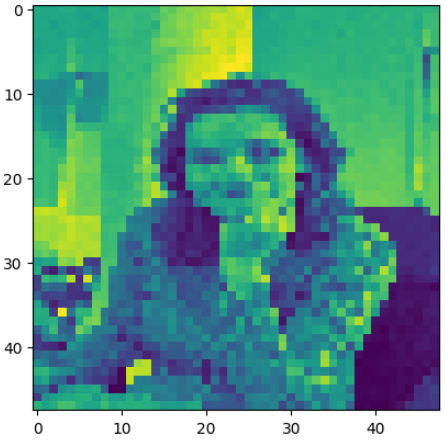
18 AVRIL 2024

PiLoJoRé

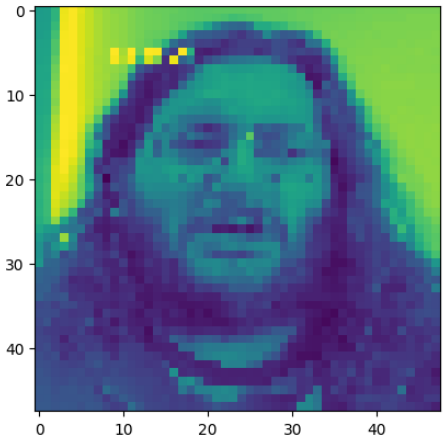
Détection des sentiments

Détection des sentiments dans les vidéos par l'intelligence artificielle (IA)

La détection des sentiments dans les vidéos par l'intelligence artificielle (IA) repose sur des techniques d'analyse d'images et de reconnaissance faciale. Il est important de noter que la détection des émotions dans les vidéos par l'IA peut présenter des défis, notamment en raison de la variabilité des expressions faciales et de la nécessité de tenir compte du contexte pour une interprétation précise.



Ce projet nous a donc permis d'explorer les défis et les techniques associés à la fois à la détection des émotions dans les vidéos par l'IA et à l'implémentation d'un modèle de détection d'émotions en Python. Pour améliorer la précision de notre modèle, nous l'avons entraîné sur de vastes ensembles de données annotées. Une fois notre modèle optimisé, nous pouvons vous présenter le prototype.



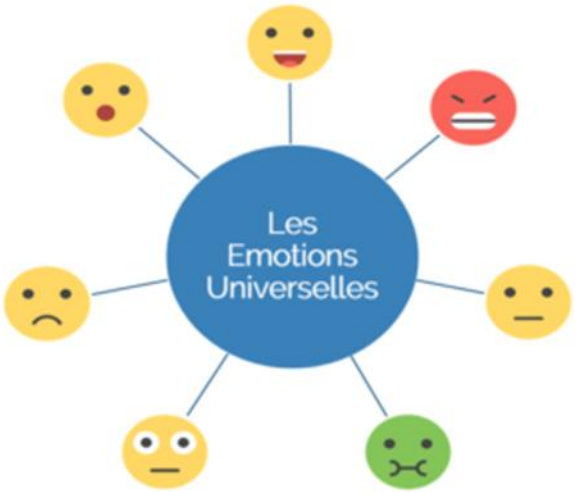
Nous avons ainsi pu mettre en pratique nos connaissances en apprentissage automatique et en traitement d'images pour développer un modèle capable de classifier les émotions à partir d'images en niveaux de gris.

Introduction :

- **Détection faciale :** L'IA commence par détecter et suivre les visages dans la vidéo. Cela peut être fait en utilisant des algorithmes de détection faciale qui identifient les caractéristiques faciales telles que les yeux, le nez et la bouche.

- **Extraction des caractéristiques :** Une fois les visages détectés, l'IA extrait des caractéristiques spécifiques de chaque visage. Ces caractéristiques sont essentielles pour comprendre les émotions exprimées par les sujets de la vidéo.

- **Classification des émotions :** En utilisant des techniques d'apprentissage automatique telles que les réseaux de neurones, l'IA analyse les caractéristiques extraites pour classer les émotions exprimées par les sujets. Les émotions couramment détectées comprennent la joie, la tristesse, la colère, la peur, le dégoût, et la surprise.



