@ -0,0 +1,1401 @@

{

"cells": [

{

"cell\_type": "code",

"execution\_count": 3,

"id": "d0fd60eb-4c5e-4506-9d07-6726f250632f",

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Requirement already satisfied: tensorflow in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (2.20.0)\n",

"Requirement already satisfied: opencv-python in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (4.12.0.88)\n",

"Requirement already satisfied: matplotlib in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (3.10.6)\n",

"Requirement already satisfied: absl-py>=1.0.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (2.3.1)\n",

"Requirement already satisfied: astunparse>=1.6.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (1.6.3)\n",

"Requirement already satisfied: flatbuffers>=24.3.25 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (25.2.10)\n",

"Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (0.6.0)\n",

"Requirement already satisfied: google\_pasta>=0.1.1 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (0.2.0)\n",

"Requirement already satisfied: libclang>=13.0.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (18.1.1)\n",

"Requirement already satisfied: opt\_einsum>=2.3.2 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (3.4.0)\n",

"Requirement already satisfied: packaging in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (25.0)\n",

"Requirement already satisfied: protobuf>=5.28.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (6.32.0)\n",

"Requirement already satisfied: requests<3,>=2.21.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (2.32.5)\n",

"Requirement already satisfied: setuptools in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (80.9.0)\n",

"Requirement already satisfied: six>=1.12.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (1.17.0)\n",

"Requirement already satisfied: termcolor>=1.1.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (3.1.0)\n",

"Requirement already satisfied: typing\_extensions>=3.6.6 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (4.14.1)\n",

"Requirement already satisfied: wrapt>=1.11.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (1.17.3)\n",

"Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (1.74.0)\n",

"Requirement already satisfied: tensorboard~=2.20.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (2.20.0)\n",

"Requirement already satisfied: keras>=3.10.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (3.11.2)\n",

"Requirement already satisfied: numpy>=1.26.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (2.2.6)\n",

"Requirement already satisfied: h5py>=3.11.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (3.14.0)\n",

"Requirement already satisfied: ml\_dtypes<1.0.0,>=0.5.1 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorflow) (0.5.3)\n",

"Requirement already satisfied: charset\_normalizer<4,>=2 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from requests<3,>=2.21.0->tensorflow) (3.4.3)\n",

"Requirement already satisfied: idna<4,>=2.5 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from requests<3,>=2.21.0->tensorflow) (3.10)\n",

"Requirement already satisfied: urllib3<3,>=1.21.1 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from requests<3,>=2.21.0->tensorflow) (2.5.0)\n",

"Requirement already satisfied: certifi>=2017.4.17 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from requests<3,>=2.21.0->tensorflow) (2025.8.3)\n",

"Requirement already satisfied: markdown>=2.6.8 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorboard~=2.20.0->tensorflow) (3.8.2)\n",

"Requirement already satisfied: pillow in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorboard~=2.20.0->tensorflow) (11.3.0)\n",

"Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorboard~=2.20.0->tensorflow) (0.7.2)\n",

"Requirement already satisfied: werkzeug>=1.0.1 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from tensorboard~=2.20.0->tensorflow) (3.1.3)\n",

"Requirement already satisfied: contourpy>=1.0.1 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from matplotlib) (1.3.3)\n",

"Requirement already satisfied: cycler>=0.10 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from matplotlib) (0.12.1)\n",

"Requirement already satisfied: fonttools>=4.22.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from matplotlib) (4.59.2)\n",

"Requirement already satisfied: kiwisolver>=1.3.1 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from matplotlib) (1.4.9)\n",

"Requirement already satisfied: pyparsing>=2.3.1 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from matplotlib) (3.2.3)\n",

"Requirement already satisfied: python-dateutil>=2.7 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from matplotlib) (2.9.0.post0)\n",

"Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from astunparse>=1.6.0->tensorflow) (0.45.1)\n",

"Requirement already satisfied: rich in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from keras>=3.10.0->tensorflow) (14.1.0)\n",

"Requirement already satisfied: namex in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from keras>=3.10.0->tensorflow) (0.1.0)\n",

"Requirement already satisfied: optree in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from keras>=3.10.0->tensorflow) (0.17.0)\n",

"Requirement already satisfied: MarkupSafe>=2.1.1 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from werkzeug>=1.0.1->tensorboard~=2.20.0->tensorflow) (3.0.2)\n",

"Requirement already satisfied: markdown-it-py>=2.2.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from rich->keras>=3.10.0->tensorflow) (4.0.0)\n",

"Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from rich->keras>=3.10.0->tensorflow) (2.19.2)\n",

"Requirement already satisfied: mdurl~=0.1 in c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages (from markdown-it-py>=2.2.0->rich->keras>=3.10.0->tensorflow) (0.1.2)\n",

"Note: you may need to restart the kernel to use updated packages.\n"

]

}

],

"source": [

"pip install tensorflow opencv-python matplotlib"

]

},

{

"cell\_type": "code",

"execution\_count": 6,

"id": "bf10df3e-e3d0-44bd-bbb4-a71133138add",

"metadata": {},

"outputs": [],

"source": [

"import cv2\n",

"import os \n",

"import random \n",

"import numpy as ny \n",

"from matplotlib import pyplot as plt"

