Image Classification – Deep Learning Project in Python with Keras

Image classification is a fascinating deep learning project. Specifically, image classification comes under the computer vision project category.

In this project, we will build a convolution neural network in Keras with python on a CIFAR-10 dataset. First, we will explore our dataset, and then we will train our neural network using python and Keras.

### About Image Classification Dataset

CIFAR-10 is a very popular computer vision dataset. This dataset is well studied in many types of deep learning research for object recognition.

This dataset consists of 60,000 images divided into 10 target classes, with each category containing 6000 images of shape 32\*32. This dataset contains images of low resolution (32\*32), which allows researchers to try new algorithms. The 10 different classes of this dataset are:

1. Airplane
2. Car
3. Bird
4. Cat
5. Deer
6. Dog
7. Frog
8. Horse
9. Ship
10. Truck

CIFAR-10 dataset is already available in the datasets module of Keras. We do not need to download it; we can directly import it from ‘keras.datasets’.

### Image classification on CIFAR-10:

1. Load the dataset from keras datasets module

Code:

from keras.datasets import cifar10  
import matplotlib.pyplot as plt  
   
(train\_X,train\_Y),(test\_X,test\_Y)=cifar10.load\_data()

2.Import the required layers and modules to create our convolution neural net architecture

Code:

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import Dropout

from keras.layers import Flatten

from keras.constraints import MaxNorm

from keras.optimizers import SGD

from tensorflow.keras.layers import Conv2D, MaxPooling2D

from keras.utils import to\_categorical

3. Convert the pixel values of the dataset to float type and then normalize the dataset

Code:

train\_x=train\_X.astype('float32')  
test\_X=test\_X.astype('float32')  
   
train\_X=train\_X/255.0  
test\_X=test\_X/255.0

* Converting pixels values of the datasets to a float32 type and normalizing them (dividing by 225.0) is a common preprocessing step in training neural network.
* By converting pixel values to float32 and normalizing them, you ensure that the input data is consistent, numerically stable, and optimized for efficient and effective training of the neural network. This preprocessing step is crucial for achieving good performance and faster convergence during model training.

4. Now perform the one-hot encoding for target classes

Code:

train\_Y=to\_categorical(train\_Y)

test\_Y=to\_categorical(test\_Y)

num\_classes=test\_Y.shape[1]

* One-hot encoding is a crucial preprocessing step when dealing with categorical target variables in classification tasks, particularly for neural networks.
* By converting categorical labels into a one-hot encoded format, we help the neural network understand and learn from the data more efficiently and accurately.

5. Create the sequential model and add the layers

* The created model is a Convolutional Neural Network designed for image classification tasks. It uses convolutional layers to extract spatial features from the input images, pooling layers to reduce dimensionality, and fully connected layers to perform classification based on the extracted features.
* Regularization techniques like dropout and max-norm constraints are used to prevent overfitting and improve generalization.
* The number of neurons in the dense layers and the architecture overall are designed to balance between model complexity and performance.
* In this model we used ReLU activation function.
* **ReLU** is used in hidden layers to introduce non-linearity, efficiently propagate gradients, and ensure sparsity.
* **Softmax** is used in the output layer to convert the model's output into a probability distribution suitable for multi-class classification tasks.
* The use of ReLU (Rectified Linear Unit) in neural networks is desirable because it introduces nonlinearity, enabling the network to recognize complex patterns ReLU is computationally efficient due to the simple threshold operation, and it helps reduce the risk of missing problems, resulting in faster and more effective training involving deeper networks

Code:

model=Sequential()  
model.add(Conv2D(32,(3,3),input\_shape=(32,32,3),  
 padding='same',activation='relu',  
 kernel\_constraint=maxnorm(3)))  
model.add(Dropout(0.2))  
model.add(Conv2D(32,(3,3),activation='relu',padding='same',kernel\_constraint=maxnorm(3)))  
model.add(MaxPooling2D(pool\_size=(2,2)))  
model.add(Flatten())  
model.add(Dense(512,activation='relu',kernel\_constraint=maxnorm(3)))  
model.add(Dropout(0.5))  
model.add(Dense(num\_classes, activation='softmax'))

6. Configure the optimizer and compile the model

* In this model i used SGD optimizer.it is used with specific parameters.
* learning\_rate=0.01, momentum=0.9, nesterov=False these are the parameters used in this SGD optimizer
* Learning\_rate it defines the step size for each iteration while moving towards the minimum of the loss function.
* Momentum helps to accelerate gradients vectors in the right direction for faster converging.
* And last parameter is nesterov which is a variant of momentum which is not used in this model.

Code:

sgd = SGD(learning\_rate=0.01, momentum=0.9, nesterov=False)

model.compile(loss='categorical\_crossentropy',

optimizer=sgd,

metrics=['accuracy'])

model.summary()

7. Train the model

Code:

model.fit(train\_X,train\_Y,

validation\_data=(test\_X,test\_Y),

epochs=10,batch\_size=32)

* After executing this the model will get trained based on the number epoch we give.
* After the model is trained, we check the accuracy of the model.

Code:

\_,acc= model.evaluate(test\_X,test\_Y)

print(acc\*100)

With this we get accuracy of the model.

8. Make a dictionary to map to the output classes and make predictions from the model

* In this step we make the prediction for the input image we give and the model we prepared will predict the image.

Code:

results={

0:'aeroplane',

1:'automobile',

2:'bird',

3:'cat',

4:'deer',

5:'dog',

6:'frog',

7:'horse',

8:'ship',

9:'truck'

}

from PIL import Image

import numpy as np

def preprocess\_image(image\_path):

# Open the image

img = Image.open(image\_path)

# Convert to grayscale if it's not already

if img.mode != 'L':

img = img.convert('L')

# Resize to 28x28 pixels

img = img.resize((28, 28))

# Convert to numpy array and normalize

img\_array = np.array(img).astype('float32') / 255.0

# Reshape to (1, 28, 28, 1) for model input

img\_array = img\_array.reshape(1, 28, 28, 1)

return img\_array

image\_path='dog.png'

im=Image.open(image\_path)

im=im.resize((32,32))

im=np.expand\_dims(im,axis=0)

im=np.array(im)

pred=model.predict([im])

pred\_class = np.argmax(pred, axis=-1)[0]

import matplotlib.pyplot as plt

# Display the image

i = Image.open(image\_path).convert('L')

plt.imshow(i, cmap='gray')

plt.axis('off')

plt.title(f"Predicted\_class:{pred\_class} Animal:{results[pred\_class]}")

plt.show()

* We can test the result on your custom image input. To improve accuracy, try increasing the epoch count to 25 for training.

Summary:

* In this keras deep learning Project, we talked about the image classification paradigm for digital image analysis.
* I used CIFAR-10 dataset and its classes. Build a CNN model for image classification on CIFAR-10 dataset.