- **Entraînement du modèle :** Pour que l'IA puisse classifier correctement les émotions, elle doit être entraînée sur de grandes quantités de données, comprenant des photos annotées où les émotions sont clairement étiquetées. Plus l'ensemble de données d'entraînement est diversifié et représentatif de différentes situations et cultures, meilleure sera la capacité de l'IA à détecter les émotions avec précision.

- **Optimisation et amélioration :** Une fois le modèle entraîné, il est souvent optimisé et amélioré en utilisant des techniques telles que le transfert d'apprentissage et le fine-tuning pour le rendre plus précis et généralisable.

- **Intégration dans les systèmes :** Une fois que le modèle est prêt, il peut être intégré dans des systèmes plus larges, tels que des plateformes de vidéos en ligne ou des logiciels de surveillance, pour fournir des informations sur les émotions des personnes dans les vidéos.

En tant qu'ingénieurs en IA, nous avons exploré la détection des sentiments dans les vidéos en utilisant des techniques avancées d'analyse d'images et de reconnaissance faciale.

Mise en œuvre :

Dans notre approche, nous avons commencé par utiliser des algorithmes de détection faciale pour détecter et suivre les visages dans la vidéo. Ces algorithmes identifient les caractéristiques faciales telles que les yeux, le nez et la bouche en Python, en utilisant des bibliothèques comme OpenCV. Une fois les visages détectés, nous avons pu extraire des caractéristiques spécifiques de chaque visage, telles que les expressions faciales et les mouvements des lèvres. Pour ce faire, nous avons utilisé des techniques d'extraction de caractéristiques en Python, en exploitant des bibliothèques comme TensorFlow. En utilisant des techniques d'apprentissage automatique telles que les réseaux de neurones, nous avons analysé les caractéristiques extraites pour classer les émotions exprimées. Nous avons entraîné nos modèles de classification d'émotions en Python, en utilisant des ensembles de données annotées et des bibliothèques telles que Keras.

Notre code définit des générateurs de données pour l'entraînement et la validation d'un modèle d'apprentissage automatique utilisant des images. Ces générateurs sont basés sur la classe ImageDataGenerator de la bibliothèque Keras, qui permet de générer des lots d'images avec des transformations aléatoires à la volée.

- Voici ce que chaque paramètre de la classe ImageDataGenerator fait :

- **rotation_range :** Rotation aléatoire des images dans la plage spécifiée en degrés. Dans ce cas, il est commenté, donc aucune rotation n'est appliquée.
- **width_shift_range :** Décalage horizontal aléatoire des images dans une plage donnée, exprimée en fraction de la largeur totale de l'image.
- **height_shift_range :** Décalage vertical aléatoire des images dans une plage donnée, exprimée en fraction de la hauteur totale de l'image.
- **horizontal_flip :** Flip horizontal aléatoire des images.
- **rescale :** Rééchelle les valeurs de pixel des images entre 0 et 1 en divisant chaque valeur de pixel par 255. Cela standardise les valeurs de pixel.
- **zoom_range :** Applique un zoom aléatoire à l'objet dans l'image.
- **validation_split :** Réserve une fraction des données pour la validation. Ici, 20% des données sont réservées à cet effet.

Le générateur train_datagen est utilisé pour augmenter les données d'entraînement en appliquant des transformations aléatoires telles que le décalage, le retournement horizontal, etc. Le générateur validation_datagen est utilisé pour la validation et ne fait que rééchelonner les valeurs de pixel des images entre 0 et 1. Ces générateurs sont ensuite utilisés pour charger les données d'entraînement et de validation à partir de répertoires contenant des images, ce qui facilite la manipulation des images en lots pendant l'entraînement du modèle.

