

ConvNets

Le principe d'utilisation d'un ConvNets reste les données à multiples dimensions. Dans notre cas les données sont des images c-à-d des données à deux dimensions. Le convnets est le réseau qui s'adapte le mieux aux images puisque celui-ci s'inspire du cortex visuel. Comme tout réseau la topologie est importante. On n'a pas chercher ici à prendre le meilleur réseau puisque l'idée d'un réseau est de l'optimiser et le but est de comprendre comment les hyperparamètres peuvent changer un réseau et ses résultats.

Ici on a créé un réseau initial et ce réseau va nous servir de base dans notre étude et en voici les paramètres : Les paramètres et le réseau choisis sont empirique et constitue une base d'étude.

Réseau initiale : Topologie

In []:

```
NUM_LAYERS = list(range(3, 6))
BATCH_SIZE = 512
EPOCHS = 300
SHUFFLE = True
learning_rates = 0.001
momentums = 0.9

def ConvNet(num_layer, filters_by_layers):
    input_layer = Input(shape=(32, 32, 3))
    hidden_layers = input_layer

    for n in range(num_layer):
        hidden_layers = Conv2D(filters_by_layers[n], (3, 3), padding="same", activation="tanh")(hidden_layers)
        hidden_layers = MaxPool2D()(hidden_layers)

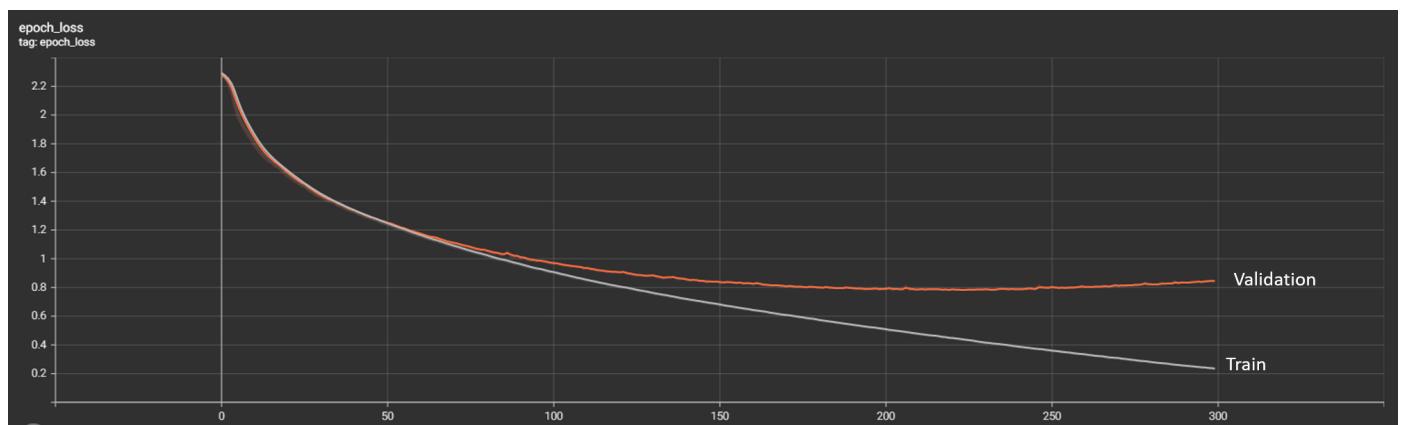
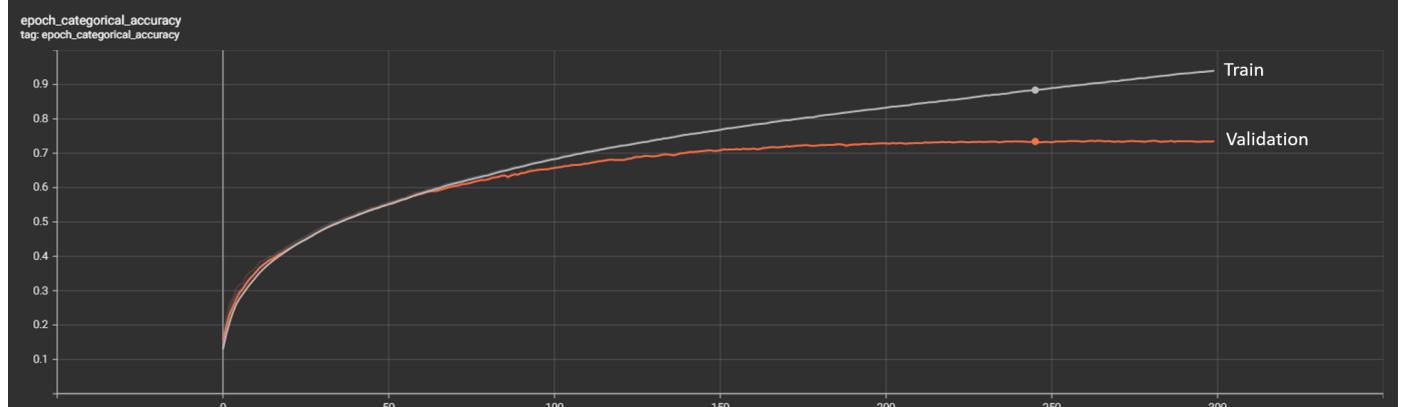
    hidden_layers = Flatten()(hidden_layers)
    output_layer = Dense(NUM_CLASSES, activation=softmax)(hidden_layers)
    return Model(input_layer, output_layer)

num_layer = int(np.random.choice(NUM_LAYERS, 1))
filters_by_layers = [32*x for x in range(1, num_layer+1)]

convnet = ConvNet(num_layer, filters_by_layers)
convnet.compile(loss=categorical_crossentropy,
                 optimizer=SGD(lr, momentum=mom),
                 metrics=categorical_accuracy)
CONVNET_LOG = os.path.join(LOG_DIR, "convnet",
                           f"convnet_ep_{EPOCHS}_bs_{BATCH_SIZE}_opt_SGD_lr_{lr}_mom_{mom}_layers_{num_layer}")
CONVNET_MODEL = os.path.join(MODEL_DIR, "convnet",
                           f"convnet_ep_{EPOCHS}_bs_{BATCH_SIZE}_opt_SGD_lr_{lr}_mom_{mom}_layers_{num_layer}")

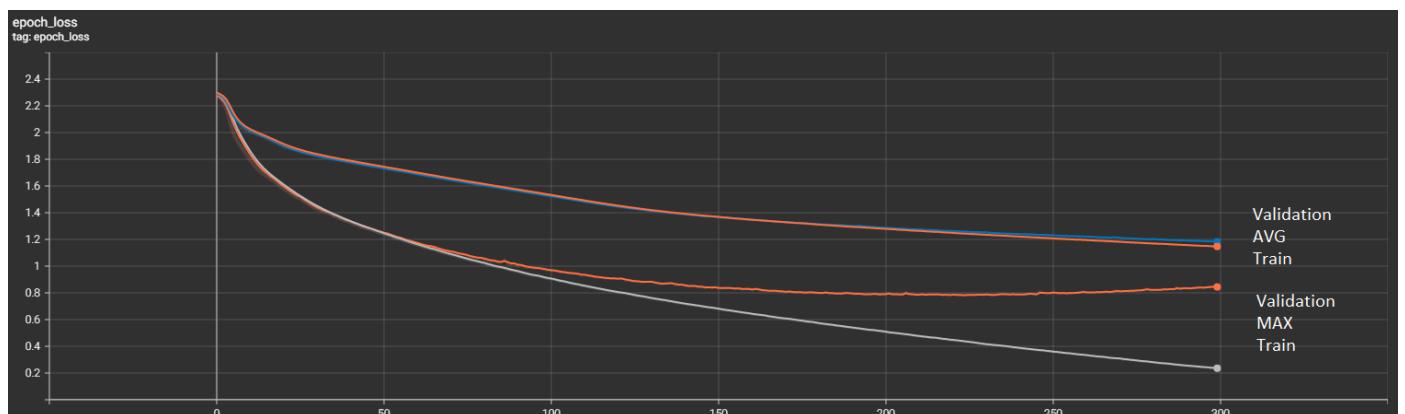
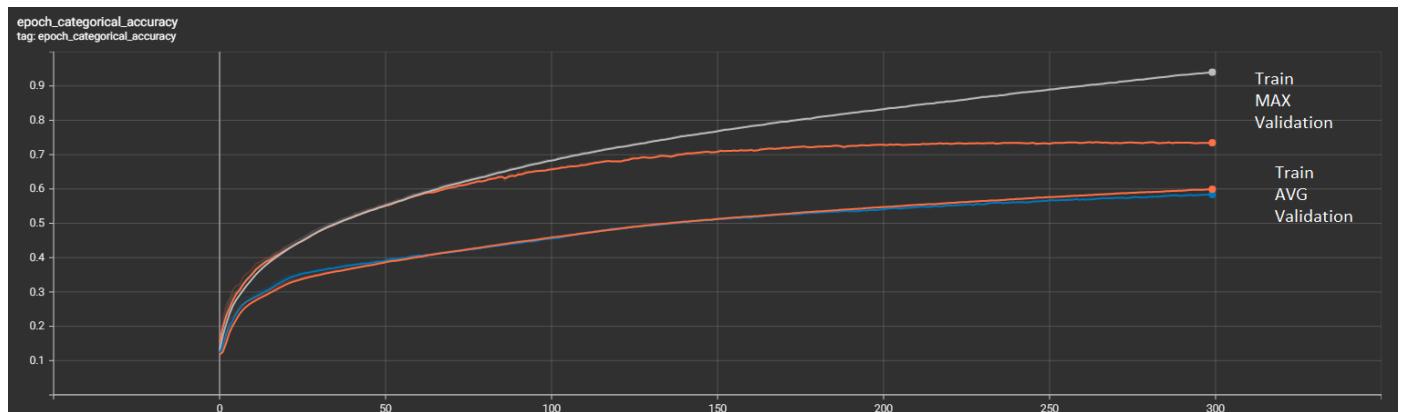
convnet.fit(x_train,
            y_train,
            batch_size=BATCH_SIZE,
            epochs=EPOCHS,
            validation_data=(x_test, y_test),
            shuffle=SHUFFLE,
            callbacks=[tf.keras.callbacks.TensorBoard(CONVNET_LOG, histogram_freq=1)])
convnet.save(CONVNET_MODEL)
```

