Assignment 1

How is our app working?

We have created a simple Flask application to test our filters. This app can be deployed and serve as fully functional REST application, which accepts on input image as bytes, filter\_type as string and returns processed image.

Main route of the app has the following structure:

@*app*.*route*("/apply\_filter", methods=["POST"])

def *apply\_filter*():

  input\_image = request.files['image']

  img: ndarray = cv2.imdecode(np.asarray(bytearray(input\_image.stream.read()), dtype=np.uint8), cv2.IMREAD\_COLOR)

  filter\_type = request.form.get('filter\_type').lower()

  amount\_str = request.form.get('amount')

  amount = int(amount\_str) *if* amount\_str != None *else* 1

  print(f'{img.shape}')

*if* filter\_type == 'gaussian\_blur':

…

*elif* filter\_type == 'greyscale':

…

*elif* filter\_type == 'sharpen':

    plain\_ndarray = some ndarray…

*else*:

*return Error…*

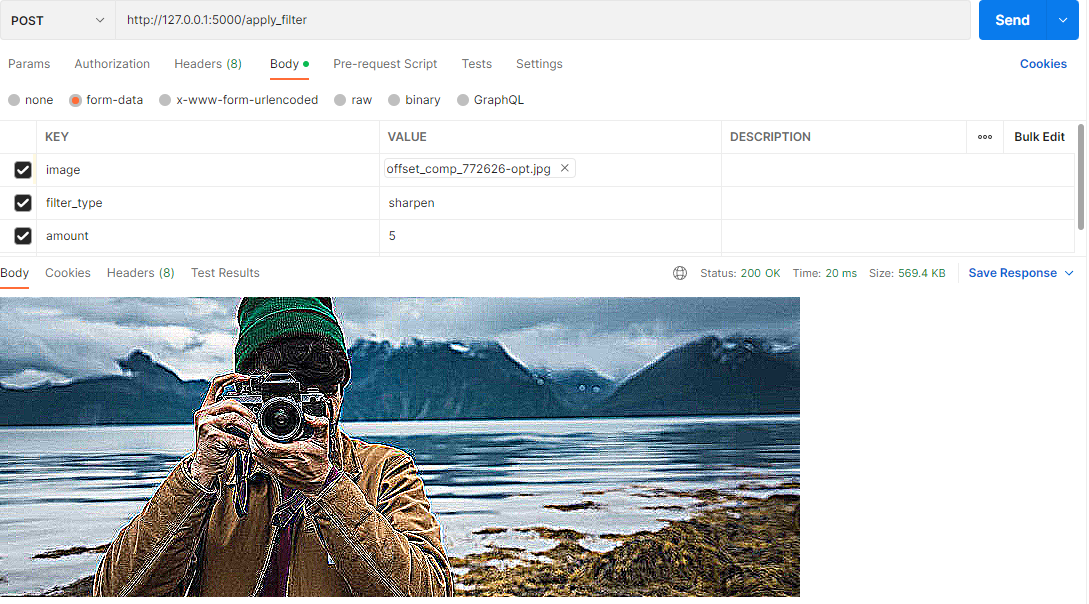
  print(f'{channel\_ndarray.shape}')

  # *channel\_ndarray should be 3 channel*

  encoded, plain\_ndarray = cv2.imencode('.png', channel\_ndarray)

  encoded, plain\_ndarray = cast(tuple[bool, ndarray], (encoded, plain\_ndarray))

*return* send\_file(BytesIO(plain\_ndarray.tobytes()), mimetype='image/gif', download\_name='out.png')

To test our application, we have used Postman. 

Sharpen filter

**Explanation:**

Sharpening technique is used to bring out detail in an image by enhancing the contrast of pixels on edges. To do this we extend the basic edge detector with sharpen\_kernel. We can control the strength of sharpen effect by choosing *amount* value. If amount = 0, then filter has no effect.

**Input shape = (300, 800, 3)**

**Ouput shape = (300, 800, 3)**

**Code:**

basic\_kernel = np.array([[0, 0, 0], [0, 1, 0], [0, 0, 0]])

sharpen\_kernel = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]]) \* amount

kernel = basic\_kernel + sharpen\_kernel

channel\_ndarray: ndarray = cv2.filter2D(img, -1, kernel)

Grayscale

**Explanation:**

From pillow library: when translating a color image to grayscale, the library uses the ITU-R 601-2 luma transform.

We just make dot product of input RPG to corresponding values according to the formula. Or we can use *cv2.cvtColor* function, which produces the same output.

**Input shape = (667, 1000, 3) Output shape = (667, 1000)**

****

**Code:**

# *channel\_ndarray = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY)*

channel\_ndarray = np.dot(img[..., :3], [0.2989, 0.5870, 0.1140])