- Voici ce que chaque paramètre des fonctions flow_from_directory fait :

- **target_size :** La taille des images après redimensionnement.
- **batch_size :** Le nombre d'images à générer par lot.
- **color_mode :** Le mode de couleur des images. Ici, "grayscale" indique que les images seront chargées en niveaux de gris.
- **class_mode :** Le mode de classification des classes. Ici, "categorical" indique que les classes sont catégoriques.
- **subset :** Indique si le générateur doit être utilisé pour l'entraînement ("training") ou la validation ("validation").

Le générateur train_generator est utilisé pour charger les données d'entraînement à partir du répertoire train_dir avec des transformations d'augmentation de données appliquées, telles que le décalage horizontal, le retournement horizontal, etc. Le générateur validation_generator est utilisé pour charger les données de validation à partir du répertoire test_dir, également avec des transformations d'augmentation de données appliquées. En utilisant ces générateurs, les images sont chargées en lots avec les transformations d'augmentation de données appliquées à la volée, ce qui permet d'augmenter la variabilité des données et d'améliorer la capacité du modèle à généraliser.

- Convolutional Layers (Conv2D) :

La première couche convolutive a 64 filtres avec une taille de noyau de (3,3), une activation ReLU et une couche d'entrée avec une taille d'image de (img_size, img_size, 1) pour les images en niveaux de gris. Les couches suivantes ont des nombres de filtres et des tailles de noyau différents pour extraire des caractéristiques plus complexes de l'image.

- Pooling Layers (MaxPool2D) :

Après chaque couche convolutive, une couche de max pooling avec un pool size de (2,2) et des strides de (2,2) est appliquée pour réduire la dimensionnalité de l'image.

- Batch Normalization :

Des couches de normalisation par lot sont ajoutées après chaque couche de pooling pour accélérer la convergence et stabiliser l'entraînement du réseau.

- Dropout :

Des couches de dropout sont utilisées pour réduire le surajustement en désactivant aléatoirement un pourcentage de neurones à chaque étape d'entraînement.

- Flatten :

Après les couches convolutives et de pooling, une couche Flatten est utilisée pour convertir les tenseurs en vecteurs, permettant ainsi de passer à une couche entièrement connectée.

- Fully Connected Layers (Dense) :

Des couches entièrement connectées suivent la couche Flatten pour effectuer la classification finale. Des activations ReLU sont utilisées dans les couches cachées pour introduire de la non-linéarité. La dernière couche utilise une activation softmax pour la classification multi-classe des émotions, avec 7 unités de sortie correspondant à 7 classes d'émotions différentes.

- Initialisation des poids :

Les poids des couches Dense sont initialisés à l'aide de l'initialiseur "he_normal", qui initialise les poids de manière adaptative pour les fonctions d'activation ReLU.

En résumé, ce modèle, est un réseau de neurones convolutionnel (CNN), et est conçu pour extraire des caractéristiques pertinentes des images, les combiner dans des couches entièrement connectées et produire des prédictions sur les émotions présentes dans les images en niveaux de gris. Nous avons défini les paramètres d'entraînement en Python, tels que le nombre d'époques et la taille des lots. Ces paramètres ont été ajustés en fonction des caractéristiques spécifiques du problème et des performances du modèle lors de l'entraînement.

- Compilation du modèle :

Le modèle est compilé avec l'optimiseur Adam et un faible taux d'apprentissage (lr=0.0001), une fonction de perte de 'categorical_crossentropy' pour la classification multi-classe et la métrique de 'précision' pour évaluer les performances du modèle lors de l'entraînement. Ce modèle est conçu pour être entraîné sur des images en niveaux de gris de taille (48, 48, 1) pour la classification d'émotions.

- **epochs** = ici variable : Cela signifie que le modèle sera entraîné sur l'ensemble des données d'entraînement pendant X époques. Une époque correspond à une passe complète à travers l'ensemble des données d'entraînement.

- **batch_size** = 64 : Pendant chaque époque, les données d'entraînement sont divisées en lots de taille 64. Le modèle sera mis à jour après chaque lot de données. Utiliser des lots permet d'accélérer le processus d'entraînement et de mieux généraliser les données.