Réseau initiale : courbe



Les courbes ci dessus montre l'apprentissage :accuracy en haut et loss en bas du réseau de base qui va nous servir de comparatif. Les courbes sont lisses sans dents de scie et une croissance progressive (qui s'affiche par une courbe plus arrondi sur le début de l'apprentissage) sans décroissance. On peut observer entre l'epochs 150-200 du surapprentissage sur les courbes d'accuracy et de loss.

Max pooling vs Average pooling



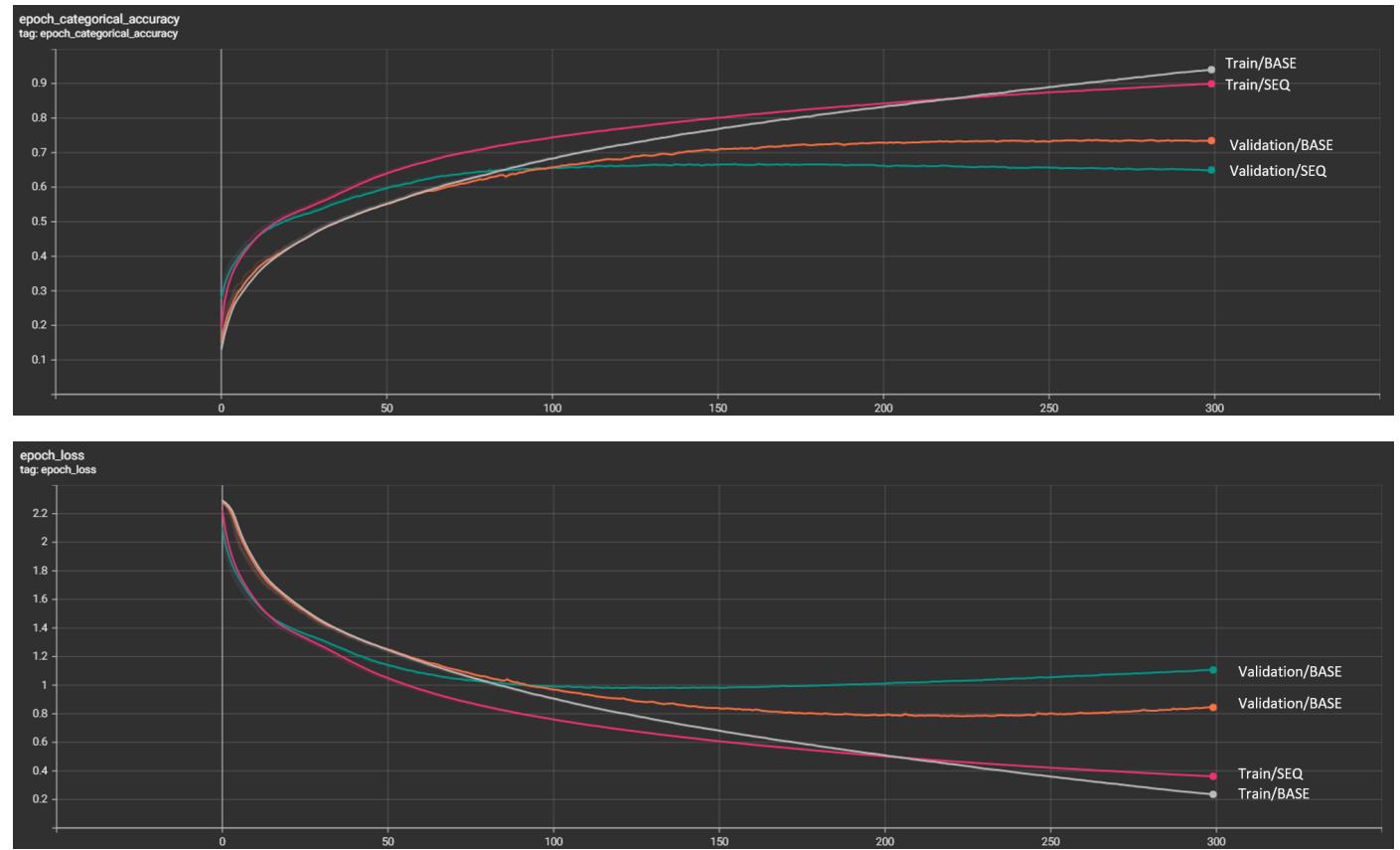
Nous allons comparer deux types de couche utilisés dans les convnet : le max pooling et l'average pooling

la courbes du dessus représente l'accuracy entre l'AVG et le MAX pooling et en dessous de la loss.s

Lorsque l'on regarde les deux courbes on remarque qu'elles ont des similitudes : une courbe lisse et stable sans dents de scie, un croissance progressive sans décroissance. Néanmoins des différences sont visibles : la courbe du réseau initial présente une accuracy plus forte avec une différence de ~10% d'accuracy supplémentaire mais un surapprentissage notable à partir de la 200ème epoch. Lorsque l'Average pooling est utilisé, l'accuracy y est moins grande mais elle ne présente pas de surapprentissage. Sur les courbes de loss le même constat y est observable : une loss plus grande mais sans surapprentissage pour l'AVG pooling contrairement aux courbes de MAX pooling.

