et du learning rate de $1*10^{-4}$ à $5*10^{-5}$ nous trouvons un loss plus faible dès les 12 premières epoch. En effet, le procédé s'est arrêté en plein milieu car google colab à bloqué l'accès aux serveurs GPU en plein entrainement, donc nous devons tenir des conclusions de nos 12 epoch au lieu des 40 espéré.

Graphique 9 – Loss



Graphique 10 – Val_loss



La courbe violette représente notre nouveau test, en comparaison de l'ancien en bleu marine. Le loss le plus faible est celui de notre nouveau test à 0.91 environ à l'epoch 12 et le val_loss est à 2.389. On peut commencer à supposer qu'il y ait un problème d'overfitting car malgré la diminution durant les divers epoch de notre loss, notre val_loss commence à oscillé. L'underfitting restait possible auparavant car notre val_loss suivait notre loss dans sa diminution, désormais il reste constant et oscille, voire il augmente. On peut, pour l'instant, supposé que nous sommes en présence d'overfitting

En parallèle des tests étaient réalisé autour des modifications de certains paramètre.

Groupe :
SCHMIDT--CASANOVA Théo
AROUNAGUIRY Jency

Modification de la régularisation L2 et des paramètres

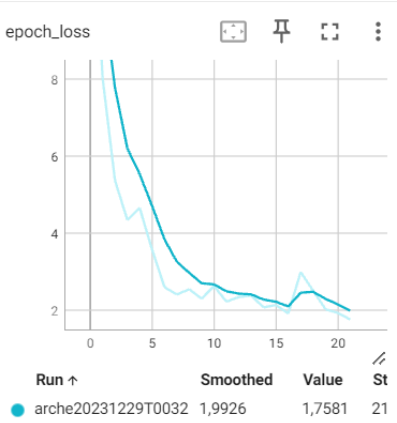
Régularisation L2

Depuis le début une régularisation L2 était déjà présent de base dans le modèle à 0.0001. Nous tentons de le modifier pour voir si cela nous permet d'obtenir de meilleur résultat.

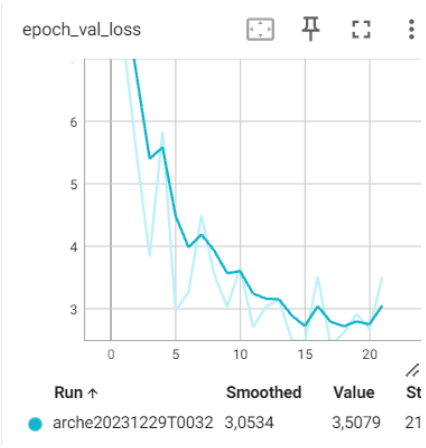
Step per epoch	Epoch	Resnet-50	Régularisation L2
30	20	Oui - Utilisé	0.01

En modifiant le poids de la régularisation L2 à 0.01 on observe que notre loss et val_loss vont diminuer par rapport au test du resnet. Nous atteignons un loss à 1.75 (Graphique 11) mais un val_loss à 3.5 (Graphique 12) en 20 epoch. Ce qui signifie qu'il y a une légère amélioration mais rien de bien surprenant par rapport à ce que nous avions obtenu avec le resnet-50.

Graphique 9 – Loss



Graphique 10 – Val_loss



Finalement un test où nous modifions le poids de la régularisation à 0.02 va nous donner un loss qui va être encore plus mauvais que notre test précédent. Ainsi la modification de la régularisation dans ce cas ne permet pas d'obtenir un meilleur loss.

Modification de divers paramètres :

Après quelques tests et des erreurs techniques nous menant à un nombre d'epoch peu élevé. Nous avons réussi à modifié les paramètres concernant : la dimension des images, Les régions d'intérêt, le nombre de validation step, le learning_rate, et les layer entrainé.

Groupe : ***GP IA***
SCHMIDT--CASANOVA Théo
AROUNAGUIRY Jency

La dimension des images à été modifié pour la dimension des images maximum et minimum de 1024 et 800, respectivement, à 128 pour les deux. Cela nous permet de concentré nos images de sélectionner les éléments qui semblent essentiel à notre modèle et les plus pertinent.

Les régions d'intérêts ont aussi été modifié. En effet, nos images ne présentant pas énormément de complexité et peu d'objet à reconnaître, nous avons diminué ce paramètre de 200 à 32. Nous avons considéré qu'il n'y avait pas besoin de présenter autant de région d'intéret sur nos images.

