Projet Convexe Optimisation

HOSSAIN Bushra

KADRI Aymen

LAGHZAOUI Ismail

TOUDJINE Juba

BDML 1

efrei PARIS
École d'ingénieurs du numérique

TABLE DES MATIERES

INTRODUCTION

La reconnaissance des émotions à partir d'expressions faciales a toujours été un problème fondamental en computer vision. Essayer de prédire les émotions d'autrui à travers les expressions du visage relève d'un vrai défi. Ce projet s'inscrit dans le cadre du **"Facial Expression Recognition Challenge"** [1], un concours organisé dans le domaine de l'apprentissage de représentations.

Dans ce contexte, notre objectif est de construire un **modèle de deep learning**, basé sur un **réseau de neurones convolutionnel (CNN)**, capable de classifier correctement les émotions à partir des images. Cependant, la performance du modèle dépend fortement de **l'optimiseur** utilisé et de ses **hyperparamètres**.

C'est pourquoi, au-delà du simple entraînement d'un modèle, ce travail s'intéresse particulièrement à **l'analyse mathématique et empirique de l'optimisation**, en étudiant le comportement de l'algorithme **Adam**, largement utilisé pour les problèmes non convexes. Nous évaluerons notamment **la convexité de la fonction de perte**, les **effets des différents paramètres d'Adam** (learning rate, β1, β2, etc.), ainsi que leur **impact sur la convergence** et la précision du modèle.

[1] Challenges in Representation Learning: Facial Expression Recognition Challenge | Kaggle

1. PRESENTATION DU DATASET

Le jeu de données utilisé dans ce projet provient du **"Facial Expression Recognition Challenge",** un concours visant à explorer l'apprentissage des représentations pour la reconnaissance d'émotions à partir d'images faciales.

Ce dataset est composé de **28 709** images pour l'entraînement, et **7 178** images pour le test. Chaque image est une image en niveaux de gris (grayscale) de **48x48 pixels**, représentant un visage humain centré.

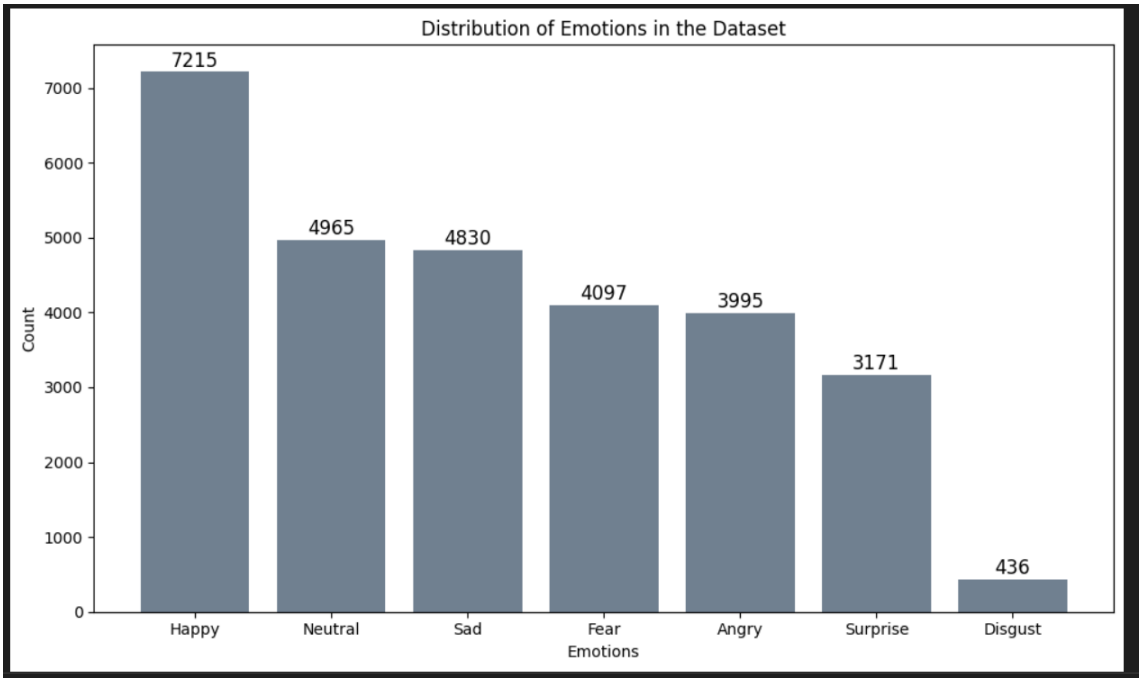
Les émotions sont codées sous forme d'entiers de 0 à 6, avec la correspondance suivante :

```
# 0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral
emotionLabels = {
    0: 'Angry',
    1: 'Disgust',
    2: 'Fear',
    3: 'Happy',
    4: 'Sad',
    5: 'Surprise',
    6: 'Neutral'
}

emotion_counts = train_data['emotion'].value_counts()
```

Chaque ligne du fichier train.csv contient :

- Une **colonne emotion** : le label de l'émotion (entre 0 et 6)
- Une **colonne pixels** : une chaîne de 2304 valeurs séparées par des espaces, représentant les intensités de chaque pixel.

