Artificial Intelligence

Tutorial week 7 – Computer vision

COMP3411/9814

1 OpenCV

This tutorial will be focused on computer vision in Python. Most of the computer vision tasks use the OpenCV library. OpenCV supports not only Python programming language but also C++, Java, etc. This open-source library is focused on computer vision and image-processing tasks. Usually, you will see this library is used along with other libraries such as Numpy, Matplotlib, Keras, or TensorFlow.

2 Setup

There are different ways to install the library. You can download the library directly from the official website or install it in Anaconda using either the Anaconda Prompt or the Anaconda Navigator. In Colab, the library is already install but usually you need to connect it with a file repository (e.g., Google Drive or GitHub) to access image files. You can find more details about installing the library in the following links:

- Official site.
- Installation with Anaconda Prompt.
- Installation with Anaconda Navigator.
- Using external data in Colab.

This tutorial will be conducted using Anaconda + Spyder, however, you can use any other API if you prefer. To work with the library, we need to import it as follows:

```
import cv2
```

3 OpenCV basics

Digital coloured images can be shown based on different colour spaces. One of the most common ones are three channels of RGB, indicating red, green and blue values in an image. In OpenCV, BGR sequence is used instead of RGB.

Digital images get noisy due to a number of factors such as poor lighting conditions, electrical interference, high ISO settings, analog to digital converting, etc. Noises also have different types such as Gaussian noise, salt and pepper noise, quantisation noise, etc. One of the most basic ways to tackle this issue is using filters. In this tutorial, you will develop algorithms for two basic filters, mean and median filters.

The mean filter is a simple sliding-window spatial filter that replaces the centre value in the window with the average (mean) of all the pixel values in the window. The window, or kernel, is usually square but can be any shape. A median filter is a nonlinear filter in which each output pixel is the median value of the window (neighbours).

3.1 Exercise:

An image named *parrots.jpg* has been given to you.

1. Read the image and show it in Spyder as a coloured and gray-scale image.

2. Print both images and their shapes. Can you make any connection between the numbers in the matrix and the image that you see?

3. Save a copy of the gray-scale image. Name the image as *parrots_gray_test.jpg*.

4. Make a copy of the original image and name it *parrots_frame_test.jpg*. Put a white 75-pixel size frame on your image and display it (hint: you should change the value of pixels).

5. Make a random noise matrix with the same dimension as your picture *parrots.jpg* in the range of [0, 50] and display your noise as an image.

6. Add this noise to your gray-scale image. Show the new image and saved it as *parrots_noisy_test.jpg*.

3.2 Exercise:

1. Develop a function that takes an image, and applies the mean filter, and shows the outcome. Try your function with the *parrots_noisy_test.jpg* from the previous exercise (consider the window-size = 3×3).

2. Develop a function that takes an image, and applies the median filter, and shows the outcome. Try your function with the *parrots_noisy_test.jpg* from the previous exercise (consider the window-size = 3×3).

3. OpenCV library offers both mean and median filters. Apply OpenCV mean (blur) and median filters on *parrots_noisy_test.jpg* and compare them to your developed algorithm. What are the pros and cons of using ready-to-use functions?

4. Try to set different kernel sizes (window-size) for your filters. What is the optimum size?

4 OpenCV intermediate

The OpenCV library is well-known for doing image processing-tasks, including videos. Videos are a series of frames (images) per second. Thus, if for a gray-scale image we have 250×250 dimension and we turn this image to a video, it will be $250 \times 250 \times t$ dimension, with t indicating time. Therefore, a colour video will have 4 dimensions (width $\times$ height $\times$ channels $\times$ t) .

Our main understanding from real-world objects (object detection) is coming from edge detection and other processes such as colour evaluation. It means our brain detects the edges of an item we are looking at and matches it with our previous knowledge. This is how we differ a circle from a square. In computer vision, we try to imitate the same approach for object detection. One of the traditional ways of finding edges in an object is using the Canny edge detector that relies on gradient of the image (pixels). Information regrading how this algorithm works can be find here.