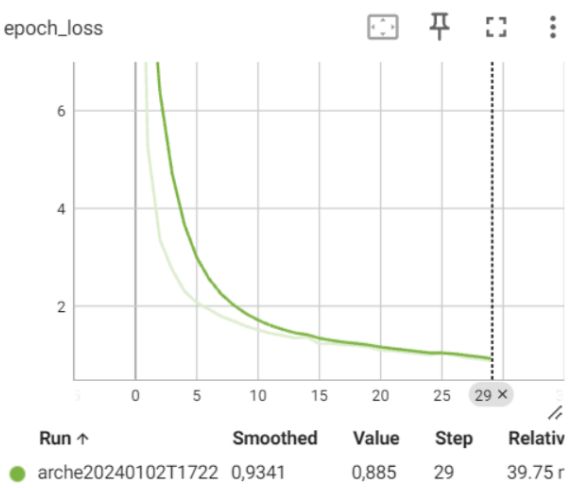
Le nombre de validation_step à été diminué de 50 à 5 pour deux raisons : un gain de temps concernant nos test mais aussi parce que nous avons peu d'image, il ne semblait pas nécessaire d'avoir autant de validation step. Nous reviendrons tout de même, plus tard, sur cette modification afin de s'assurer que cela n'impacte pas réellement nos résultats (dans les prochaines sections).

Finalement le learning_rate est passé de 0.001 à $1*10^{-12}$ afin de diminuer le nombre de saut dans notre fonction de cout et un entrainement sur tous les layers pour que toutes les données soit pris en compte par notre modèle.

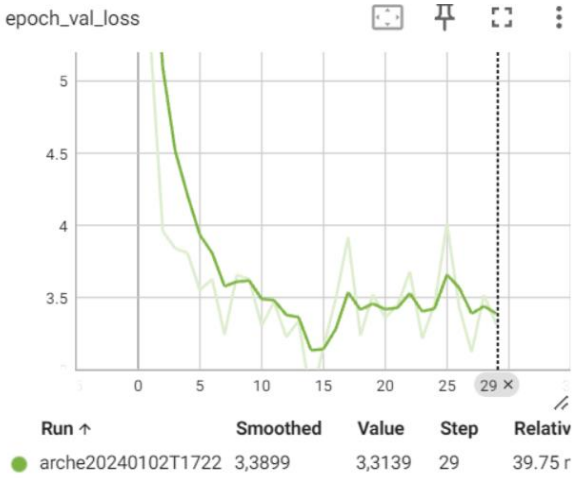
Step per epoch	Epoch	Resnet-50	Méthode du gradient descent	Learning rate	ROIs	IMAGE_MAX_DIM et IMAGE_MIN_DIM	Validation_step	Layers entrainé
30	29	Oui - Utilisé	SGD	$1*10^{-12}$	32	128	5	All

Ainsi nous observons avec tous nos paramètres modifiés une énorme amélioration qui nous permet de passé notre loss sous 1 (Graphique 11), et plus exactement à 0.885. Finalement notre val_loss est à oscille, voire augmente (Graphique 12), notre supposition sur l'overfitting se renforce.

Graphique 11 – Loss



Graphique 12 – Val_loss



Test finaux

Nous avons donc pu en extraire des paramètres qui font diminuer notre loss et nous font en déduire un possible overfitting.
Nous allons donc réaliser nos derniers tests avec des paramètres qui nous ont sembler optimal pour diminuer le loss.

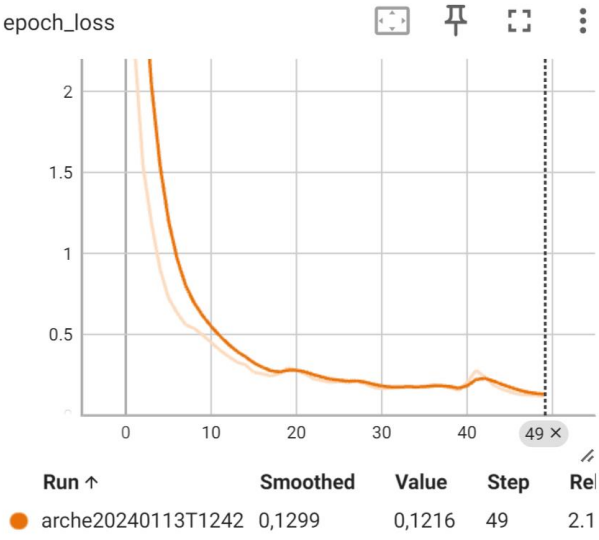
Ainsi tout nos paramètres seront les suivants :

Step per epoch	75
Epoch	50
Resnet-50	Oui - Utilisé
Méthode du gradient descent	Adam
Learning rate	$5*10^{-5}$
ROIs	32
IMAGE_MAX_DIM et IMAGE_MIN_DIM	128
Validation step	5
Layers entrainé	All
Régularisation L2	0.0001