En résumé, le modèle a été entraîné sur l'ensemble des données d'entraînement pendant 3, puis 15 puis 40 époques, chaque époque utilisant des lots de taille 64 pour mettre à jour les poids du modèle.

Figure 1 : Le graphique de gauche représente l'évolution de la précision d'entraînement (en bleu) et de validation (en orange) au fil des 15 époques. Le second graphique montre comment les valeurs de perte d'entraînement (en bleu) et de validation (en orange) varient au cours de l'entraînement du modèle.

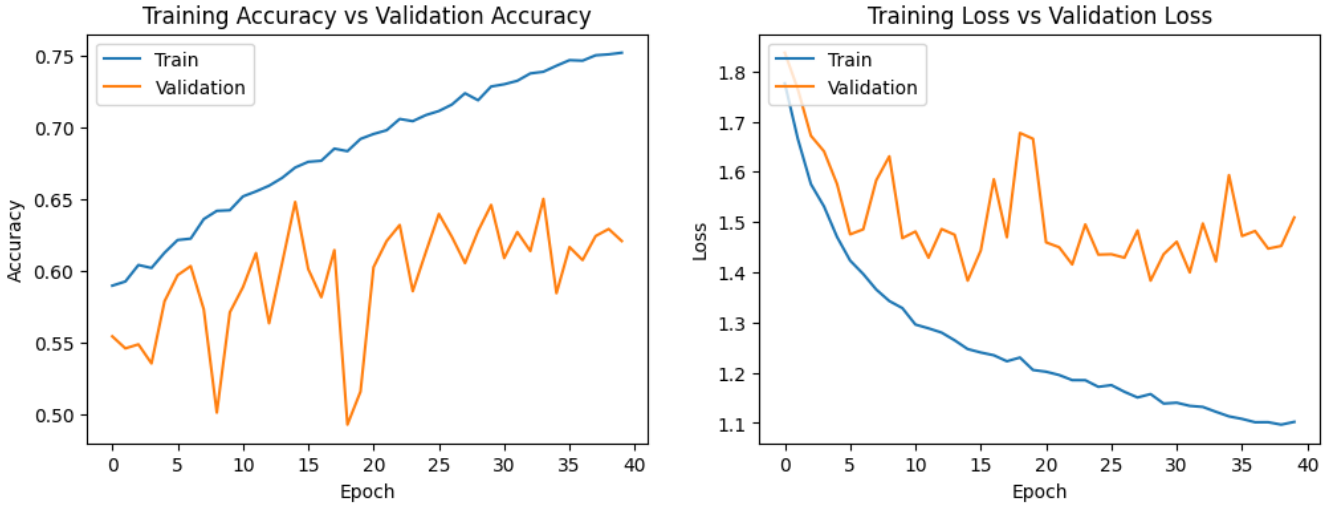


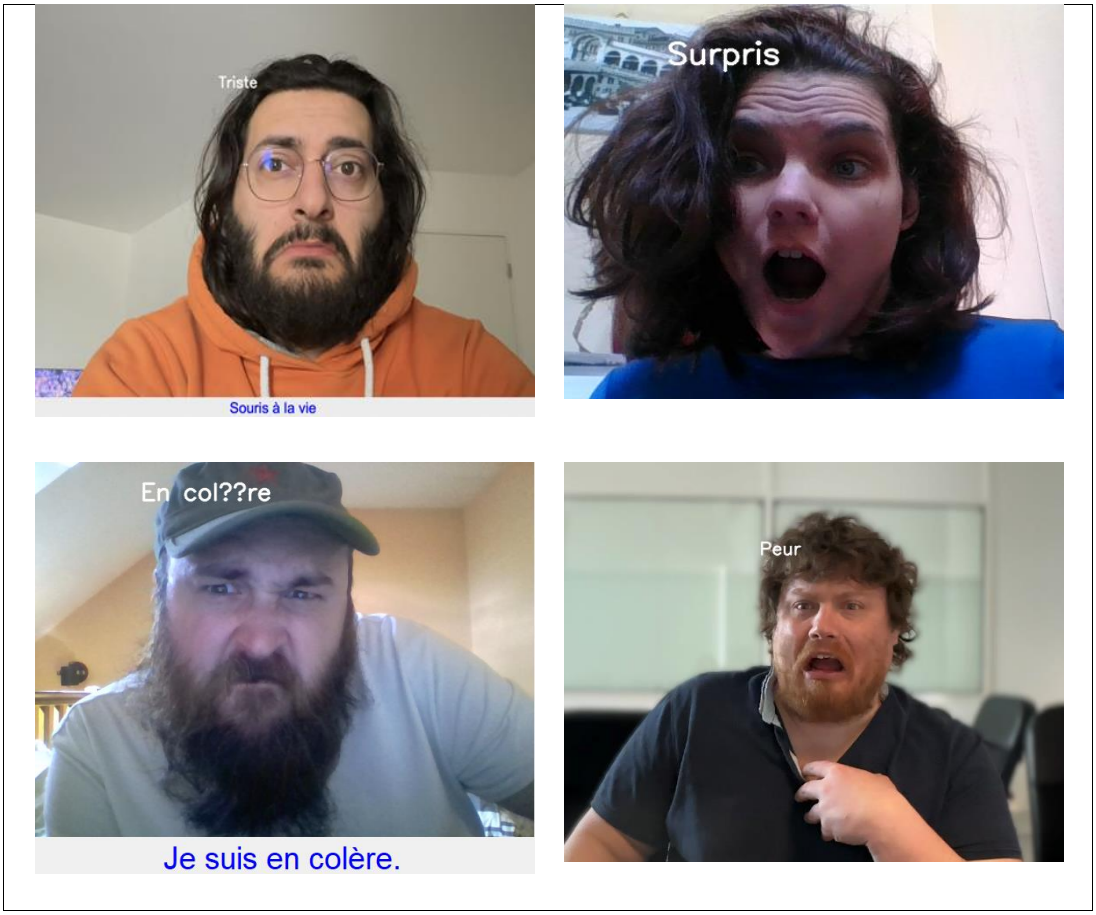
Figure 2 : Le graphique de gauche représente l'évolution de la précision d'entraînement (en bleu) et de validation (en orange) au fil des 40 époques. Le second graphique montre comment les valeurs de perte d'entraînement (en bleu) et de validation (en orange) varient au cours de l'entraînement du modèle.

Ce modèle a un total de 11,328,007 paramètres, dont 11,317,895 sont entraînables et 10,112 ne sont pas entraînables. Les paramètres non entraînables correspondent aux paramètres des couches de BatchNormalization.

