ChatGPT citation

1. " I want to show loaded image and probability graph in one figure." prompt. ChatGPT, May 24 Version, OpenAI, 20 June. 2023, chat.openai.com/chat.

To display the loaded image and the probability graph in one figure, you can modify the code as follows:

import torch

import torch.nn.functional as F

import numpy as np

import matplotlib.pyplot as plt

import os

# Load the model (replace 'model' with your actual model)

model = YourModel()

# Create a figure and axes for subplots

fig, axes = plt.subplots(2, 1, figsize=(6, 12))

# Iterate over the images

for file\_num in range(1, 9):

for index in range(10):

# Load the image

image\_path = f"/content/drive/MyDrive/DeepLearning/MNIST\_normalized/test{file\_num}/normalized\_{index}.jpeg"

image = Image.open(image\_path)

# Preprocess the image (similar to previous code)

...

# Convert the normalized image to a PyTorch tensor

normalized\_img\_tensor = torch.from\_numpy(normalized\_img).float()

# Perform inference

with torch.no\_grad():

logits = model.forward(normalized\_img\_tensor.view(1, 784))

probabilities = F.softmax(logits, dim=1)

probabilities = probabilities.data.numpy().squeeze()

# Print the probabilities

for p in probabilities:

print(round(p\*100, 3))

# Clear the axes

axes[0].clear()

axes[1].clear()

# Plot the image

axes[0].imshow(image, cmap='gray')

axes[0].axis("off")

# Plot the probability distribution

axes[1].barh(np.arange(10), probabilities)

axes[1].set\_yticks(np.arange(10))

axes[1].set\_xlim(0, 1.1)

axes[1].set\_xlabel("Probability")

axes[1].set\_ylabel("Digit")

# Show the figure

plt.show()

# Wait for a short time before showing the next image

plt.pause(0.1)

# Close the figure

plt.close(fig)

In this modified code, a figure with two subplots is created at the beginning. Within the loop that iterates over the images, the axes are cleared using **axes[0].clear()** and **axes[1].clear()** before plotting the image and probability distribution respectively. The loaded image is displayed in the first subplot using **axes[0].imshow**, and the probability distribution is plotted in the second subplot using **axes[1].barh**. The figure is shown using **plt.show()**, and a short pause is added using **plt.pause(0.1)** to allow for better visualization. Finally, the figure is closed using **plt.close(fig)**.

This will display the loaded image and the corresponding probability graph in one figure, and then proceed to show the next image and probability graph after a short pause.

텍스트, 스크린샷, 소프트웨어, 웹사이트이(가) 표시된 사진

자동 생성된 설명

1. " I want to preprocess MNIST-like data with my handwriting. How can I do that with using one of these libraries matplotlib/pillow/openCV?” prompt. ChatGPT, May 24 Version, OpenAI, 20 June. 2023, chat.openai.com/chat.

To preprocess MNIST-like data with your handwriting using one of the libraries like Matplotlib, Pillow, or OpenCV, you can follow these general steps:

1. Capture or import your handwriting samples: You can either scan your handwritten digits or capture images using a camera or smartphone. Make sure to capture clear and well-lit images.
2. Load the image using the library: Depending on the library you choose, you can load the image using the respective functions. For example, if you're using Pillow, you can use the **Image.open()** function to load the image.

from PIL import Image

image = Image.open("handwritten\_digit.jpg")

1. Convert the image to grayscale: Since MNIST data is grayscale, convert the loaded image to grayscale if it's not already in that format. This step is essential for simplifying the data and reducing complexity.

image = image.convert("L")

1. Apply image preprocessing techniques: Various techniques can be applied to preprocess the image and enhance the quality of the handwritten digits. Some common preprocessing techniques include:
   * Thresholding: Convert the image to binary by setting a threshold value to separate the foreground (digits) from the background.
   * Denoising: Remove noise from the image to improve digit clarity.
   * Normalization: Scale the image to a specific size or range to make it consistent with the MNIST dataset.

You can use the functions provided by the libraries to apply these techniques. For example, OpenCV offers various image processing functions like **cv2.threshold()**, **cv2.fastNlMeansDenoising()**, etc.

1. Resize the image: Resize the preprocessed image to match the dimensions of the MNIST dataset. The original MNIST dataset contains 28x28 pixels, so you need to resize your image accordingly.

resized\_image = image.resize((28, 28))

1. Convert the image to a numpy array: Convert the resized image to a numpy array to make it compatible with machine learning libraries like TensorFlow or PyTorch.

import numpy as np

img\_array = np.array(resized\_image)

1. Normalize the pixel values: Normalize the pixel values of the image to a specific range, usually between 0 and 1. This step helps in training the neural network more effectively.

normalized\_img = img\_array / 255.0

Now you have preprocessed your handwriting image and converted it into a format similar to the MNIST dataset. You can use this preprocessed data for tasks such as digit recognition or training a neural network model.

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

1. " I want to use only black and white without gray. How can I change the values that are gray around, especially not the part where the numbers are written, to completely black (0)?” prompt. ChatGPT, May 24 Version, OpenAI, 20 June. 2023, chat.openai.com/chat.

To convert the gray areas surrounding the numbers to pure black (0), you can apply a thresholding operation to the image. Thresholding will convert pixel values above a certain threshold to white (255) and pixel values below the threshold to black (0). Here's an updated version of the code that includes thresholding:

import numpy as np

import cv2

from PIL import Image

# Load the image

image = Image.open("/content/drive/MyDrive/DeepLearning/MNIST\_0.jpeg")

# Convert the image to grayscale

gray\_image = image.convert("L")

# Invert the colors

inverted\_image = 255 - np.array(gray\_image)

# Apply thresholding to convert gray areas to pure black

\_, thresholded\_img = cv2.threshold(inverted\_image, 127, 255, cv2.THRESH\_BINARY)

# Resize the thresholded image to (28, 28)

resized\_image = Image.fromarray(thresholded\_img).resize((28, 28))

# Convert the resized image to a numpy array

img\_array = np.array(resized\_image)

# Expand the dimensions of the image array to (28, 28, 1)

normalized\_img = np.expand\_dims(img\_array, axis=2)

# Normalize the image

normalized\_img = normalized\_img / 255.0

# Check the shape of the normalized image

print(normalized\_img.shape)

In this code, the line **\_, thresholded\_img = cv2.threshold(inverted\_image, 127, 255, cv2.THRESH\_BINARY)** applies a thresholding operation using a threshold value of 127. Pixels with values above 127 are set to 255 (white), and pixels with values below 127 are set to 0 (black). The resulting thresholded image is then resized and processed as before.

After running this code, the **normalized\_img** will contain the thresholded and normalized image with black text on a white background, and the gray areas will be completely black (0).

텍스트, 스크린샷, 소프트웨어, 웹사이트이(가) 표시된 사진

자동 생성된 설명