]

},

{

"cell\_type": "code",

"execution\_count": 7,

"id": "9a849d56-c835-491c-aed6-219a0b827dd4",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"\u001b[31mSignature:\u001b[39m\n",

"plt.imshow(\n",

" X: \u001b[33m'ArrayLike | PIL.Image.Image'\u001b[39m,\n",

" cmap: \u001b[33m'str | Colormap | None'\u001b[39m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" norm: \u001b[33m'str | Normalize | None'\u001b[39m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" \*,\n",

" aspect: \u001b[33m\"Literal['equal', 'auto'] | float | None\"\u001b[39m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" interpolation: \u001b[33m'str | None'\u001b[39m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" alpha: \u001b[33m'float | ArrayLike | None'\u001b[39m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" vmin: \u001b[33m'float | None'\u001b[39m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" vmax: \u001b[33m'float | None'\u001b[39m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" colorizer: \u001b[33m'Colorizer | None'\u001b[39m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" origin: \u001b[33m\"Literal['upper', 'lower'] | None\"\u001b[39m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" extent: \u001b[33m'tuple[float, float, float, float] | None'\u001b[39m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" interpolation\_stage: \u001b[33m\"Literal['data', 'rgba', 'auto'] | None\"\u001b[39m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" filternorm: \u001b[33m'bool'\u001b[39m = \u001b[38;5;28;01mTrue\u001b[39;00m,\n",

" filterrad: \u001b[33m'float'\u001b[39m = \u001b[32m4.0\u001b[39m,\n",

" resample: \u001b[33m'bool | None'\u001b[39m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" url: \u001b[33m'str | None'\u001b[39m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" data=\u001b[38;5;28;01mNone\u001b[39;00m,\n",

" \*\*kwargs,\n",

") -> \u001b[33m'AxesImage'\u001b[39m\n",

"\u001b[31mDocstring:\u001b[39m\n",

"Display data as an image, i.e., on a 2D regular raster.\n",

"\n",

"The input may either be actual RGB(A) data, or 2D scalar data, which\n",

"will be rendered as a pseudocolor image. For displaying a grayscale\n",

"image, set up the colormapping using the parameters\n",

"``cmap='gray', vmin=0, vmax=255``.\n",

"\n",

"The number of pixels used to render an image is set by the Axes size\n",

"and the figure \*dpi\*. This can lead to aliasing artifacts when\n",

"the image is resampled, because the displayed image size will usually\n",

"not match the size of \*X\* (see\n",

":doc:`/gallery/images\_contours\_and\_fields/image\_antialiasing`).\n",

"The resampling can be controlled via the \*interpolation\* parameter\n",

"and/or :rc:`image.interpolation`.\n",

"\n",

"Parameters\n",

"----------\n",

"X : array-like or PIL image\n",

" The image data. Supported array shapes are:\n",

"\n",

" - (M, N): an image with scalar data. The values are mapped to\n",

" colors using normalization and a colormap. See parameters \*norm\*,\n",

" \*cmap\*, \*vmin\*, \*vmax\*.\n",

" - (M, N, 3): an image with RGB values (0-1 float or 0-255 int).\n",

" - (M, N, 4): an image with RGBA values (0-1 float or 0-255 int),\n",

" i.e. including transparency.\n",

"\n",

" The first two dimensions (M, N) define the rows and columns of\n",

" the image.\n",

"\n",

" Out-of-range RGB(A) values are clipped.\n",

"\n",

"cmap : str or `~matplotlib.colors.Colormap`, default: :rc:`image.cmap`\n",

" The Colormap instance or registered colormap name used to map scalar data\n",

" to colors.\n",

"\n",

" This parameter is ignored if \*X\* is RGB(A).\n",

"\n",

"norm : str or `~matplotlib.colors.Normalize`, optional\n",

" The normalization method used to scale scalar data to the [0, 1] range\n",

" before mapping to colors using \*cmap\*. By default, a linear scaling is\n",

" used, mapping the lowest value to 0 and the highest to 1.\n",

"\n",

" If given, this can be one of the following:\n",

"\n",

" - An instance of `.Normalize` or one of its subclasses\n",

" (see :ref:`colormapnorms`).\n",

" - A scale name, i.e. one of \"linear\", \"log\", \"symlog\", \"logit\", etc. For a\n",

" list of available scales, call `matplotlib.scale.get\_scale\_names()`.\n",

" In that case, a suitable `.Normalize` subclass is dynamically generated\n",

" and instantiated.\n",

"\n",

" This parameter is ignored if \*X\* is RGB(A).\n",

"\n",

"vmin, vmax : float, optional\n",

" When using scalar data and no explicit \*norm\*, \*vmin\* and \*vmax\* define\n",

" the data range that the colormap covers. By default, the colormap covers\n",

" the complete value range of the supplied data. It is an error to use\n",

" \*vmin\*/\*vmax\* when a \*norm\* instance is given (but using a `str` \*norm\*\n",