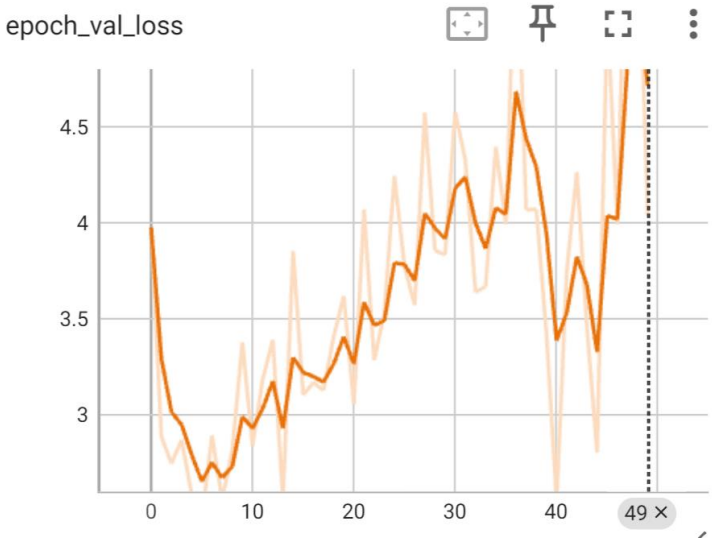
Après avoir correctement configuré nos paramètres nous lançons l'un de nos dernier test.

Groupe :
SCHMIDT--CASANOVA Théo
AROUNAGUIRY Jency

Graphique 13 – Loss



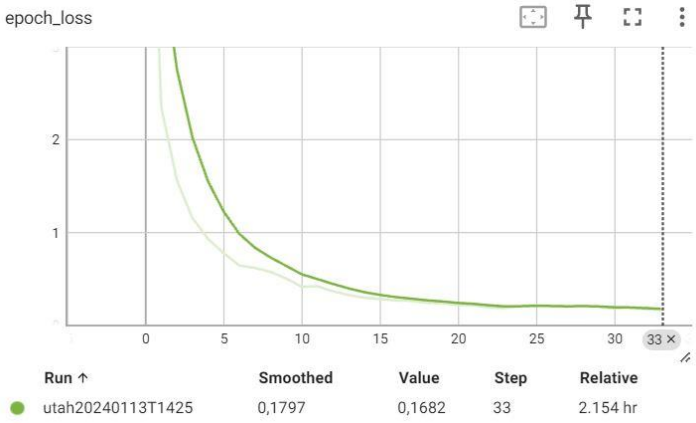
Graphique 14 – Val_loss



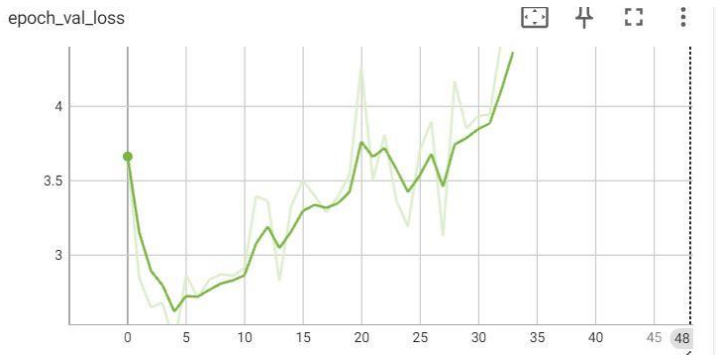
On observe Une réelle diminution de notre loss à partir de ces paramètres. Cependant il y a une réelle présence d'overfitting dans notre modèle comme nous l'indique le val_loss qui oscille et augmente énormément.

Et pour s'assurer que notre validation step n'impact pas notre dernier test nous pouvons vérifier en augmentant le nombre de validation step à 33.
Et on observe que notre val_loss augmente (Graphique 15 et 16) et oscille encore une fois. Ainsi il y a réellement la présence d'overfitting malgré les divers changement que nous avons effectué pour diminuer le val_loss avec le loss.

Graphique 15 – Loss



Graphique 16 – Val_loss



Conclusion

Ainsi dans cet entrainement de notre modèle avec un fichier pré-entrainé sur le dataset coco nous avons pu ajouter un resnet-50 qui diminuait notre loss un peu. Nous remarquions un loss et val_loss élevé que nous tentions de réduire.

Notamment au travers de divers test effectué parallèlement :

- Par une modification de la méthode d'optimisation du gradient descent et du learning rate qui nous mènera à utiliser la méthode Adam et un learning rate très faible et proche de 0 avec un nombre de step per epoch à 75.
- Une modification des paramètres tel que la dimension maximum et minimum de nos image, le nombre de région d'intéret, les layers qui seront entrainé et la régularisation L2

Ces divers tests nous ont amené à un ensemble de paramètre qui nous permettent de réduire fortement le loss et se rapprocher de 1. Cependant nous ne réussissons pas à réduire le val_loss, signe de notre overfitting et de notre modèle.

Nous pouvons supposer que pour résoudre ce problème il faudra se pencher sur notre base de données ou notre labellisation. Peut-être en ajoutant des images d'arche d'Utah nous aurait permis de résoudre ce problème. Ou finalement modifié les images en les rendant plus complexe pour que le modèle s'adapte aux images du fichier validation.