Cependant, on note un déséquilibre : par exemple, l'émotion **Disgust** est sous représenté, et cela peut affecter la performance de notre modèle lors de son entraînement.



Il faut noter également que les expressions du visage sont très subjectives à chacun, par exemple, un visage **'sad'** peut être confondu avec un visage **'fear'**.

On peut donc avoir des ambiguïtés des expressions, certaines expressions peuvent se ressembler visuellement ce qui rend la tâche de classification d'autant plus complexe.

2. DATA PRE-PROCESSING

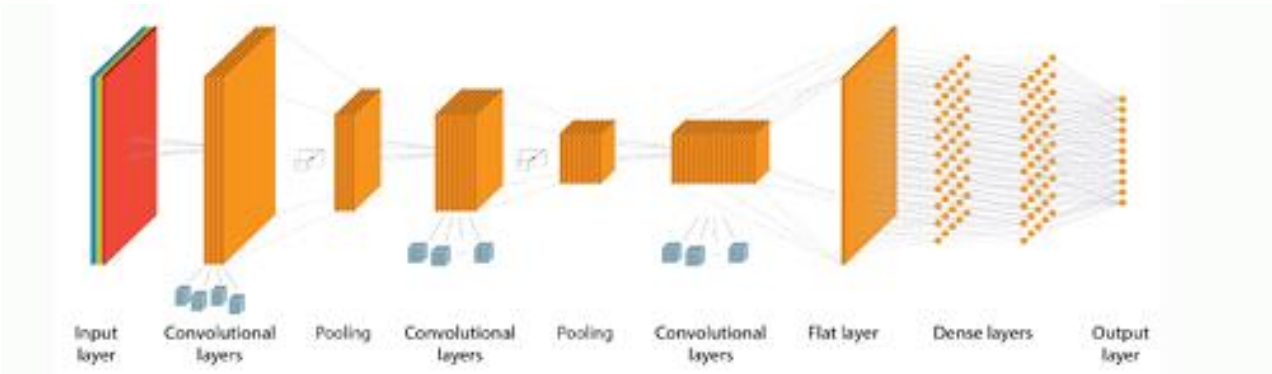
Lors de la préparation des données pour être exploitables par notre réseau de neurones CNN nous avons effectué ceci :

```
X = train_data['pixels'].apply(lambda x: np.array(x.split()).astype('float32'))
y = train_data['emotion']

# 48x48 pixels
X = np.stack(X.values)
X = X.reshape(-1, 48, 48, 1)

# Normalize
X = X / 255.0

# Convert labels to one-hot encoding
y = to_categorical(y, num_classes=7)

# Split the data into training and validation sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Convertir d'abord chaque pixel en tableau (48,48,1)
- Normaliser la valeur des pixels pour qu'on obtient des valeurs entre 0 et 1
- Faire du one-hot encoding à 7 dimensions pour que nos labels d'émotions soient compatibles avec notre fonction de perte categorical_crossentropy
- On sépare les données en train (80%) et test (20%)

3. ARCHITECTURE DU MODELE CNN

Pour résoudre notre problème de classification d'expressions faciales, nous avons choisi de faire un CNN (réseau de neurones convolutionnel) qui est efficace pour le traitement d'images en computer vision.



Notre modèle est composé de plusieurs couches de convolution et de couches denses pour la classification finale :

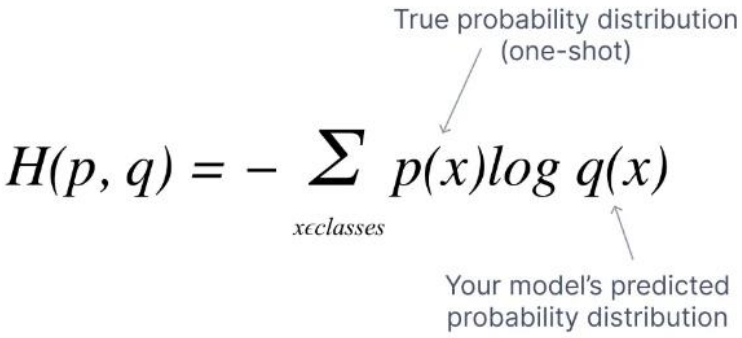
- **4 blocs de convolution**
  - ➔ Chacun contenant une couche Conv2D
  - ➔ Une activation Relu pour la non-linéarité
  - ➔ Un MaxPooling2D pour réduire la dimension
  - ➔ Et un dropout pour limiter l'overfitting
