Classificació d'imatges CNN amb Keras

Aquest projecte consisteix en la classificació d'imatges mitjançant una xarxa neuronal convolucional (CNN) utilitzant la llibreria Keras. S'ha utilitzat el dataset "Places" que conté imatges de diferents llocs al voltant del món.

Descripció de les llibreries a utilitzar:

os
random
shutill
numpy
matplotlib
sklearn
keras
tensorflow

Estructura del projecte

Preparació de les dades
Les imatges originals es troben a la carpeta `places`
que conté un 10% del total de 123.634 imatges. Nosaltres hem preparat un dataset reduit places_reduced. Aquest dataset conte 23 carpetes de la a-w i dins de cada lletra conte un numero de carpetes amb el nom d'una etiqueta i en aquestes carpetes es on es troben les imatges. Aquestes imatges s'han dividit en tres conjunts: train, test i validació. La proporció d'aquesta divisió és del 70%, 20% i 10%, respectivament. En total, hi ha 86.448 imatges a la carpeta de train, 24.644 a la de test i 12.542 a la de validació.

Metodologia

Primer hem preprocessat les imatges, les de train les hem augmentat per a conseguir un millor equilibri, i despres hem preprocessat una normalització a Train, Test i Validació. hem dividit train, amb sklearn, i hem fet una arquitectura CNN per entrenar, pero hem patit molts sobreajustaments i overfiting. Despres de diferents intents, Hem decidit utilitzar directament el train_generator, per entrenar.

Arquitectura de la xarxa neuronal convolucional

L'arquitectura de la xarxa neuronal convolucional utilitzada és la següent:

```
model = Sequential()

# Afegim la primera capa convolucional i normalització
model.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=(128, 128, 3)))
model.add(BatchNormalization())

# Afegim capes convolucionals i de pooling
model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

# Afegim regularització
model.add(Dropout(0.25))

# Afegim més capes convolucionals i de pooling
model.add(Conv2D(filters=128, kernel_size=(3,3), activation='relu'))
model.add(Conv2D(filters=128, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(2,2), strides=(2,2), activation='relu'))  # Modificació per solucion
model.add(Conv2D(filters=256, kernel_size=(3,3), activation='relu'))
model.add(Conv2D(filters=256, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

# Afegim regularització
model.add(Dropout(0.5))

# Apliquem flatten per passar a una capa totalment connectada
model.add(Flatten())

# Afegim capes totalment connectades
model.add(Dense(units=512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(units=256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(units=128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Afegim l'última capa totalment connectada amb activació softmax per la classificació multiclasse
model.add(Dense(units=num_classes, activation='softmax'))

# Compilem el model amb l'optimitzador Adam
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

```

Aquesta arquitectura és una xarxa neuronal convolucional (CNN) utilitzada per a la classificació d'imatges. A continuació, es descriuen les funcions i característiques de les diferents capes de la xarxa:

Capa convolucional i normalització: la primera capa convolucional té 32 filtres amb una mida del kernel de 3x3 i una funció d'activació ReLU. La normalització per lot s'utilitza per normalitzar les sortides de la capa anterior.
Capes convolucionals i de pooling: la segona i tercera capes convolucionals tenen 64 filtres amb una mida de kernel de 3x3 i una funció d'activació ReLU. Després d'això, s'aplica una capa de max pooling amb una mida de 2x2 per reduir la mida de la representació.
Regularització: s'aplica una regularització Dropout amb una taxa de 0.25 per reduir el sobreajustament.
Més capes convolucionals i de pooling: la quarta capa convolucional té 128 filtres amb una mida de kernel de 3x3 i una funció d'activació ReLU. A continuació, s'aplica una capa de max pooling amb una mida de 2x2. Les següents capes convolucionals tenen

256 filtres amb una mida de kernel de 2x2 i 3x3, respectivament, amb una funció d'activació ReLU i una capa de max pooling amb una mida de 2x2.
Regularització: s'aplica una regularització Dropout amb una taxa de 0.5 per reduir el sobreajustament.
Capes totalment connectades: es fa un flatten per transformar la sortida de les capes anteriors en un vector i es passa a través de tres capes totalment connectades amb funcions d'activació ReLU i normalització per lot. La sortida de l'última capa totalment connectada té una funció d'activació softmax per a la classificació multiclasse.
Compilació: el model s'entrena amb l'optimitzador Adam amb una funció de pèrdua de "categorical_crossentropy" per classificació multiclasse i s'avalua en precisió.
En resum, aquesta arquitectura consta de diverses capes convolucionals i de max pooling per extreure característiques de les imatges, seguides de capes totalment connectades per a la classificació. Les capes de regularització Dropout i normalització per lot es fan servir per evitar el sobreajustament.