" name together with \*vmin\*/\*vmax\* is acceptable).\n",

"\n",

" This parameter is ignored if \*X\* is RGB(A).\n",

"\n",

"colorizer : `~matplotlib.colorizer.Colorizer` or None, default: None\n",

" The Colorizer object used to map color to data. If None, a Colorizer\n",

" object is created from a \*norm\* and \*cmap\*.\n",

"\n",

" This parameter is ignored if \*X\* is RGB(A).\n",

"\n",

"aspect : {'equal', 'auto'} or float or None, default: None\n",

" The aspect ratio of the Axes. This parameter is particularly\n",

" relevant for images since it determines whether data pixels are\n",

" square.\n",

"\n",

" This parameter is a shortcut for explicitly calling\n",

" `.Axes.set\_aspect`. See there for further details.\n",

"\n",

" - 'equal': Ensures an aspect ratio of 1. Pixels will be square\n",

" (unless pixel sizes are explicitly made non-square in data\n",

" coordinates using \*extent\*).\n",

" - 'auto': The Axes is kept fixed and the aspect is adjusted so\n",

" that the data fit in the Axes. In general, this will result in\n",

" non-square pixels.\n",

"\n",

" Normally, None (the default) means to use :rc:`image.aspect`. However, if\n",

" the image uses a transform that does not contain the axes data transform,\n",

" then None means to not modify the axes aspect at all (in that case, directly\n",