Nous avons évalué les performances du modèle sur les données de validation à la fin de chaque époque pour suivre sa progression.

```
Epoch 1/20
359/359 ━━━━━━━━━━━━━━━━━━━━ 476s 1s/step - accuracy: 0.7644 - loss: 1.0721 - val_accuracy: 0.6327 -
Epoch 2/20
359/359 ━━━━━━━━━━━━━━━━━━━━ 464s 1s/step - accuracy: 0.7683 - loss: 1.0554 - val_accuracy: 0.6271 -
Epoch 3/20
359/359 ━━━━━━━━━━━━━━━━━━━━ 470s 1s/step - accuracy: 0.7682 - loss: 1.0648 - val_accuracy: 0.5971 -
Epoch 4/20
359/359 ━━━━━━━━━━━━━━━━━━━━ 467s 1s/step - accuracy: 0.7740 - loss: 1.0527 - val_accuracy: 0.6369 -
Epoch 5/20
359/359 ━━━━━━━━━━━━━━━━━━━━ 467s 1s/step - accuracy: 0.7746 - loss: 1.0606 - val_accuracy: 0.6173 -
Epoch 6/20
359/359 ━━━━━━━━━━━━━━━━━━━━ 469s 1s/step - accuracy: 0.7737 - loss: 1.0406 - val_accuracy: 0.5349 -
Epoch 7/20
359/359 ━━━━━━━━━━━━━━━━━━━━ 468s 1s/step - accuracy: 0.7749 - loss: 1.0422 - val_accuracy: 0.5419 -
Epoch 8/20
359/359 ━━━━━━━━━━━━━━━━━━━━ 491s 1s/step - accuracy: 0.7778 - loss: 1.0444 - val_accuracy: 0.6271 -
Epoch 9/20
359/359 ━━━━━━━━━━━━━━━━━━━━ 550s 2s/step - accuracy: 0.7828 - loss: 1.0345 - val_accuracy: 0.6334 -
```