- **2 couches denses**
  - ➔ Une couche avec 256 neurones
  - ➔ Une couche avec 512 neurones
- **Une couche de sortie**
  - ➔ Une couche Dense avec 7 neurones et utilisation du softmax

En choix d'optimisation, notre modèle sera compilé avec **Adam**, et la fonction de perte **categorical_crossentropy**, avec comme métrique d'évaluation **l'accuracy**.

4. ETUDE DE LA FONCTION DE PERTE (CONVEXITE ET DIFFERENTIABILITE)

Nous utilisons la fonction de perte ***categorical_crossentropy***, qui est adaptée pour de la classification multi-classes en one-hot encoding.

La fonction de perte est une combinaison de l'entropie croisée et de la sortie softmax.



Son but : mesurer la dissimilarité entre la distribution prédite par le modèle (donc après le softmax), avec la distribution réelle des labels.

4.1 CONVEXITE DE LA FONCTION

La fonction est-elle convexe ?

Non, seule la fonction du cross entropy est convexe si elle est appliquée à une sortie softmax linéaire. Par exemple, on sait que dans un modèle linéaire, la loss est convexe et donc on a un seul minimum global.

Cependant, dans notre cas, un réseau de neurones (CNN) est non linéaire, et cette propriété est donc contredite.

Les activations ReLU, les convolutions et les max-pooling rendent notre fonction **non-convexe,** et donc ne remplit pas la condition de cette formule

- A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex iff:

$$\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n, \forall \lambda \in [0,1] : f\left(\lambda \mathbf{x} + (1-\lambda)\,\mathbf{y}\right) \leq \lambda f\left(\mathbf{x}\right) + (1-\lambda) f\left(\mathbf{y}\right)$$

(en remplaçant la fonction f= H(p,q), en posant f(q) = H(p,q) avec p notre label one-hot)

Problème : On aura donc plusieurs minimaux locaux, des plateaux de perte, ainsi que des oscillations. Notre fonction de perte **n'est ni convexe, ni concave**.

4.2 DIFFERENTIABILITE DE LA FONCTION

La fonction est-elle différentiable ?

