|  |
| --- |
| from keras.models import Sequential |
|  | from keras.layers import Convolution2D |
|  | from keras.layers import MaxPooling2D |
|  | from keras.layers import Flatten |
|  | from keras.layers import Dense |
|  | from keras.models import model\_from\_json |
|  | import matplotlib.pyplot as plt |
|  | import warnings |
|  | warnings.filterwarnings('ignore') |
|  | batch\_size = 32 |
|  |  |
|  | from tensorflow.keras.preprocessing.image import ImageDataGenerator |
|  |  |
|  | # All images will be rescaled by 1./255 |
|  | train\_datagen = ImageDataGenerator(rescale=1/255) |
|  |  |
|  | # Flow training images in batches of 128 using train\_datagen generator |
|  | train\_generator = train\_datagen.flow\_from\_directory( |
|  | 'body', # This is the source directory for training images |
|  | target\_size=(200, 200), # All images will be resized to 200 x 200 |
|  | batch\_size=batch\_size, |
|  | # Specify the classes explicitly |
|  | classes = ['00-front','01-rear','02-side'], |
|  | # Since we use categorical\_crossentropy loss, we need categorical labels |
|  | class\_mode='categorical') |
|  |  |
|  | import tensorflow as tf |
|  | #cnn Model |
|  | model = tf.keras.models.Sequential([ |
|  | # Note the input shape is the desired size of the image 200x 200 with 3 bytes color |
|  | # The first convolution |
|  | tf.keras.layers.Conv2D(16, (3,3), activation='relu', input\_shape=(200, 200, 3)), |
|  | tf.keras.layers.MaxPooling2D(2, 2), |
|  | # The second convolution |
|  | tf.keras.layers.Conv2D(32, (3,3), activation='relu'), |
|  | tf.keras.layers.MaxPooling2D(2,2), |
|  | # The third convolution |
|  | tf.keras.layers.Conv2D(64, (3,3), activation='relu'), |
|  | tf.keras.layers.MaxPooling2D(2,2), |
|  | # The fourth convolution |
|  | tf.keras.layers.Conv2D(64, (3,3), activation='relu'), |
|  | tf.keras.layers.MaxPooling2D(2,2), |
|  | # The fifth convolution |
|  | tf.keras.layers.Conv2D(64, (3,3), activation='relu'), |
|  | tf.keras.layers.MaxPooling2D(2,2), |
|  | # Flatten the results to feed into a dense layer |
|  | tf.keras.layers.Flatten(), |
|  | # 128 neuron in the fully-connected layer |
|  | tf.keras.layers.Dense(128, activation='relu'), |
|  | # 5 output neurons for 5 classes with the softmax activation |
|  | tf.keras.layers.Dense(3, activation='softmax') |
|  | ]) |
|  |  |
|  | model.summary() |
|  |  |
|  | from tensorflow.keras.optimizers import RMSprop |
|  | early = tf.keras.callbacks.EarlyStopping(monitor='val\_loss',patience=5) |
|  | model.compile(loss='categorical\_crossentropy', |
|  | optimizer=RMSprop(lr=0.001), |
|  | metrics=['accuracy']) |
|  |  |
|  | total\_sample=train\_generator.n |
|  |  |
|  | n\_epochs = 20 |
|  |  |
|  | history = model.fit\_generator( |
|  | train\_generator, |
|  | steps\_per\_epoch=int(total\_sample/batch\_size), |
|  | epochs=n\_epochs, |
|  | verbose=1) |
|  |  |
|  |  |
|  |  |
|  |  |
|  | model.save('body.h5') |
|  |  |
|  |  |
|  |  |
|  | acc = history.history['accuracy'] |
|  |  |
|  | loss = history.history['loss'] |
|  |  |
|  | epochs = range(1, len(acc) + 1) |
|  |  |
|  | # Train and validation accuracy |
|  | plt.plot(epochs, acc, 'b', label=' accurarcy') |
|  |  |
|  | plt.title(' accurarcy') |
|  | plt.legend() |
|  |  |
|  | plt.figure() |
|  |  |
|  | # Train and validation loss |
|  | plt.plot(epochs, loss, 'b', label=' loss') |
|  | plt.title(' loss') |
|  | plt.legend() |
|  | plt.show() |