" call `.Axes.set\_aspect` if desired).\n",

"\n",

"interpolation : str, default: :rc:`image.interpolation`\n",

" The interpolation method used.\n",

"\n",

" Supported values are 'none', 'auto', 'nearest', 'bilinear',\n",

" 'bicubic', 'spline16', 'spline36', 'hanning', 'hamming', 'hermite',\n",

" 'kaiser', 'quadric', 'catrom', 'gaussian', 'bessel', 'mitchell',\n",

" 'sinc', 'lanczos', 'blackman'.\n",

"\n",

" The data \*X\* is resampled to the pixel size of the image on the\n",

" figure canvas, using the interpolation method to either up- or\n",

" downsample the data.\n",

"\n",

" If \*interpolation\* is 'none', then for the ps, pdf, and svg\n",

" backends no down- or upsampling occurs, and the image data is\n",

" passed to the backend as a native image. Note that different ps,\n",

" pdf, and svg viewers may display these raw pixels differently. On\n",

" other backends, 'none' is the same as 'nearest'.\n",

"\n",

" If \*interpolation\* is the default 'auto', then 'nearest'\n",

" interpolation is used if the image is upsampled by more than a\n",

" factor of three (i.e. the number of display pixels is at least\n",

" three times the size of the data array). If the upsampling rate is\n",

" smaller than 3, or the image is downsampled, then 'hanning'\n",

" interpolation is used to act as an anti-aliasing filter, unless the\n",

" image happens to be upsampled by exactly a factor of two or one.\n",

"\n",

" See\n",

" :doc:`/gallery/images\_contours\_and\_fields/interpolation\_methods`\n",

" for an overview of the supported interpolation methods, and\n",

" :doc:`/gallery/images\_contours\_and\_fields/image\_antialiasing` for\n",

" a discussion of image antialiasing.\n",

"\n",

" Some interpolation methods require an additional radius parameter,\n",

" which can be set by \*filterrad\*. Additionally, the antigrain image\n",

" resize filter is controlled by the parameter \*filternorm\*.\n",

"\n",

"interpolation\_stage : {'auto', 'data', 'rgba'}, default: 'auto'\n",

" Supported values:\n",

"\n",

" - 'data': Interpolation is carried out on the data provided by the user\n",

" This is useful if interpolating between pixels during upsampling.\n",

" - 'rgba': The interpolation is carried out in RGBA-space after the\n",

" color-mapping has been applied. This is useful if downsampling and\n",

" combining pixels visually.\n",

" - 'auto': Select a suitable interpolation stage automatically. This uses\n",

" 'rgba' when downsampling, or upsampling at a rate less than 3, and\n",

" 'data' when upsampling at a higher rate.\n",

"\n",

" See :doc:`/gallery/images\_contours\_and\_fields/image\_antialiasing` for\n",

" a discussion of image antialiasing.\n",

"\n",

"alpha : float or array-like, optional\n",

" The alpha blending value, between 0 (transparent) and 1 (opaque).\n",

" If \*alpha\* is an array, the alpha blending values are applied pixel\n",

" by pixel, and \*alpha\* must have the same shape as \*X\*.\n",

"\n",

"origin : {'upper', 'lower'}, default: :rc:`image.origin`\n",

" Place the [0, 0] index of the array in the upper left or lower\n",

" left corner of the Axes. The convention (the default) 'upper' is\n",

" typically used for matrices and images.\n",

"\n",

" Note that the vertical axis points upward for 'lower'\n",

" but downward for 'upper'.\n",

"\n",

" See the :ref:`imshow\_extent` tutorial for\n",

" examples and a more detailed description.\n",

"\n",

"extent : floats (left, right, bottom, top), optional\n",

" The bounding box in data coordinates that the image will fill.\n",

" These values may be unitful and match the units of the Axes.\n",

" The image is stretched individually along x and y to fill the box.\n",

"\n",

" The default extent is determined by the following conditions.\n",

" Pixels have unit size in data coordinates. Their centers are on\n",

" integer coordinates, and their center coordinates range from 0 to\n",

" columns-1 horizontally and from 0 to rows-1 vertically.\n",

"\n",

" Note that the direction of the vertical axis and thus the default\n",

" values for top and bottom depend on \*origin\*:\n",

"\n",

" - For ``origin == 'upper'`` the default is\n",

" ``(-0.5, numcols-0.5, numrows-0.5, -0.5)``.\n",

" - For ``origin == 'lower'`` the default is\n",

" ``(-0.5, numcols-0.5, -0.5, numrows-0.5)``.\n",

"\n",

" See the :ref:`imshow\_extent` tutorial for\n",

" examples and a more detailed description.\n",

"\n",

"filternorm : bool, default: True\n",

" A parameter for the antigrain image resize filter (see the\n",

" antigrain documentation). If \*filternorm\* is set, the filter\n",

" normalizes integer values and corrects the rounding errors. It\n",

" doesn't do anything with the source floating point values, it\n",

" corrects only integers according to the rule of 1.0 which means\n",

" that any sum of pixel weights must be equal to 1.0. So, the\n",

" filter function must produce a graph of the proper shape.\n",

"\n",

"filterrad : float > 0, default: 4.0\n",

" The filter radius for filters that have a radius parameter, i.e.\n",

" when interpolation is one of: 'sinc', 'lanczos' or 'blackman'.\n",

"\n",

"resample : bool, default: :rc:`image.resample`\n",

" When \*True\*, use a full resampling method. When \*False\*, only\n",

" resample when the output image is larger than the input image.\n",

"\n",

"url : str, optional\n",

" Set the url of the created `.AxesImage`. See `.Artist.set\_url`.\n",

"\n",

"Returns\n",

"-------\n",

"`~matplotlib.image.AxesImage`\n",

"\n",

"Other Parameters\n",

"----------------\n",

"data : indexable object, optional\n",

" If given, all parameters also accept a string ``s``, which is\n",

" interpreted as ``data[s]`` if ``s`` is a key in ``data``.\n",

"\n",

"\*\*kwargs : `~matplotlib.artist.Artist` properties\n",

" These parameters are passed on to the constructor of the\n",

" `.AxesImage` artist.\n",

"\n",

"See Also\n",

"--------\n",

"matshow : Plot a matrix or an array as an image.\n",

"\n",

"Notes\n",

"-----\n",

"\n",

".. note::\n",

"\n",

" This is the :ref:`pyplot wrapper <pyplot\_interface>` for `.axes.Axes.imshow`.\n",

"\n",

"Unless \*extent\* is used, pixel centers will be located at integer\n",

"coordinates. In other words: the origin will coincide with the center\n",

"of pixel (0, 0).\n",

"\n",

"There are two common representations for RGB images with an alpha\n",

"channel:\n",

"\n",

"- Straight (unassociated) alpha: R, G, and B channels represent the\n",

" color of the pixel, disregarding its opacity.\n",

"- Premultiplied (associated) alpha: R, G, and B channels represent\n",

" the color of the pixel, adjusted for its opacity by multiplication.\n",

"\n",

"`~matplotlib.pyplot.imshow` expects RGB images adopting the straight\n",

"(unassociated) alpha representation.\n",

"\u001b[31mSource:\u001b[39m \n",

"@\_copy\_docstring\_and\_deprecators(Axes.imshow)\n",

"\u001b[38;5;28;01mdef\u001b[39;00m imshow(\n",

" X: ArrayLike | PIL.Image.Image,\n",

" cmap: str | Colormap | \u001b[38;5;28;01mNone\u001b[39;00m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" norm: str | Normalize | \u001b[38;5;28;01mNone\u001b[39;00m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" \*,\n",

" aspect: Literal[\u001b[33m\"equal\"\u001b[39m, \u001b[33m\"auto\"\u001b[39m] | float | \u001b[38;5;28;01mNone\u001b[39;00m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" interpolation: str | \u001b[38;5;28;01mNone\u001b[39;00m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" alpha: float | ArrayLike | \u001b[38;5;28;01mNone\u001b[39;00m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" vmin: float | \u001b[38;5;28;01mNone\u001b[39;00m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" vmax: float | \u001b[38;5;28;01mNone\u001b[39;00m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" colorizer: Colorizer | \u001b[38;5;28;01mNone\u001b[39;00m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" origin: Literal[\u001b[33m\"upper\"\u001b[39m, \u001b[33m\"lower\"\u001b[39m] | \u001b[38;5;28;01mNone\u001b[39;00m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" extent: tuple[float, float, float, float] | \u001b[38;5;28;01mNone\u001b[39;00m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" interpolation\_stage: Literal[\u001b[33m\"data\"\u001b[39m, \u001b[33m\"rgba\"\u001b[39m, \u001b[33m\"auto\"\u001b[39m] | \u001b[38;5;28;01mNone\u001b[39;00m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" filternorm: bool = \u001b[38;5;28;01mTrue\u001b[39;00m,\n",

" filterrad: float = \u001b[32m4.0\u001b[39m,\n",

" resample: bool | \u001b[38;5;28;01mNone\u001b[39;00m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" url: str | \u001b[38;5;28;01mNone\u001b[39;00m = \u001b[38;5;28;01mNone\u001b[39;00m,\n",