Oui, presque partout si notre hypothèse de départ est que les entrées sont des probabilités strictement positive (ce qui est le cas par le softmax).

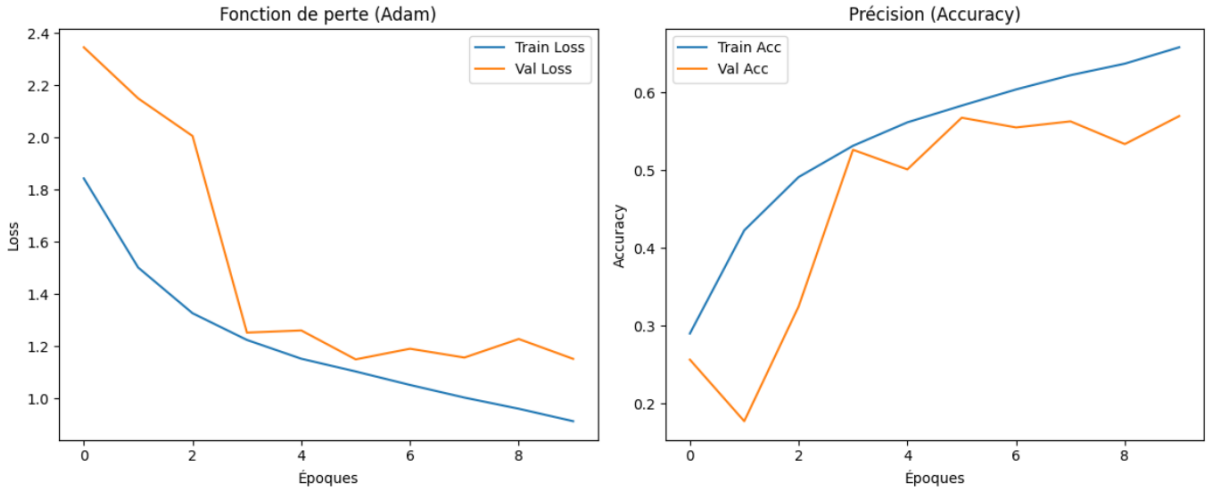
Cependant la fonction ReLU définie comme telle dans les couches cachées :

$$ReLU(x) = max(x, 0)$$

N'est pas différentiable en 0 car les points où x=0 sont rares.

4.3 ANALYSE EMPIRIQUE SUR 10 EPOQUES

L'évolution de la perte et de l'accuracy est présentée dans la figure ci-dessus. On observe :



- Une **forte baisse du Train Loss**, montrant une bonne capacité d'apprentissage.
- Le **val_loss** suit la courbe du train_loss, mais avec plus de bruit (instabilité), notamment entre les époques 3 à 9 indiquant un début **d'overfitting.**
- L'**accuracy** de validation dépasse les **56%** dès la 6e époque, ce qui est significatif pour un problème à 7 classes.

Ainsi, malgré la non-convexité de la fonction de perte, l'optimiseur Adam permet une descente stable et efficace.

5. ANALYSE DE L'OPTIMISEUR ADAM

L'algorithme ***ADAM (Adaptive moment estimation)*** est une méthode d'optimisation stochastique. Il combine le **momentum (**historique des gradients) et l'adaptation du **learning rate** pour chaque paramètre.

5.1 ALGORITHME ADAM – FORMULATION MATHEMATIQUE

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

[2]

L'algorithme Adam met à jour les poids du modèle en combinant deux moyennes (le gradient moyen et le carré du gradient), corrigées du biais, pour adapter dynamiquement le pas de descente à chaque paramètre, ce qui est adapté pour les fonctions non-convexe comme la nôtre.

5.2 INTERET D'ADAM FACE A LA NON-CONVEXITE

Dans le cas de notre CNN, la fonction de perte est appliquée à une architecture non linéaire (ReLU, convolutions), ce qui rend le problème **non convexe**.

[2] Adam Optimization Algorithm | Towards Data Science

Adam n'assure **pas** la convergence vers un minimum global, mais il **améliore la stabilité** de la descente grâce à :

- Une **accélération adaptative** (momentum)