Comment peut on expliquer cette différence : que se passe t-il dans les couches ? Max pooling prend la valeur maximum de la portion d'image couverte par le kernel et AVG pooling sa valeur moyenne. La différence se tient ici. Etant donné que les valeurs prises ne sont pas les mêmes alors la capacité à capturer l'information va changer aussi. Les caractéristiques des deux layers vont drastiquement influer sur les résultats. Il est possible que l'utilisation d'un MAX pooling peut provoquer un manque de généralisation dans la compréhension des données et du problème.

Séquentiel vs Fonctionnal



Nous allons comparer ici deux réseaux qui utilisent les mêmes caractéristiques avec fonction d'activation optimizer learning rate et momentum identique, seul la topologie va changer. Nous avons un réseau séquentiel et un réseau fonctionnel. L'accuracy est au dessus et la loss en dessous.

Lorsque l'on regarde l'apprentissage des deux réseaux on remarque que les deux courbes sont similaires dans leur stabilité mais aussi dans leur surapprentissage. Les deux réseaux présentent exactement les mêmes progressions. On peut en conclure que la topologie d'un réseau influe moins sur les résultats que la modification des hyperparamètres du réseau.

Réseau Séquentiel

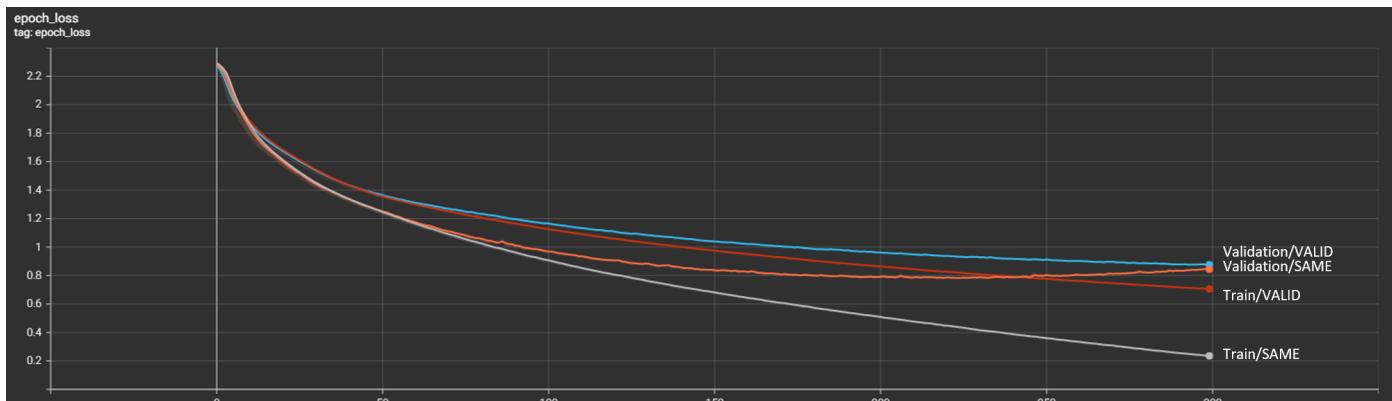
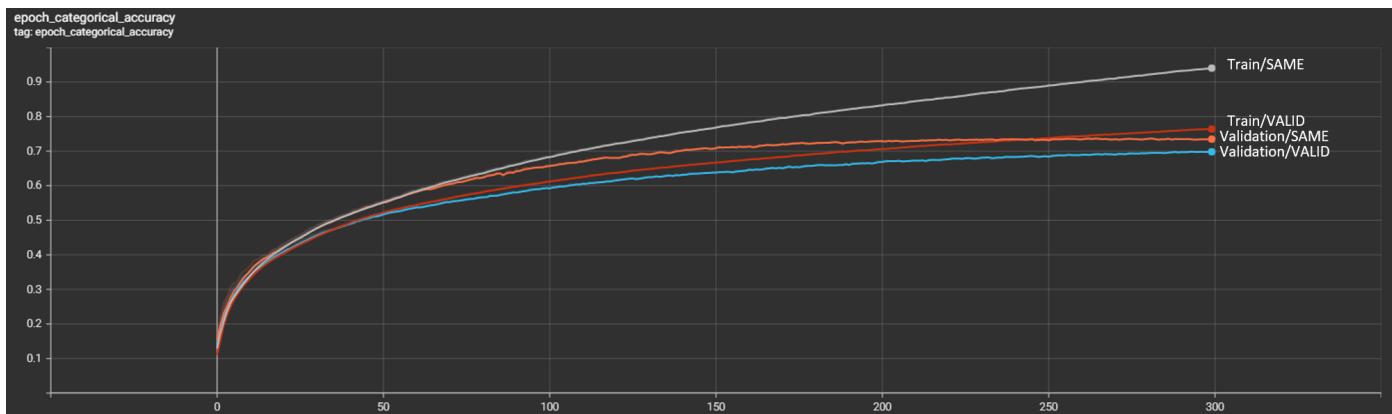
In []:

```
BATCH_SIZE = 512
EPOCHS = 300
SHUFFLE = True
learning_rates = 0.001
momentums = 0.9

convnet = Sequential()
convnet.add(Conv2D(128, input_shape=(32, 32, 3), kernel_size=(3, 3), padding="same", activation="relu"))
convnet.add(Conv2D(64, kernel_size=(3, 3), padding="same", activation=tanh))
convnet.add(Conv2D(32, kernel_size=(3, 3), padding="same", activation=tanh))
convnet.add(MaxPool2D())
convnet.add(Flatten())
convnet.add(Dense(NUM_CLASSES, activation=softmax))

convnet.compile(loss=categorical_crossentropy,
                 optimizer=SGD(learning_rates, momentum=momentum),
                 metrics=categorical_accuracy)
CONVNET_LOG = os.path.join(LOG_DIR, "convnet",
                           f"convnet_sequential_ep_{EPOCHS}_bs_{BATCH_SIZE}_MAXPOOLING_opt_SGD_lr_{learning_rate}_momentum_{momentum}_shuff_{SHUFFLE}_epoch_{EPOCHS}_log")
CONVNET_MODEL = os.path.join(MODEL_DIR, "convnet",
                           f"convnet_sequential_ep_{EPOCHS}_bs_{BATCH_SIZE}_opt_SGD_lr_{learning_rate}_momentum_{momentum}_shuff_{SHUFFLE}_epoch_{EPOCHS}_model.h5")
convnet.fit(x_train,
            y_train,
            batch_size=BATCH_SIZE,
            epochs=EPOCHS,
            validation_data=(x_test, y_test),
            shuffle=SHUFFLE,
            callbacks=[tf.keras.callbacks.TensorBoard(CONVNET_LOG, histogram_freq=1)])
convnet.save(CONVNET_MODEL)
```