" data=\u001b[38;5;28;01mNone\u001b[39;00m,\n",

" \*\*kwargs,\n",

") -> AxesImage:\n",

" \_\_ret = gca().imshow(\n",

" X,\n",

" cmap=cmap,\n",

" norm=norm,\n",

" aspect=aspect,\n",

" interpolation=interpolation,\n",

" alpha=alpha,\n",

" vmin=vmin,\n",

" vmax=vmax,\n",

" colorizer=colorizer,\n",

" origin=origin,\n",

" extent=extent,\n",

" interpolation\_stage=interpolation\_stage,\n",

" filternorm=filternorm,\n",

" filterrad=filterrad,\n",

" resample=resample,\n",

" url=url,\n",

" \*\*({\u001b[33m\"data\"\u001b[39m: data} \u001b[38;5;28;01mif\u001b[39;00m data \u001b[38;5;28;01mis\u001b[39;00m \u001b[38;5;28;01mnot\u001b[39;00m \u001b[38;5;28;01mNone\u001b[39;00m \u001b[38;5;28;01melse\u001b[39;00m {}),\n",

" \*\*kwargs,\n",

" )\n",

" sci(\_\_ret)\n",

" \u001b[38;5;28;01mreturn\u001b[39;00m \_\_ret\n",

"\u001b[31mFile:\u001b[39m c:\\users\\kkart\\appdata\\local\\programs\\python\\python313\\lib\\site-packages\\matplotlib\\pyplot.py\n",

"\u001b[31mType:\u001b[39m function"

]

},

"metadata": {},

"output\_type": "display\_data"

}

],

"source": [

"plt.imshow??"

]

},

{

"cell\_type": "code",

"execution\_count": 49,

"id": "97091e80-1607-4e05-b2ad-63ba4054e882",

"metadata": {},

"outputs": [],

"source": [

"from tensorflow.keras.models import Model\n",

"from tensorflow.keras.layers import Layer, Conv2D, Dense, MaxPooling2D, Input, Flatten\n",

"import tensorflow as tf"

]

},

{

"cell\_type": "code",

"execution\_count": 11,

"id": "65b93346-eed4-4c7b-a468-5d33263bb434",

"metadata": {},

"outputs": [],

"source": [

"gpus = tf.config.experimental.list\_physical\_devices('GPU')\n",

"for gpu in gpus:\n",

" tf.config.experimental.set\_memory\_growth(gpu, True)"

]

},

{

"cell\_type": "code",

"execution\_count": 12,

"id": "0c4728df-8f3a-4a11-b434-36e1614d7dba",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"0"

]

},

"execution\_count": 12,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"len(gpus)"

]

},

{

"cell\_type": "code",

"execution\_count": 13,

"id": "396c5c0d-ae2f-4d79-8937-927491cd4ead",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"[]"

]

},

"execution\_count": 13,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"gpus"

]

},

{

"cell\_type": "code",

"execution\_count": 16,

"id": "9dadb7dc-72cb-4a52-8be8-1c44d3f03487",

"metadata": {},

"outputs": [],

"source": [

"for gpu in gpus:\n",

" print(gpu)"

]

},

{

"cell\_type": "code",

"execution\_count": 17,

"id": "36bbd212-b9f7-4cc8-a3c4-6065c570986f",

"metadata": {},

"outputs": [],

"source": [

"POS\_PATH = os.path.join('data', 'positive')\n",

"NEG\_PATH = os.path.join('data', 'negative')\n",

"ANC\_PATH = os.path.join('data', 'anchor')"

]

},

{

"cell\_type": "code",

"execution\_count": 18,

"id": "dafa2c9f-c20c-4ade-993d-b5c44ef813bb",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"'data\\\\positive'"

]

},

"execution\_count": 18,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"POS\_PATH"

]

},

{

"cell\_type": "code",

"execution\_count": 21,

"id": "4fdfd0a0-7180-4099-b6eb-05cfb751785b",

"metadata": {},

"outputs": [],

"source": [

"os.makedirs(POS\_PATH, exist\_ok=True)\n",

"os.makedirs(NEG\_PATH, exist\_ok=True)\n",

"os.makedirs(ANC\_PATH, exist\_ok=True)"

]

},

{

"cell\_type": "code",

"execution\_count": 22,

"id": "3a06d72f-0e93-4fb6-8a1f-9a75c9a526a8",

"metadata": {},

"outputs": [],

"source": [

"# http://vis-www.cs.umass.edu/lfw"

]

},

{

"cell\_type": "code",

"execution\_count": 25,

"id": "5cb87f04-8819-4998-987a-9b724544d96b",

"metadata": {},

"outputs": [],

"source": [

"!tar -xf lfw.tgz"

]

},

{

"cell\_type": "code",

"execution\_count": 38,

"id": "a9fd9941-f4f8-4fc9-a2c2-e963de79b432",

"metadata": {},

"outputs": [],

"source": [

"import shutil"

]