- Une **régularisation naturelle** sur les updates
- Une **robustesse aux plateaux** ou zones de gradient très faible

Cela permet d'éviter certains pièges courants dans l'optimisation non convexe : stagnation, oscillations ou divergence.

5.3 HYPERPARAMETRES ANALYSES

Adam contient plusieurs hyparamètres qui influencent la stabilité et la vitesse de convergence. On va donc choisir ceux-là :

Hyperparamètre	**Utilité**
α (*learning rate*)	Contrôle la **vitesse d'apprentissage**. C'est le **pas de descente fixe** utilisé à chaque itération pour mettre à jour les poids.
β1	Coefficient de pondération du **1er moment** (moyenne des gradients). Il agit comme un **momentum**, c'est-à-dire qu'il introduit une **inertie** dans la descente pour lisser les directions de mise à jour.
β2	Coefficient pour le **2nd moment** (moyenne des gradients au carré). Il permet de **stabiliser l'optimisation** en adaptant dynamiquement le learning rate à la variance du gradient.
Weight decay	Terme de **régularisation L2** ajouté aux poids du modèle, qui vise à **réduire l'overfitting** en pénalisant les grandes valeurs de paramètres.

5.3.1 ETUDE DU LEARNING RATE

Dans notre étude on compare différentes valeurs : 0.0001, 0.001 et 0.01.

Observations :

- **α =0.01** : on observe des oscillations et une instabilité de la Loss tout au long des époques

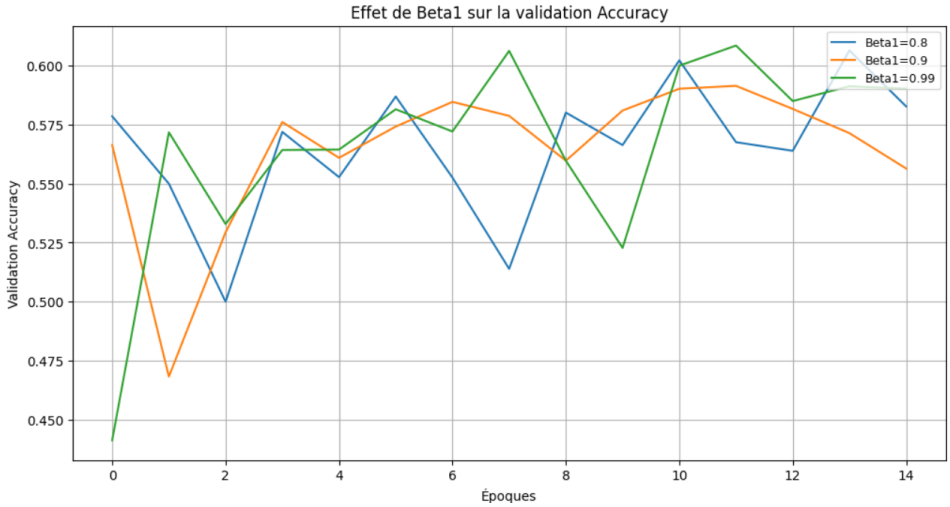
- **α =0.001** : On a une convergence rapide, moins d'oscillation mais toujours aussi instable visuellement
- **α =0.0001** : On observe une meilleure stabilité de la Loss, mais lente.

Un pas trop grand peut conduire à des divergences et à une convergence trop rapide, alors qu'un pas trop petit puis conduire à un ralentissement de l'optimisation et à du overfitting.

Nous choisissons donc un pas assurant une certaine stabilité, donc **α =0.0001**

5.3.2 ETUDE DU BETA 1

Pour contrôler l'influence du momentum, afin d'accélérer la convergence et de lisser la pente de la fonction loss, nous avons testé 3 valeurs : **0.8 , 0.9** et **0.99**.