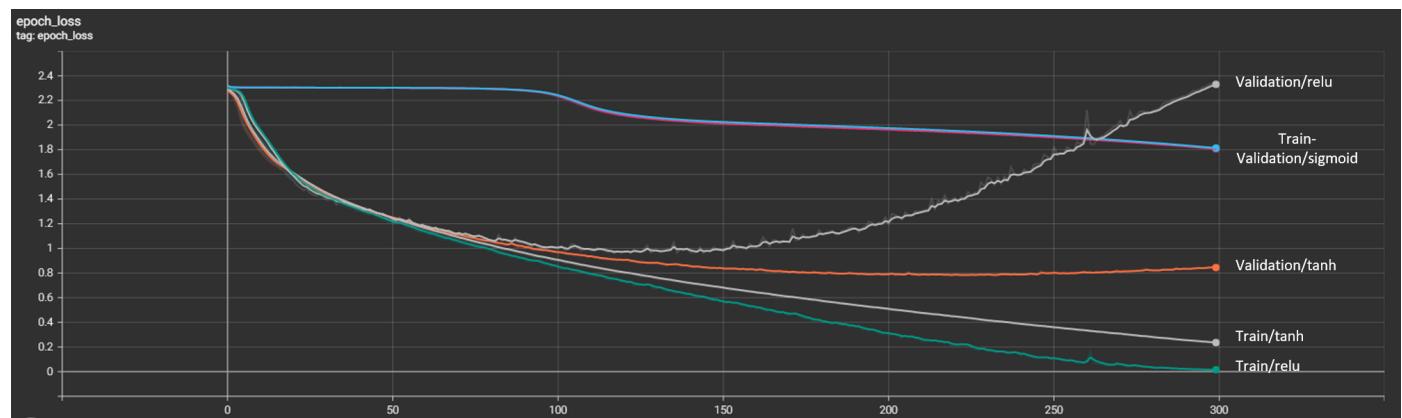
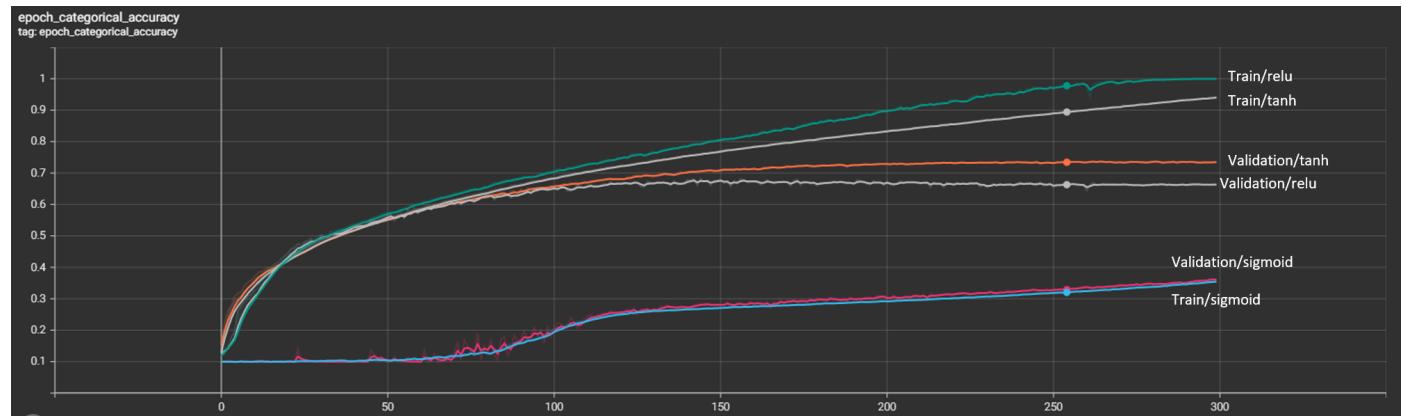
Padding = same vs Padding = valid



Nous allons comparer deux versions d'un même paramètre qu'est le padding avec le padding en "same" ou en "valid". La courbe d'accuracy est au dessus et de loss en dessous. Lorsque l'on regarde l'apprentissage des deux réseaux on remarque que les deux courbes sont similaires dans leur stabilité. Le réseau avec un padding "VALID" présente un plus faible écart avec la courbe de validation, contrairement à celle du padding "SAME" qui présente un surapprentissage plus accerbé. La courbe de loss présente la même tendance que pour l'accuracy : une décroissance moins élevée pour le padding "VALID" mais sans surapprentissage.

Comment peut on expliquer cette différence : que se passe t-il avec le padding ? Le principe du padding est dans notre cas de compenser le manque d'information présent sur les bords d'une image lorsque le kernel passe. Dans le cas où le padding est en "valid" l'image est conservée telle qu'initiallement donné au modèle. Cela implique que si le kernel arrive en bord d'image, il est possible d'avoir une perte d'information due à la dimension du kernel et de l'image. Dans l'autre cas donc "same", des valeurs vont être ajoutées de part et d'autre de l'image pour conserver les bords et donc conserver de l'information. Cela peut se remarquer car pour deux réseaux similaires, en début d'apprentissage, on peut voir que l'accuracy ne croît pas autant entre les deux versions. Cette perte d'information permet au réseau ayant le paramètre padding="valid" d'avoir une meilleure généralisation et donc moins de surapprentissage puisque celui n'obtient pas l'ensemble de l'information de l'image. Néanmoins l'utilisation d'un padding "VALID" réduit le nombre de bonne prédiction du réseau.

Fonction d'activation tanh relu sigmoid



Nous allons comparer 3 fonctions d'activations que sont tanh relu et sigmoid sur l'ensemble du réseau. Les courbes des différentes fonctions d'activations sont inscrites sur l'image pour plus de clarté avec au dessus l'accuracy et en dessous la loss.