},

{

"cell\_type": "code",

"execution\_count": 57,

"id": "fd57b40e-0446-4e58-a2de-f78d27c237ed",

"metadata": {},

"outputs": [],

"source": [

"#for directory in os.listdir('lfw'):\n",

" # for file in os.listdir(os.path.join('lfw', directory)): # Already moved the pictures from lfw to negative folder so this function will not work anymore!!!\n",

" # EX\_PATH = os.path.join('lfw', directory, file)\n",

" # NEW\_PATH = os.path.join(NEG\_PATH, file)\n",

" # os.replace(EX\_PATH, NEW\_PATH)"

]

},

{

"cell\_type": "code",

"execution\_count": 58,

"id": "410fe56c-2d40-4a78-951f-51132fca5cb0",

"metadata": {},

"outputs": [],

"source": [

"#os.listdir('lfw')"

]

},

{

"cell\_type": "code",

"execution\_count": 56,

"id": "a415c07c-034f-4fca-9782-fcb4e9d7cc7c",

"metadata": {},

"outputs": [],

"source": [

"#for directory in os.listdir('lfw'):\n",

" # for file in os.listdir(os.path.join('lfw', directory)): # Already moved the files, this function will not work anymore because it depends on the path\n",

" # print (os.path.join(NEG\_PATH, file))"

]

},

{

"cell\_type": "code",

"execution\_count": 111,

"id": "a0376d0e-f777-4f3c-a212-9b9be37a5821",

"metadata": {},

"outputs": [],

"source": [

"cap = cv2.VideoCapture(0)\n",

"while cap.isOpened():\n",

" ret, frame = cap.read()\n",

"\n",

" cv2.imshow('Image Collection', frame)\n",

"\n",

" if cv2.waitKey(1) & 0XFF == ord('q'):\n",

" break\n",

"cap.release()\n",

"cv2.destroyAllWindows()"

]

},

{

"cell\_type": "code",

"execution\_count": 45,

"id": "a5bfb50f-8dcd-4dce-b070-5fea58434ee8",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"\u001b[31mDocstring:\u001b[39m\n",

"waitKey([, delay]) -> retval\n",

". @brief Waits for a pressed key.\n",

". \n",

". The function waitKey waits for a key event infinitely (when \\f$\\texttt{delay}\\leq 0\\f$ ) or for delay\n",

". milliseconds, when it is positive. Since the OS has a minimum time between switching threads, the\n",

". function will not wait exactly delay ms, it will wait at least delay ms, depending on what else is\n",

". running on your computer at that time. It returns the code of the pressed key or -1 if no key was\n",

". pressed before the specified time had elapsed. To check for a key press but not wait for it, use\n",

". #pollKey.\n",

". \n",

". @note The functions #waitKey and #pollKey are the only methods in HighGUI that can fetch and handle\n",

". GUI events, so one of them needs to be called periodically for normal event processing unless\n",

". HighGUI is used within an environment that takes care of event processing.\n",

". \n",

". @note The function only works if there is at least one HighGUI window created and the window is\n",

". active. If there are several HighGUI windows, any of them can be active.\n",

". \n",

". @param delay Delay in milliseconds. 0 is the special value that means \"forever\".\n",

"\u001b[31mType:\u001b[39m builtin\_function\_or\_method"

]

},

"metadata": {},

"output\_type": "display\_data"

}

],

"source": [

"cv2.waitKey??"

]

},

{

"cell\_type": "code",

"execution\_count": 47,

"id": "2dd1d0c5-4b73-413a-b9a8-a39a2636f079",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"array([[[150, 172, 172],\n",

" [149, 171, 172],\n",

" [147, 170, 172],\n",

" ...,\n",

" [ 96, 88, 91],\n",

" [127, 118, 123],\n",

" [135, 126, 131]],\n",

"\n",

" [[149, 171, 171],\n",

" [148, 170, 171],\n",

" [147, 170, 172],\n",

" ...,\n",

" [129, 123, 126],\n",

" [146, 140, 144],\n",

" [150, 144, 148]],\n",

"\n",

" [[147, 169, 169],\n",

" [147, 169, 170],\n",

" [148, 170, 172],\n",

" ...,\n",

" [143, 143, 145],\n",

" [151, 150, 153],\n",

" [154, 153, 156]],\n",

"\n",

" ...,\n",

"\n",

" [[132, 119, 92],\n",

" [131, 118, 91],\n",

" [128, 116, 89],\n",

" ...,\n",

" [188, 198, 200],\n",

" [189, 200, 201],\n",

" [191, 202, 203]],\n",

"\n",

" [[135, 118, 93],\n",

" [132, 117, 92],\n",

" [130, 117, 90],\n",

" ...,\n",

" [177, 187, 189],\n",

" [176, 188, 189],\n",

" [177, 189, 190]],\n",

"\n",

" [[134, 116, 92],\n",

" [132, 116, 91],\n",

" [131, 117, 90],\n",

" ...,\n",

" [170, 180, 182],\n",

" [169, 181, 182],\n",

" [169, 181, 182]]], shape=(480, 640, 3), dtype=uint8)"

]

},

"execution\_count": 47,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"frame"

]

},

{

"cell\_type": "code",

"execution\_count": 50,

"id": "05fb6388-6ca3-4083-b086-44537b76e014",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<matplotlib.image.AxesImage at 0x1ef7b25c050>"