Ainsi nous observons que :

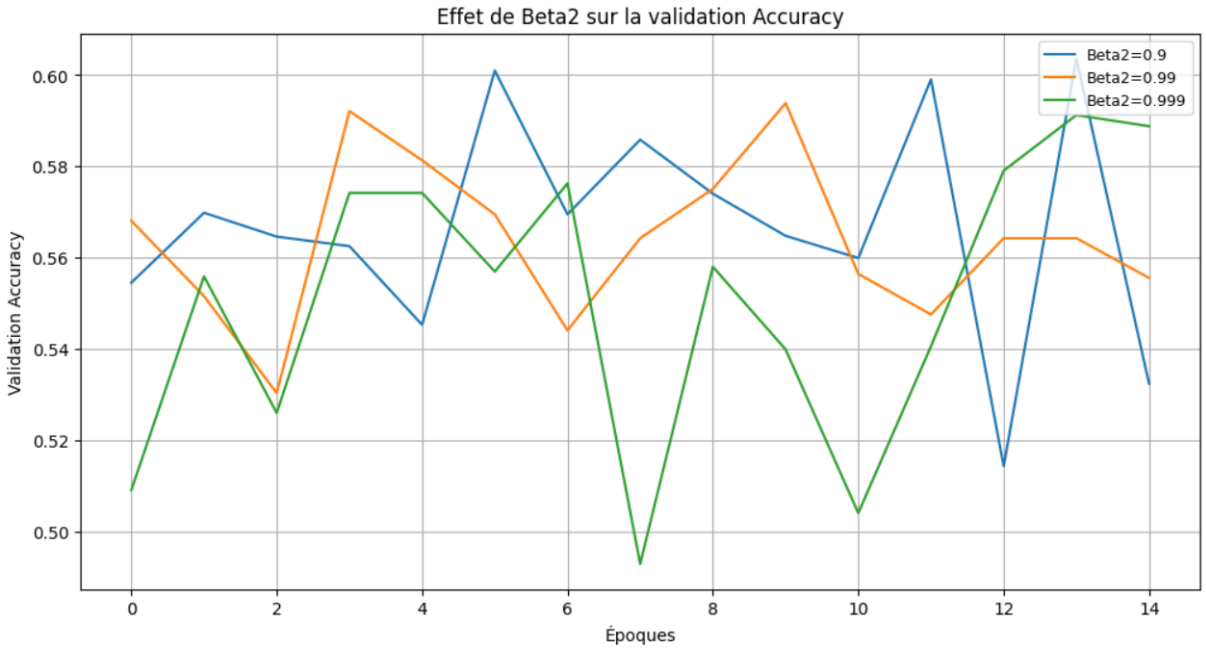
- $\beta_1 = 0.8$: le modèle réagit rapidement mais est instable car la validation loss fluctue tout au long des epochs.
- $\beta_1 = 0.9$: la valeur par défaut de l'algorithme Adam assurant une certaine stabilité
- $\beta_1 = 0.99$: une valeur élevée conduit à un apprentissage stable mais ralentit la convergence et l'adaptation

Choix pour le modèle après calcul de l'accuracy atteignant 60% d'accuracy :

$\beta_1 = 0.99$

5.3.3 ETUDE DU BETA 2

Pour le paramètre B2 permettant de lisser les fluctuations des gradients quadratiques afin de stabiliser la mise à jour des poids, nous testerons 3 valeurs : **0.9, 0.99,** et **0.999**.



Nous observons ainsi que :

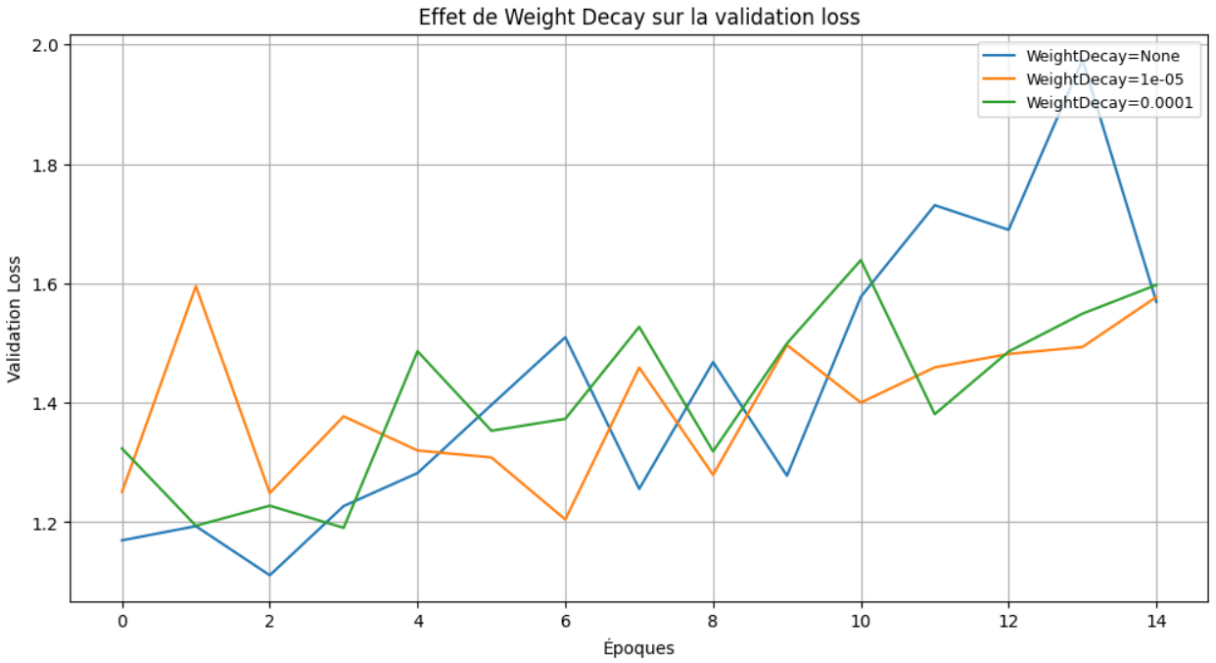
- $\beta_2 = 0.9$: le modèle apprend vite au début mais les résultats flucutent d'une époque à l'autre, le training est donc instable.
- $\beta_2 = 0.99$: bon équilibre, apprentissage rapide et résultats stables. La perte ne varie pas beaucoup.
- $\beta_2 = 0.999$: l'algorithme réagit trop lentement, apprentissage lent et le modèle ne s'adapte pas correctement, puisqu'il lisse trop les gradients.

Choix pour le modèle maximisant notre validation accuracy → **β2 = 0.99**

5.3.4 ETUDE DU WEIGHT DECAY

Cette technique de régularisation L2 agit comme une pénalité pour les grands poids du modèle afin d'éviter l'overfitting.

Nous testons 3 valeurs : Aucune régularisation (None), une faible (1^{e}-05), modérée (0.0001).



On observe ainsi :

- **None** : l'accuracy atteint rapidement 60% mais instabilité de la validation loss et augmente au fur et à mesure des époques
- **1e-05** : accuracy atteignant 60% et évolution relativement stable
- **0.0001** : validation loss plus élevée mais régularisation trop forte donc risque d'overfitting.

Choix pour le modèle maximisant notre validation accuracy → **weight decay= 1e-05**

Ainsi les meilleurs hyperparamètres choisis sont :