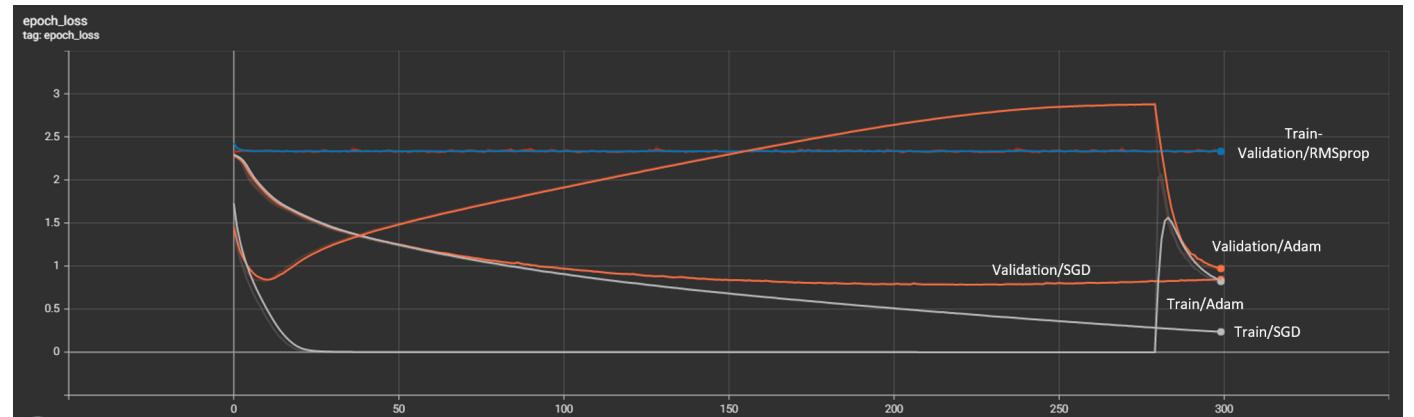
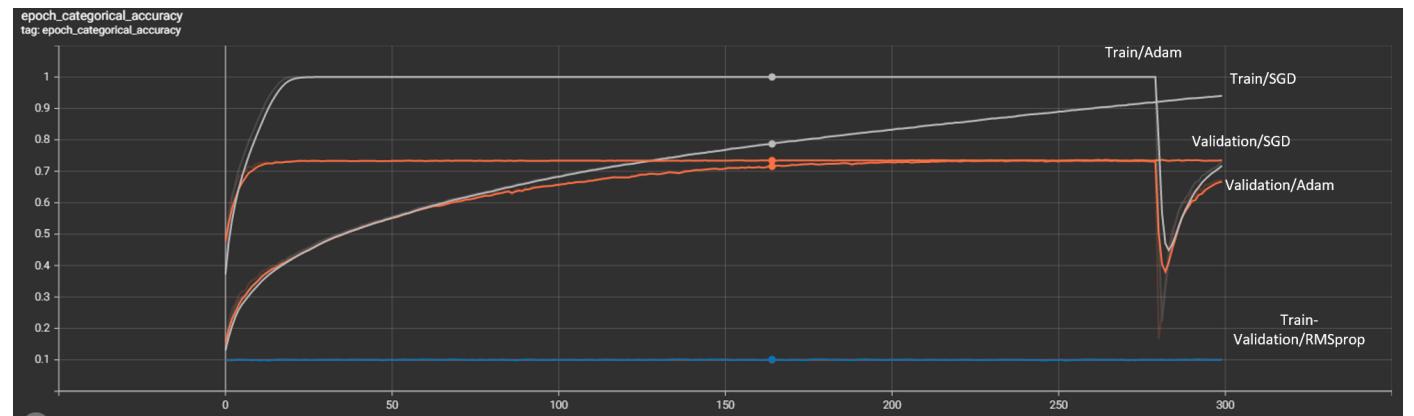
Deux profils de courbe peuvent être observer sur l'image :

- Le premier est celui de la fonction sigmoid qui présente une accuracy basse et quelques dents de scie mais sans surapprentissage
- Le deuxième est celui de tanh et de relu avec une croissance forte (plus forte pour ReLu) relativement stable mais avec un aussi fort surapprentissage pour les deux fonctions.

Les courbes de loss présentes les mêmes types de profils :

- Sigmoid avec une décroissance faible mais stable
- Tanh et ReLu en dent de scie et une nette croissance pour ReLu qui croît après l'epoch 100

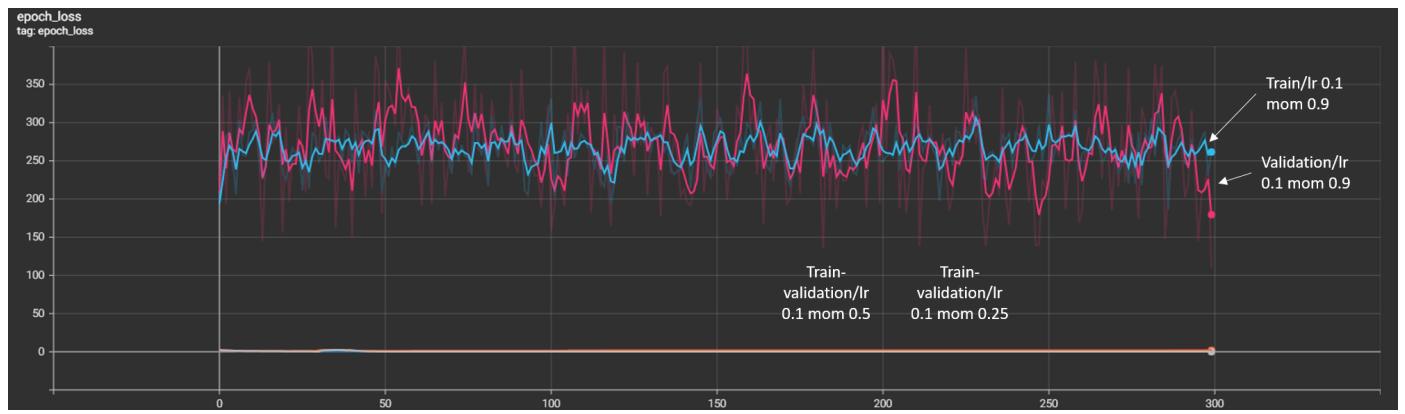
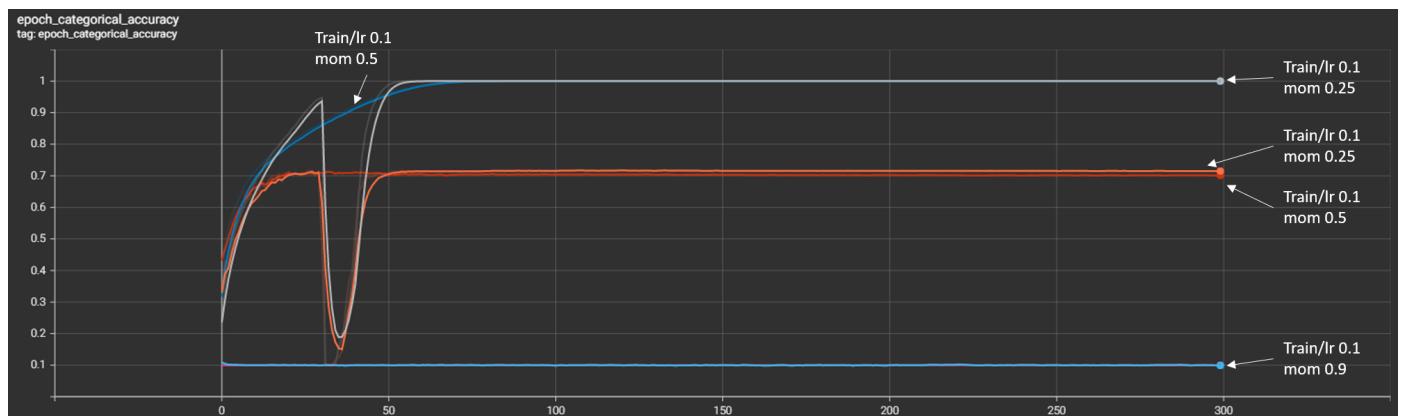
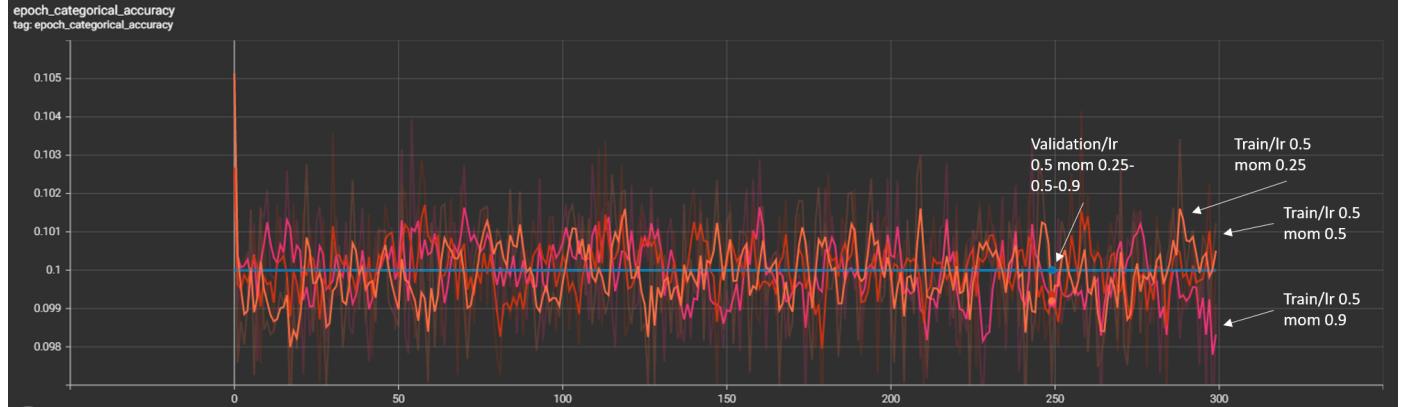
Optimizer SGD ADAM RMSprop