]

},

"execution\_count": 50,

"metadata": {},

"output\_type": "execute\_result"

},

{

"data": {

"image/png": "",

"text/plain": [

"<Figure size 640x480 with 1 Axes>"

]

},

"metadata": {},

"output\_type": "display\_data"

}

],

"source": [

"plt.imshow(frame)"

]

},

{

"cell\_type": "code",

"execution\_count": 51,

"id": "b5a513f1-fbe3-486b-80d8-ad16474ec5fa",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"(480, 640, 3)"

]

},

"execution\_count": 51,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"frame.shape"

]

},

{

"cell\_type": "code",

"execution\_count": 52,

"id": "31f33e4f-7328-40f2-9db7-22dbfc59415a",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"(250, 250, 3)"

]

},

"execution\_count": 52,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"frame[:250,:250, :].shape"

]

},

{

"cell\_type": "code",

"execution\_count": 53,

"id": "7fb08cfb-e1a7-4385-95de-3c16c56d3bd9",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<matplotlib.image.AxesImage at 0x1ef0217ead0>"

]

},

"execution\_count": 53,

"metadata": {},

"output\_type": "execute\_result"

},

{

"data": {

"image/png": "",

"text/plain": [

"<Figure size 640x480 with 1 Axes>"

]

},

"metadata": {},

"output\_type": "display\_data"

}

],

"source": [

"plt.imshow(frame[:250,:250, :])"

]

},

{

"cell\_type": "code",

"execution\_count": 59,

"id": "2ed1c394-8269-4d52-acf5-227f2de13d3b",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<matplotlib.image.AxesImage at 0x1ef021d34d0>"

]

},

"execution\_count": 59,

"metadata": {},

"output\_type": "execute\_result"

},

{

"data": {

"image/png": "",

"text/plain": [

"<Figure size 640x480 with 1 Axes>"

]

},

"metadata": {},

"output\_type": "display\_data"

}

],

"source": [

"plt.imshow(frame[120:120+250, 200:200+250, :])"

]

},

{

"cell\_type": "code",

"execution\_count": 60,

"id": "a09db0f5-ea39-4858-9e26-79f633df4a13",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<matplotlib.image.AxesImage at 0x1ef02278410>"

]

},

"execution\_count": 60,

"metadata": {},

"output\_type": "execute\_result"

},

{

"data": {

"image/png": "",

"text/plain": [

"<Figure size 640x480 with 1 Axes>"

]

},

"metadata": {},

"output\_type": "display\_data"

}

],

"source": [

"plt.imshow(frame[120:120+250, 200:200+250, 0])"

]

},

{

"cell\_type": "code",

"execution\_count": 61,

"id": "6752be9c-8f49-41f1-b594-3088243c9ab4",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<matplotlib.image.AxesImage at 0x1ef022b5310>"

]

},

"execution\_count": 61,

"metadata": {},

"output\_type": "execute\_result"

},

{

"data": {

"image/png": "",

"text/plain": [

"<Figure size 640x480 with 1 Axes>"

]

},

"metadata": {},

"output\_type": "display\_data"

}

],

"source": [

"plt.imshow(frame[120:120+250, 200:200+250, 2])"

]

},

{

"cell\_type": "code",

"execution\_count": 63,

"id": "5231a9f1-4e77-4c27-86ff-5bdfea311826",

"metadata": {},

"outputs": [],

"source": [

"#creating unique images\n",

"import uuid"

]

},

{

"cell\_type": "code",

"execution\_count": 66,

"id": "cb51f3e3-30d5-4c60-9f58-130f6c8dab77",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"'data\\\\anchor\\\\aa4ddad1-8a1a-11f0-9993-e59bf518a7e5.jpg'"

]

},

"execution\_count": 66,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"os.path.join(ANC\_PATH, '{}.jpg'.format(uuid.uuid1()))"

]

},

{

"cell\_type": "code",

"execution\_count": 110,

"id": "8471584c-3e9a-40db-8ea8-4293e8d62c1c",

"metadata": {},

"outputs": [],

"source": [

"cap = cv2.VideoCapture(0)\n",

"while cap.isOpened():\n",

" ret, frame = cap.read()\n",

"\n",

" #cut down frame to 250x250px\n",

" frame = frame[120:120+250, 200:200+250, :]\n",

"\n",

" #collect anchors\n",

" if cv2.waitKey(1) & 0XFF == ord('a'):\n",

" imgname = os.path.join(ANC\_PATH, '{}.jpg'.format(uuid.uuid1()))\n",

" cv2.imwrite(imgname, frame)\n",

"\n",

" #collect positives\n",

" if cv2.waitKey(1) & 0XFF == ord('p'):\n",

" imgname = os.path.join(POS\_PATH, '{}.jpg'.format(uuid.uuid1()))\n",

" cv2.imwrite(imgname, frame)\n",

"\n",

" #show image back to screen\n",

" cv2.imshow('Image Collection', frame)\n",

"\n",

" if cv2.waitKey(1) & 0XFF == ord('q'):\n",

" break\n",

"cap.release()\n",

"cv2.destroyAllWindows()"