Conclusion :



En adaptant un modèle existant et en l'entraînant avec le jeu de données "fer-2013", nous avons pu acquérir une expérience pratique précieuse dans le domaine de l'apprentissage automatique. Bien que

nous n'ayons pas créé le modèle à partir de zéro, le processus d'entraînement et d'optimisation du modèle nous a permis de mieux comprendre les défis et les nuances de la détection des émotions dans les médias visuels.

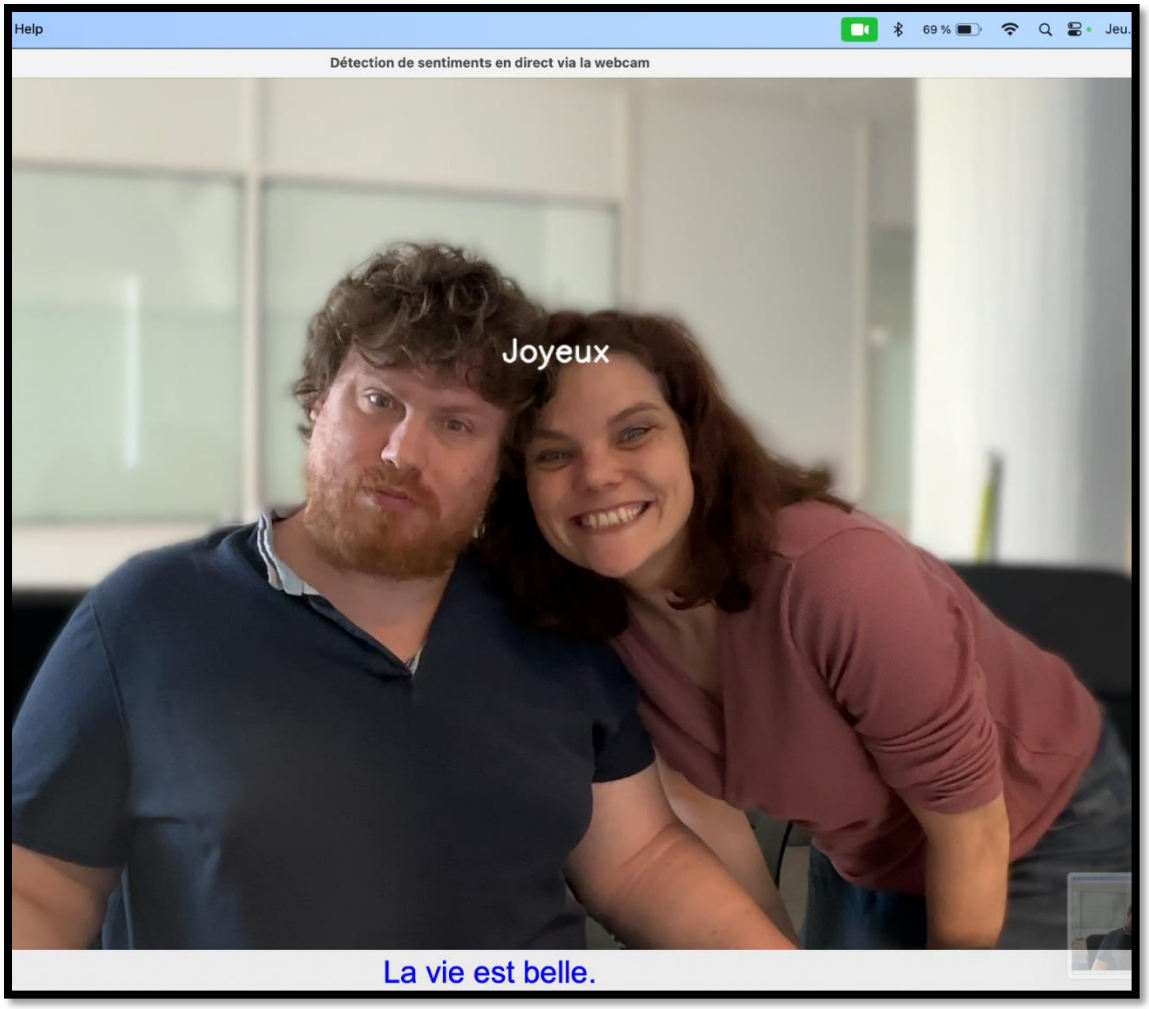
Ce projet illustre notre capacité à travailler avec des données réelles et à appliquer des techniques avancées d'apprentissage automatique pour résoudre des problèmes complexes. Bien que notre modèle soit basé sur des fondations existantes, notre contribution réside dans notre capacité à adapter et à optimiser ces outils pour répondre à des besoins spécifiques dans le domaine de la détection des émotions dans les vidéos.