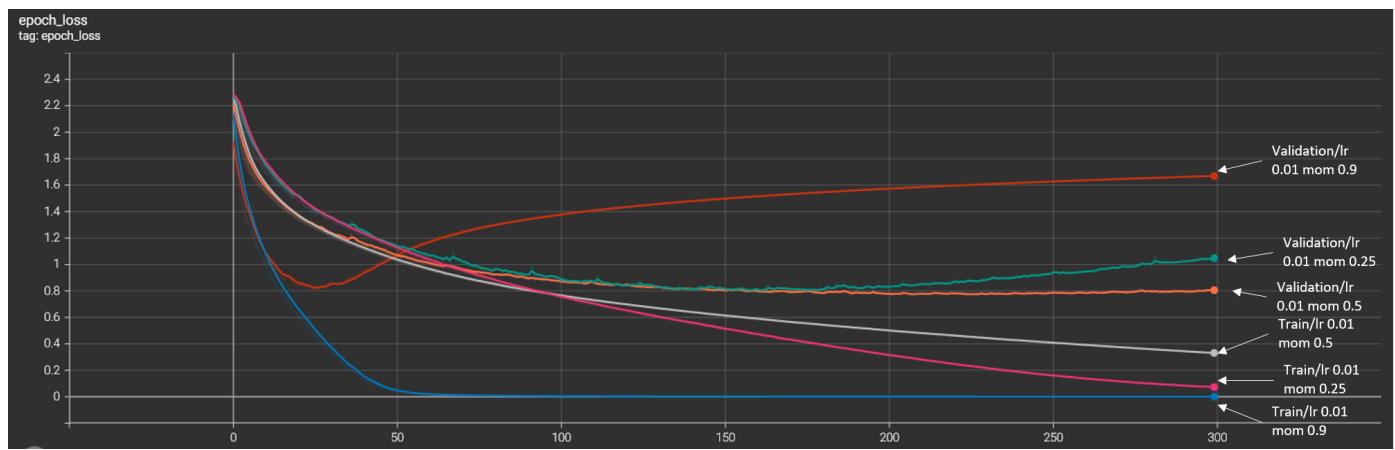
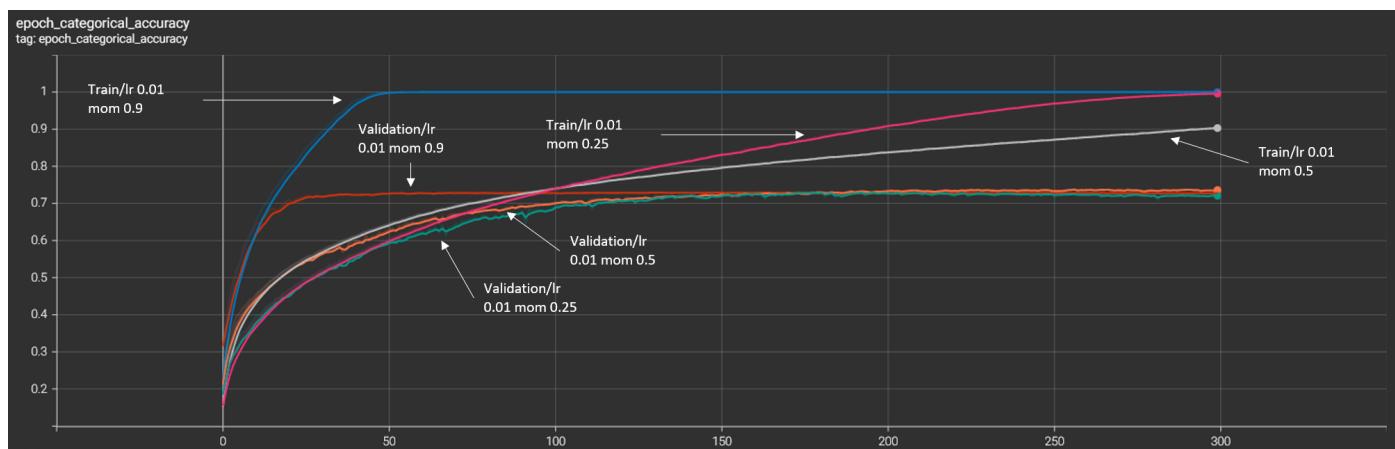
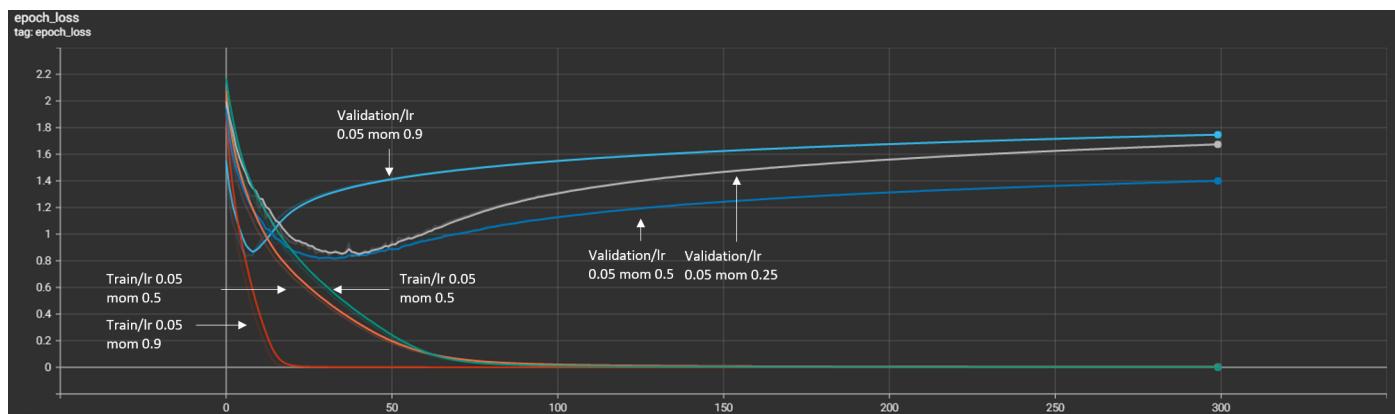
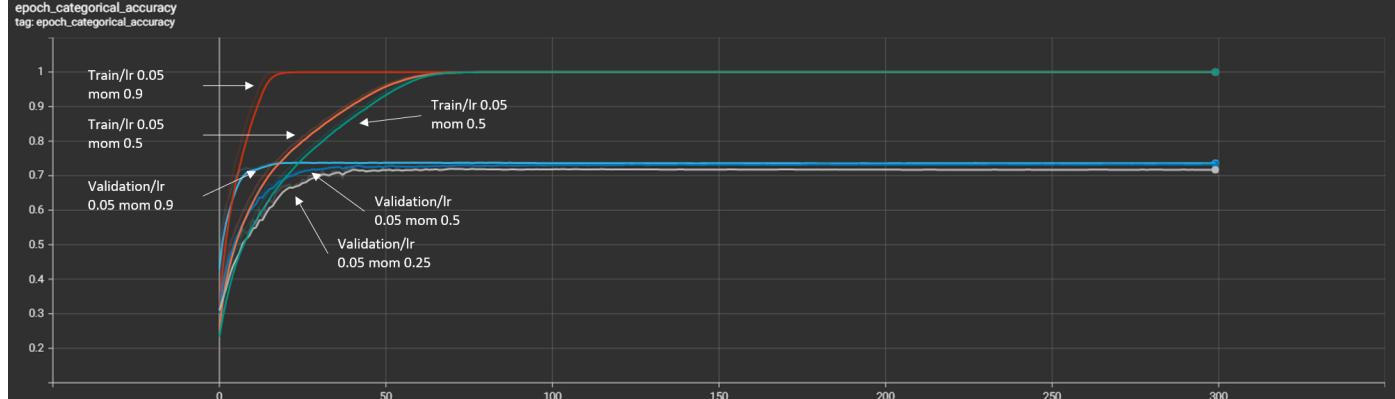