Entrenament del model

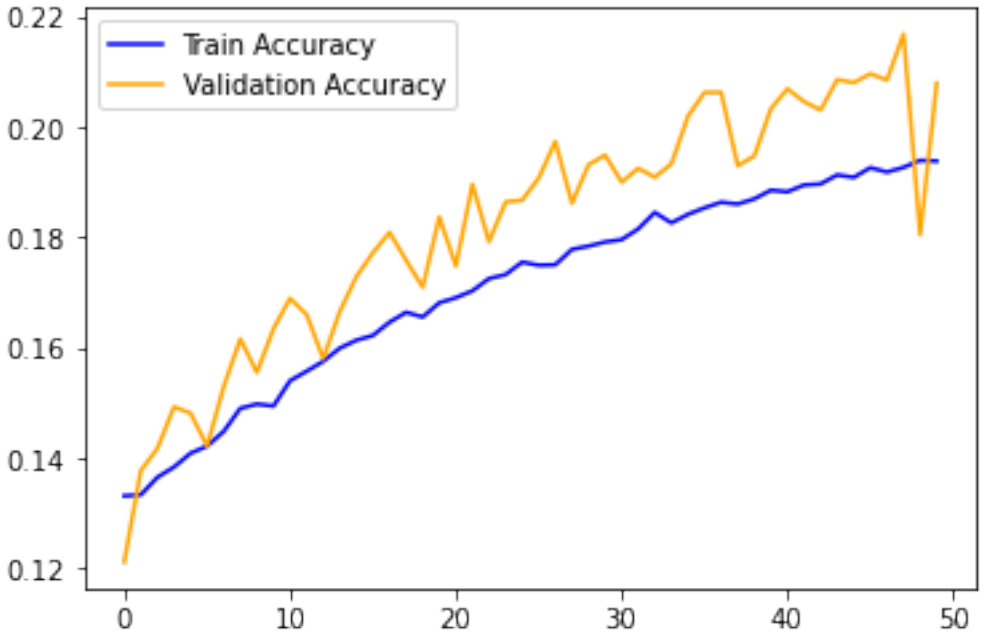
Hem relitzat entrenament a train_generator, epochs=50, hem anat guardant els models en diferents epochs quan val_loss es redueix.

```
EPOCHS = 50

# Define el nom del fitxer on es guardarà el model
checkpoint_filepath = 'model-{epoch:02d}-{val_loss:.2f}.h5'
early_stop_callback = EarlyStopping(
    monitor='val_loss',  # monitoritzem la pèrdua en el conjunt de validació
    patience=5,  # aturem l'entrenament si no hi ha millora durant 5 èpoques
    restore_best_weights=True  # restaurem els pesos del millor model entrenat fins al moment
)
# Defineix el callback ModelCheckpoint

checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=False,
    monitor='val_loss',
    mode='min',
    save_best_only=True
)

history = model5c.fit(
    train_generator,
    steps_per_epoch=STEPS_PER_EPOCH,
    epochs=EPOCHS,
    validation_data=val_generator,
    validation_steps=VALIDATION_STEPS,
    callbacks=[checkpoint_callback]
)
```



El resultat obtingut no ha sigut del tot bo, es necessari millorar l'arquitectura i entrenar molt més.

```
Test accuracy: 0.20970621705055237
```

```
Model 2 test accuracy: 0.20625710487365723
Model 3 test accuracy: 0.17460639774799347
Model 4 test accuracy: 0.2094627469778061
Model 5 test accuracy: 0.18255965411663055
Model 6 test accuracy: 0.1398717761039734
Model 7 test accuracy: 0.12510144710540771
Model 8 test accuracy: 0.14287453889846802
Model 9 test accuracy: 0.16612562537193298
Model 10 test accuracy: 0.14835253357887268
Model 11 test accuracy: 0.1993994414806366
Model 12 test accuracy: 0.14445707201957703
Model 13 test accuracy: 0.1729832887649536
Model 14 test accuracy: 0.1593897044658661
```