Références :

https://www.kaggle.com/code/laxmivatsalyadaita/emotion-detector-fer-2013/notebook

https://meritis.fr/blog/reconnaissance-des-emotions/

https://github.com/keras-team/keras



Annexe :

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 48, 48, 32)	320
conv2d_1 (Conv2D)	(None, 48, 48, 64)	18,496
batch_normalization (BatchNormalization)	(None, 48, 48, 64)	256
max_pooling2d (MaxPooling2D)	(None, 24, 24, 64)	0
conv2d_2 (Conv2D)	(None, 24, 24, 128)	204,928
batch_normalization_1 (BatchNormalization)	(None, 24, 24, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 128)	0
conv2d_3 (Conv2D)	(None, 12, 12, 512)	590,336
batch_normalization_2 (BatchNormalization)	(None, 12, 12, 512)	2,048
max_pooling2d_2 (MaxPooling2D) conv2d_4 (Conv2D)	(None, 6, 6, 512) (None, 6, 6, 512)	0 2,359,808
batch_normalization_3 (BatchNormalization)	(None, 6, 6, 512)	2,048
max_pooling2d_3 (MaxPooling2D)	(None, 3, 3, 512)	0
conv2d_5 (Conv2D)	(None, 3, 3, 512)	2,359,808
batch_normalization_4 (BatchNormalization)	(None, 3, 3, 512)	2,048
max_pooling2d_4 (MaxPooling2D)	(None, 2, 2, 512)	0
conv2d_6 (Conv2D)	(None, 2, 2, 512)	2,359,808
batch_normalization_5 (BatchNormalization)	(None, 2, 2, 512)	2,048
max_pooling2d_5 (MaxPooling2D)	(None, 1, 1, 512)	0
conv2d_7 (Conv2D)	(None, 1, 1, 512)	2,359,808
batch_normalization_6 max_pooling2d_6 (MaxPooling2D)	(None, 1, 1, 512) (None, 1, 1, 512)	2,048 0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 256)	131,328
batch_normalization_7	(None, 256)	1,024

(BatchNormalization)		
dense_1 (Dense)	(None, 512)	131,584
batch_normalization_8 (BatchNormalization)	(None, 512)	2,048
dense_2 (Dense)	(None, 512)	262,656
batch_normalization_9 (BatchNormalization)	(None, 512)	2,048
dense_3 (Dense)	(None, 512)	262,656
batch_normalization_10 (BatchNormalization)	(None, 512)	2,048
dense_4 (Dense)	(None, 512)	262,656
batch_normalization_11 (BatchNormalization)	(None, 512)	2,048
dense_5 (Dense)	(None, 7)	3,591