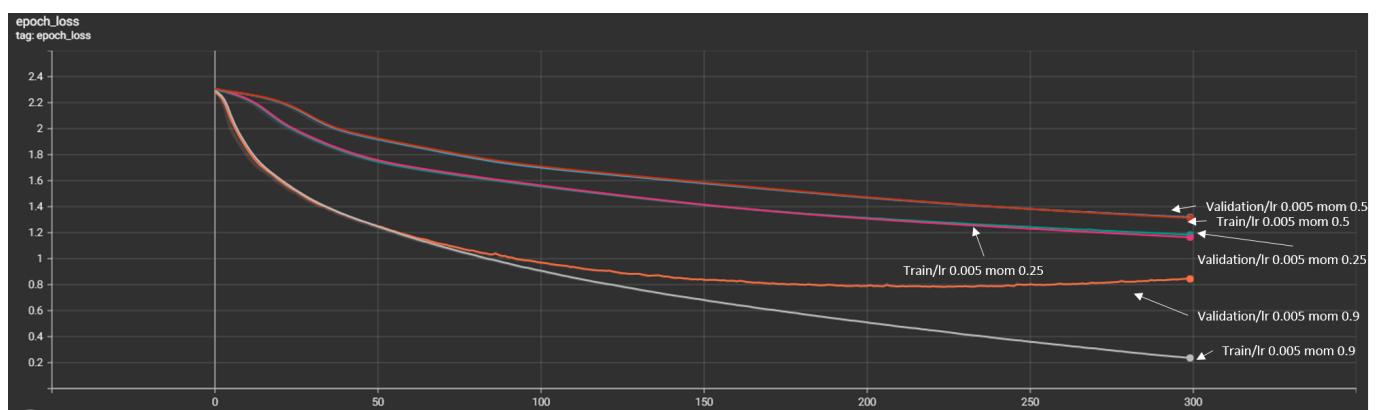
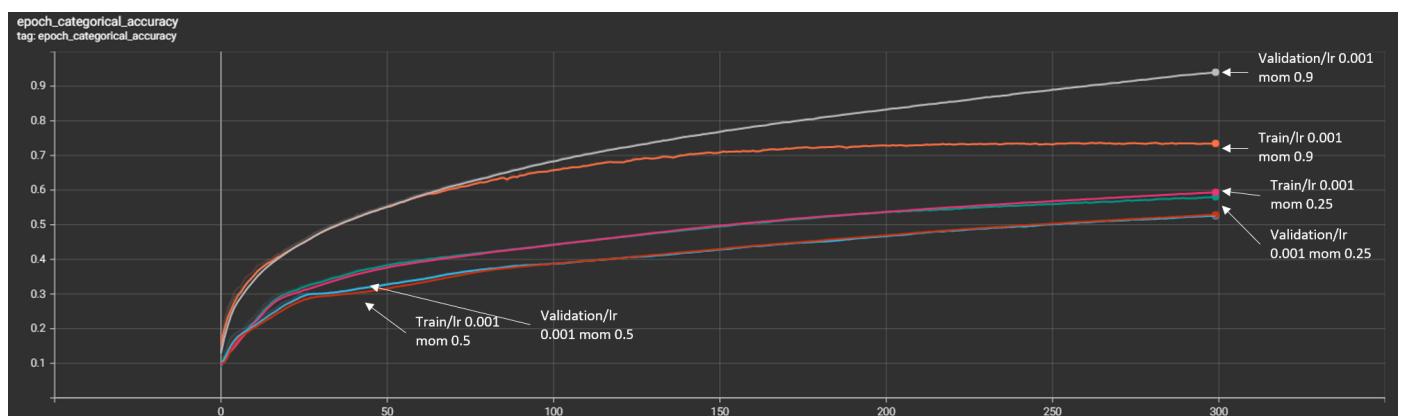
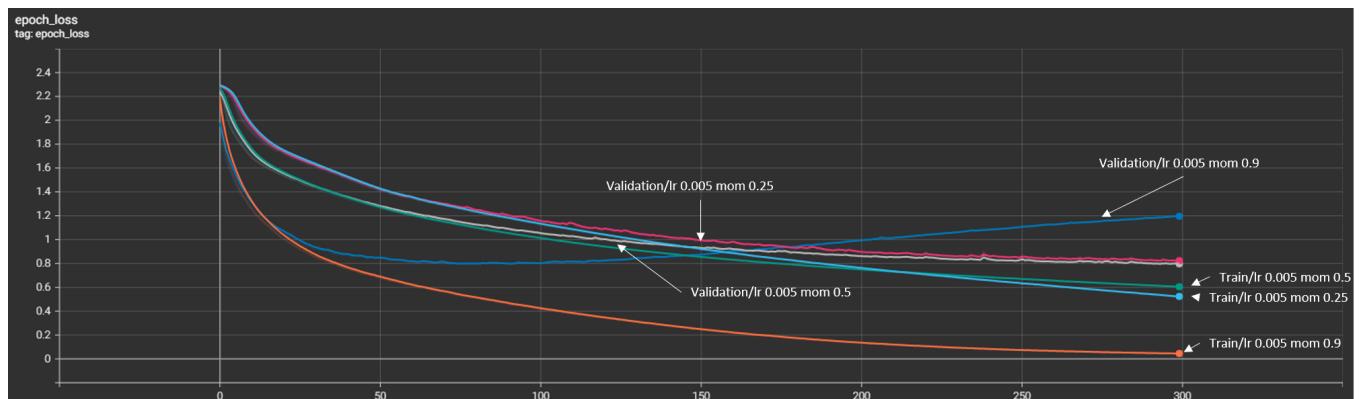
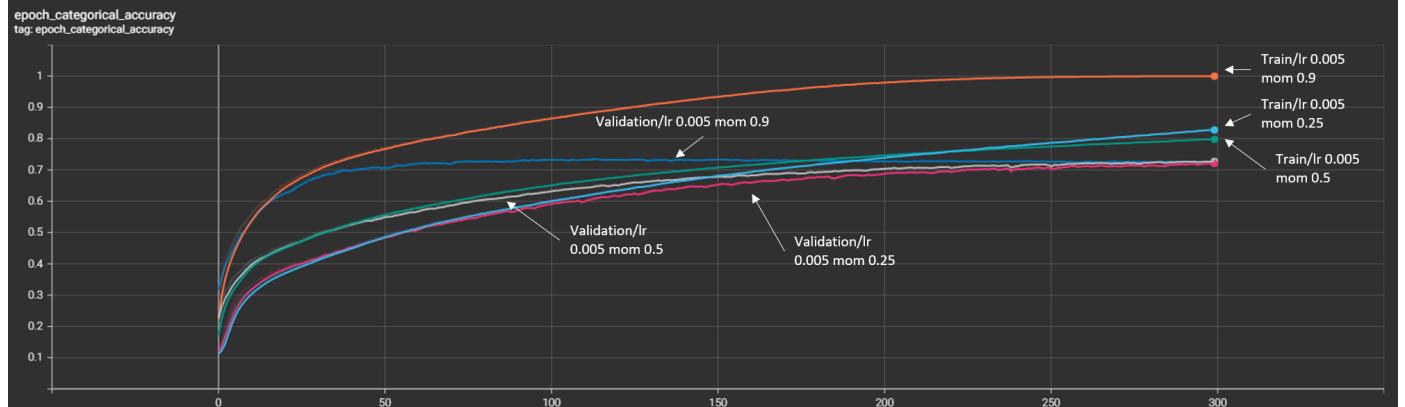
Nous allons comparer 3 optimiseurs que sont SGD Adam et RMSprop. On peut observer 3 types de courbes différents avec l'accuracy au dessus et la loss en dessous :