4.1 Exercise:

1. Play the given video named *tom_and_jerry.mp4* in a media player app. Using OpenCV, print the number of frames per second, the total number of frames in the video, and the duration of the video in seconds. Display the frame number 1002. Describe what you see.

2. Use the OpenCV Canny filter to find the edges of the video and display it along with the original one. Can you understand the content of the video only if you see the second video (edge video)?

3. Change the lower and upper threshold of the Canny filter and report how it affects edge detection.

5 Neural network-based image classification

As you have seen so far, dealing with images due to their dimension is computationally expensive. If we want to train a neural network to learn about an image dataset, it will be expensive and we need to provide many samples since there are many features (pixels) to learn about. However, providing the neural network with a big dataset is not always possible since data collection and labelling can be expensive as well. One of the solutions is using transfer learning. Transfer learning is a technique that uses a deep neural network knowledge that is already trained on a big dataset, and then add some extra layers to the end of the network and train it with the target dataset. More information regarding this approach can be found here.

To do this exercise you need to know some details about the deep neural network VGG16 and the dataset that we are going to use. VGG16 is a convolutional neural network model for image classification tasks. This network has 16 layers and relies on a large number of hyper-parameters. VGG16 is considered one of the best deep convolutional neural networks. More information regarding this network is here.

Moreover, in this exercise, we will use a dataset named MNIST. The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. MNIST is not a small dataset that needs transfer learning necessarily; however, in the following exercise, we will use it for our transfer learning technique.

5.1 Exercise:

1. Use the following line in your command prompt to install the pool of datasets in Tensorflow.

   ```
   pip install tensorflow_datasets
   ```

   After that, you can call the necessary libraries:

   ```
   from tensorflow.keras.utils import to_categorical
   import tensorflow as tf
   import datetime
   import matplotlib.pyplot as plt
   ```

2. Load the MNIST dataset. The original images in the dataset are 28×28 and are between 0-255. See the diagram in Figure 1, we call this part **pre-processing** which is the steps that the image should take to be prepared to be fed into the network. Follow these steps and explain why we divided the images to 255?

3. Load the pre-trained VGG16 model and download its weights, downloading the weights takes some time. Read the weights regarding VGG16 trained on *imagenet* dataset (see below).

   ```
   base_model = VGG16(weights="imagenet", input_shape=(32, 32, 3))
   ```

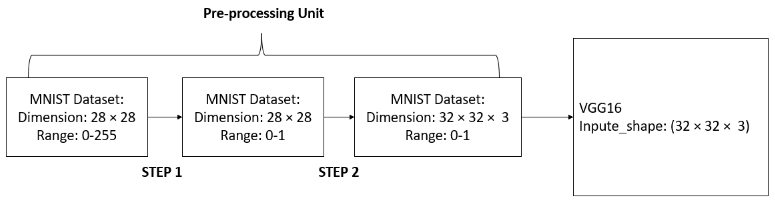
Pre-processing Unit
MNIST Dataset: Dimension: 28 × 28 Range: 0-255
MNIST Dataset: Dimension: 28 × 28 Range: 0-1
MNIST Dataset: Dimension: 32 × 32 × 3 Range: 0-1
VGG16 Inpute_shape: (32 × 32 × 3)
STEP 1
STEP 2

Figure 1: Image preprocessing steps.

4. Show the VGG16 model summary and report how many parameters, in general, should be trained.

   ```
   base_model.summary()
   ```

5. Add the following layers to your VGG16 network and train your new network for 10 epochs, validation split=0.2, and batch size=32. Consider your optimizer is Adam', your loss is categorical cross-entropy, and metric is accuracy.

   ```
   flatten_layer = layers.Flatten()
   dense_layer_1 = layers.Dense(50, activation='relu')
   dense_layer_2 = layers.Dense(20, activation='relu')
   prediction_layer = layers.Dense(10, activation='softmax')
   ```

6. Plot accuracy and loss of training and test sets based on epochs in two different plots (one plot for accuracy of test and train and the second plot for the loss of test and train) and report the training time.

7. Read 10th image in the dataset (image index=10), feed it to the trained network and print the predicted and actual class.