```
best_lr = 0.0001
best_beta1 = 0.99
best_beta2 = 0.999
best_decay = 1e-5
```

```
model.compile(optimizer=Adam(learning_rate=best_lr,beta_1=best_beta1, beta_2=best_beta2, epsilon=1e-7,
              weight_decay=best_decay, amsgrad=False),
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

5.4 VITESSE DE CONVERGENCE

L'analyse de la vitesse de convergence montre l'efficacité avec laquelle chaque configuration atteint un seuil de 60 % de précision sur les données de validation.

```
LR=0.0001 atteint 60% val_acc à l'époque 1
LR=0.001 atteint 60% val_acc à l'époque 13
LR=0.01 n'atteint pas 60% val_acc
Beta1=0.8 atteint 60% val_acc à l'époque 11
Beta1=0.9 n'atteint pas 60% val_acc
Beta1=0.99 atteint 60% val_acc à l'époque 8
Beta2=0.9 atteint 60% val_acc à l'époque 6
Beta2=0.99 n'atteint pas 60% val_acc
Beta2=0.999 n'atteint pas 60% val_acc
WeightDecay=None atteint 60% val_acc à l'époque 3
WeightDecay=1e-05 n'atteint pas 60% val_acc
WeightDecay=0.0001 n'atteint pas 60% val_acc
```

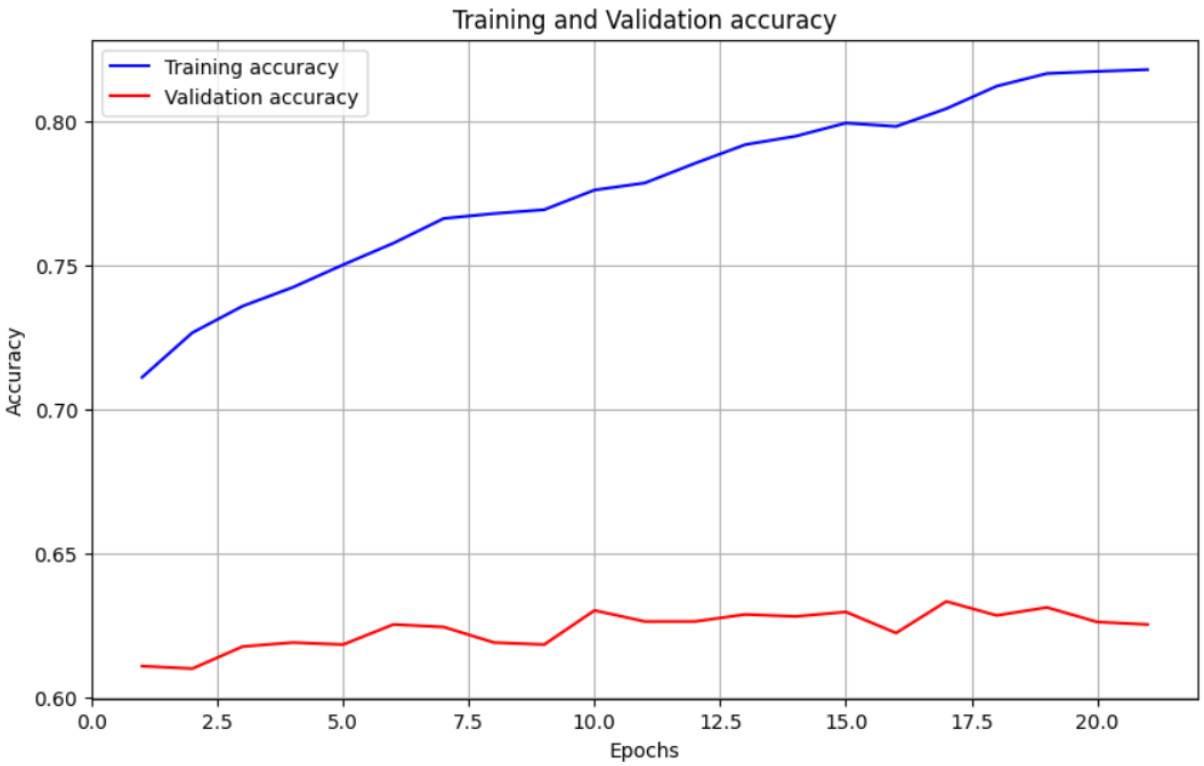
On remarque donc que pour

- **LR = 0.0001**, nous avons une convergence très rapide dès l'époque 1.
- **Momentum B1 = 0.99** : Nous avons une convergence rapide à l'époque 8 contre l'époque 11 où B1 = 0.8
- **B2 = 0.9** : atteint dès la 6e époque pour 60% de val acc

- Enfin, un **weightdecay=None** permet une montée rapide en précision mais peut mener à du overfitting, d'où notre choix pour le 1e-50.