- RMSprop qui montre une croissance en dent de scie (ici ne s'affiche pas mais en zoom oui) et un manque de croissance en accuracy et de décroissance en loss
- Adam qui montre une croissance très forte et monte très vite sur les 100% d'accuracy mais présente tout aussi rapidement un fort surapprentissage avant même de dépasser les 50 epochs. On peut aussi observer une chute sous forme de pic vers l'epoch 350 en accuracy sur la validation et sur le train. La courbe de loss quant à elle présente la même tendance avec une forte décroissance pour arriver à 0 de loss puis un pic de croissance pour remonter à environ 1 de loss sur le train et la validation.
- SGD quant à lui montre un profil avec courbe sans dents de scie, du surapprentissage vers la fin sans pics de croissance ou décroissance sur l'accuracy ou la loss

Learning Rate et momentum







Ordre des images par learning rate : 0.5, 0.1, 0.05, 0.01, 0.005, 0.001

Nous allons ici étudier l'impact du learning rate sur l'apprentissage ainsi que le momentum. On peut diviser cela en plusieurs parties : [0.5, 0.1, 0.05, 0.01, 0.005, 0.001] pour le learning rate et [0.25, 0.5, 0.9] pour le momentum

- Pour 0.5 de learning rate et [0.25, 0.5, 0.9] de momentum:

L'accuracy ne dépasse pas les 10% avec une progression en dents de scie pour l'accuracy sur la courbe de de train et une stagnation sur la partie validation. La loss est

- Pour 0.1 de learning rate et [0.25, 0.5, 0.9] de momentum:

Deux profils peuvent être vu : le premier avec des croissances allant jusqu'à 100% d'accuracy et un profil qui ne dépasse pas les 10%. Les deux profils similaires sont vu lorsque le momentum est à 0.25 et 0.5. On peut observer que sur les courbes un fort surapprentissage, une progression stable et un fort pic avant l'epoch 50 qui diminue drastiquement l'accuracy mais qui croît de nouveau est observé. Cette tendance est en miroir avec la loss qui elle aussi présente le même type de pic : forte décroissance, pic de croissance, et redécroissance. Le profil avec l'accuracy ne dépassant pas les 10% est lorsque le momentum est à 0.9. L'accuracy décroît, est en dents de scie mais sans surapprentissage.

- Pour 0.05 de learning rate et [0.25, 0.5, 0.9] de momentum:

Les trois courbes présentes des profils similaires : accuracy forte avec 100% atteint avant l'epoch 80 sur la courbe de train, courbe stable et un fort surapprentissage. Les courbes de validation ne dépassent pas les 75% d'accuracy. La loss présente pour les trois courbes présentes aussi des profils similaires avec une forte baisse sur la courbe de validation. Un pic plus prononcé est visible sur la courbe au learning de 0.05 et de momentum 0.9, alors que pour les deux autres il s'agirait plutôt d'un puit mais qui dans les trois cas remontent. On peut remarquer que sur la courbe de loss de train que la décroissant est le même dans sa tendance mais pas dans sa progression. Plus le momentum est fort plus la courbe à tendance à fortement diminuer puis rejoint un plateau beaucoup plus que pour des valeurs faibles.

- Pour 0.01 de learning rate et [0.25, 0.5, 0.9] de momentum:

Cette fois ci on remarque concernant la partie accuracy que les courbes sont plus tassés. Sur la courbe de train celle ci sont stable sans dents de scie avec un convergence plus rapide pour un lr de 0.01 et momentum 0.9. La croissance de la courbe avec lr 0.01 et momentum 0.25 est moins fort que pour lr 0.01 et momentum 0.5 mais celle ci finit pas la dépasser. Les trois profils présentes un surapprentissage. Les courbes de validation sont moins stable et ne dépasse pas les 75% d'accuracy mais contrairement au train, elles convergent tous au même point c-à-d 75% d'accuracy. Les courbes de loss sont miroir en train par rapport au courbe de l'accuracy mais pas en validation : une décroissance sous forme de puit est visible puis une forte croissance avec un lr de 0.01 et momentum de 0.9. Cette tendance se fait voir aussi avec un lr de 0.01 et un momentum de 0.25 qui recroît en fin d'apprentissage. Seul la courbe au lr de 0.01 et de momentum de 0.5 décroît constamment.

- Pour 0.005 de learning rate et [0.25, 0.5, 0.9] de momentum:

Ici on peut observer deux profils : pour un lr de 0.005 et un momentum de 0.25 et 0.5 la croissance de la courbe est plus lent progressif en train et en validation avec un faible signe de surapprentissage. L'accuracy ne dépasse pas les 80%. La loss est miroir à ce profil. Lorsque le lr est de 0.005 et le momentum est de 0.9, l'accuracy atteint les 100% mais au détriment d'un fort surapprentissage. Les tendances y sont identiques avec un apprentissage similaire en vitesse mais pas en accuracy. En terme de loss le profil y est

miroir à l'accuracy : décroissance forte avec cette fois ci un faible regain en fin d'apprentissage.

- Pour 0.001 de learning rate et [0.25, 0.5, 0.9] de momentum:

Pour finir on peut observer que les profils de courbes sont relativement les mêmes en écartant leur accuracy : croissance faible mais progressive avec un plateau moins prononcé en validation et en train. En loss cette tendance est identique. Néanmoins on peut y observer des différences et la première est que pour un lr de 0.001 et un momentum de 0.9 on y voit du surapprentissage et une courbe plus arrondi comparé aux deux autres réseaux. Ces réseaux ne sont pas en surapprentissage mais on une accuracy plus faible aux alentours de 50% en train comparé au 100% en train et 60% en validation du paramétrage lr de 0.001 et un momentum de 0.9