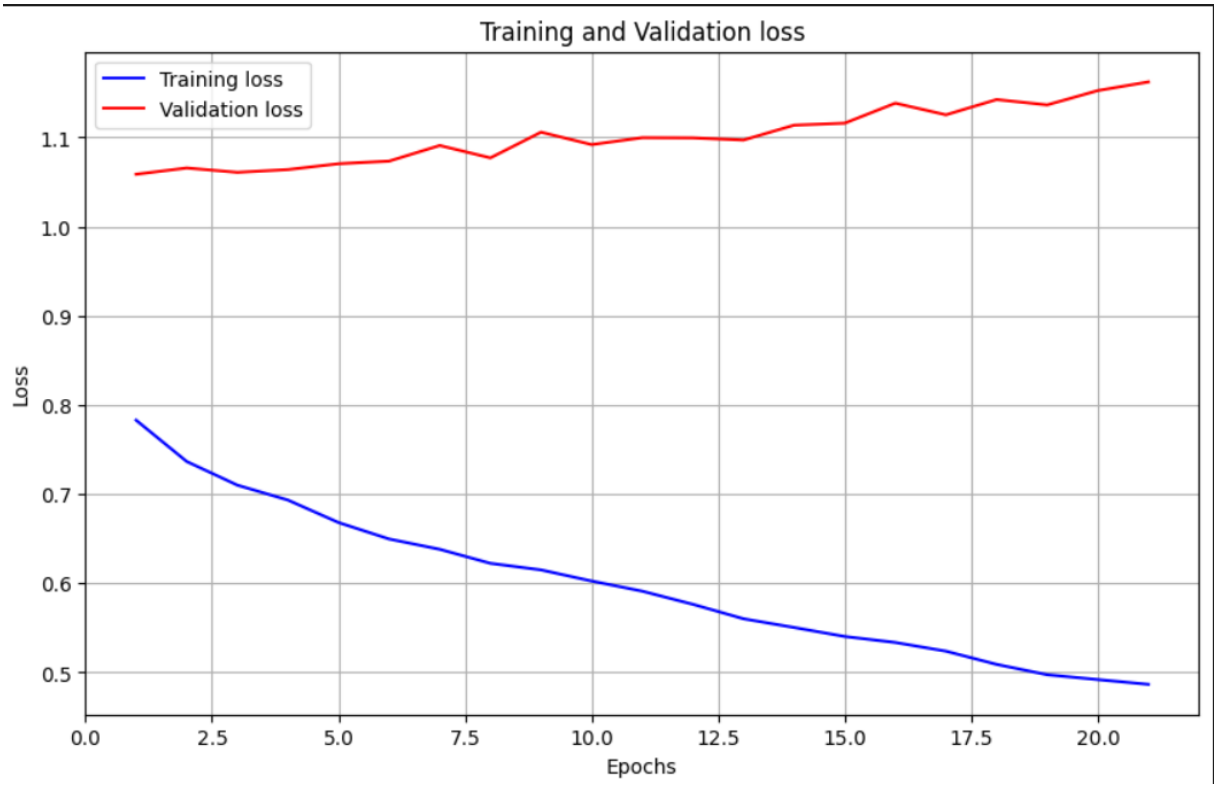
6. ENTRAINEMENT DU MODELE ET RESULTATS

Après avoir testé et comparé plusieurs configurations d'hyperparamètres de l'optimiseur Adam, nous avons obtenu une **accuracy de 61.09%** sur le test data avec **une loss finale de validation** autour de **1.06.**



On remarque que l'accuracy du training continue d'augmenter de façon régulière jusqu'à plus de 80%.

L’accuracy de Validation elle, oscille autour de 61% ce qui montre qu’il y a de l’overfitting à partir d’un certain nombre d’époque. (notre accuracy baisse drastiquement).



Malgré un certain écart entre les performances sur les ensembles d'entraînement et de validation, le modèle atteint un bon compromis général grâce à un **réglage minutieux des hyperparamètres**, en particulier ceux d’Adam.

CONCLUSION

Ce projet nous a permis d'aborder un problème d'optimisation sur une fonction non-convexe dans le cadre de reconnaissance faciale pour un problème de classification d'émotions.

L'analyse a montré que les propriétés mathématiques ainsi que les tests sur plusieurs valeurs d'hyparamètres étaient importantes afin de choisir les meilleures valeurs pour notre modèle finale.

ANNEXE & REFERENCES

Architecture du CNN : Face expression recognition with Deep Learning

Dataset : Challenges in Representation Learning: Facial Expression Recognition Challenge | Kaggle

Algorithme Adam: Adam — PyTorch 2.7 documentation

Adam Optimization Algorithm | Towards Data Science

Fonction de Loss: What Is Cross-Entropy Loss Function? | GeeksforGeeks