]

},

{

"cell\_type": "code",

"execution\_count": 75,

"id": "6143c693-ab13-487c-bc55-806e9e95f472",

"metadata": {},

"outputs": [],

"source": [

"anchor = tf.data.Dataset.list\_files(ANC\_PATH + '/\*.jpg').take(300)\n",

"positive = tf.data.Dataset.list\_files(POS\_PATH + '/\*.jpg').take(300)\n",

"negative = tf.data.Dataset.list\_files(NEG\_PATH + '/\*.jpg').take(300)"

]

},

{

"cell\_type": "code",

"execution\_count": 78,

"id": "324ef7bd-21c5-4a17-8f82-23ba8aa2d8d3",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"'data\\\\anchor/\*.jpg'"

]

},

"execution\_count": 78,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"ANC\_PATH + '/\*.jpg'"

]

},

{

"cell\_type": "code",

"execution\_count": 76,

"id": "2a588885-bea5-4b37-85ca-c4c79cfd5a60",

"metadata": {},

"outputs": [],

"source": [

"dir\_test = anchor.as\_numpy\_iterator()"

]

},

{

"cell\_type": "code",

"execution\_count": 77,

"id": "6c899b04-f4af-4043-8c73-ebb96aaa4a0d",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"b'data\\\\anchor\\\\b9d5dae4-8a1b-11f0-abe5-e59bf518a7e5.jpg'"

]

},

"execution\_count": 77,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"dir\_test.next()"

]

},

{

"cell\_type": "code",

"execution\_count": 79,

"id": "8b12ac11-513e-442b-9027-50f437891325",

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"b'data\\\\anchor\\\\b8d1ed4a-8a1b-11f0-8f58-e59bf518a7e5.jpg'\n"

]

}

],

"source": [

"print(dir\_test.next())"

]

},

{

"cell\_type": "code",

"execution\_count": 85,

"id": "2c13de74-8e33-40d2-ac93-17879fa9841e",

"metadata": {},

"outputs": [],

"source": [

"def preprocess(file\_path):\n",

" \n",

" byte\_img = tf.io.read\_file(file\_path)\n",

" img = tf.io.decode\_jpeg(byte\_img)\n",

" img = tf.image.resize(img, (100,100))\n",

" img = img / 255.0\n",

" return img"

]

},

{

"cell\_type": "code",

"execution\_count": 82,

"id": "9ee05e45-bd8b-4fdb-a9ab-f279a6695f5c",

"metadata": {},

"outputs": [],

"source": [

"img = preprocess('data\\\\anchor\\\\b8d1ed4a-8a1b-11f0-8f58-e59bf518a7e5.jpg')"

]

},

{

"cell\_type": "code",

"execution\_count": 83,

"id": "19f8756e-1dd1-40ae-954b-c7221d859c31",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"np.float32(0.08848039)"

]

},

"execution\_count": 83,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"img.numpy().min()"

]

},

{

"cell\_type": "code",

"execution\_count": 84,

"id": "7fca8df4-4e08-4568-a951-d66c8df87e88",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<matplotlib.image.AxesImage at 0x1ef023d5450>"

]

},

"execution\_count": 84,

"metadata": {},

"output\_type": "execute\_result"

},

{

"data": {

"image/png": "",

"text/plain": [

"<Figure size 640x480 with 1 Axes>"

]

},

"metadata": {},

"output\_type": "display\_data"

}

],

"source": [

"plt.imshow(img)"

]

},

{

"cell\_type": "code",

"execution\_count": 94,

"id": "6c1cd1ed-eafe-4b35-b112-405eb1810ab0",

"metadata": {},

"outputs": [],

"source": [

"dataset = anchor.concatenate(positive).concatenate(negative)\n",

"dataset = dataset.map(preprocess)"

]

},

{

"cell\_type": "code",

"execution\_count": 88,

"id": "acc143ad-a482-4d1b-8fa4-3031c1a8a431",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<tf.Tensor: shape=(), dtype=int32, numpy=1>"

]

},

"execution\_count": 88,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"tf.ones\_like(1)"

]

},

{

"cell\_type": "code",

"execution\_count": 89,

"id": "26a64582-3382-49e0-a4f5-92ffd67c437d",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<tf.Tensor: shape=(5,), dtype=float32, numpy=array([1., 1., 1., 1., 1.], dtype=float32)>"

]

},

"execution\_count": 89,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"tf.ones\_like([1,1,1,4.35235,2352.4])"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "27260115-6eb0-4334-b8c5-3659bacb227d",

"metadata": {},

"outputs": [],

"source": [

"# (anchor, positive) => 1,1,1,1,1\n",

"# (anchor, negative) => 0,0,0,0,0"

]

},

{

"cell\_type": "code",

"execution\_count": 97,

"id": "fb679572-bf6a-453c-a933-8e2e1f0def23",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<tf.Tensor: shape=(300,), dtype=float32, numpy=\n",

"array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,\n",

" 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.], dtype=float32)>"

]

},

"execution\_count": 97,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"tf.ones(len(anchor))"

]

},

{

"cell\_type": "code",

"execution\_count": 98,

"id": "dc39f05c-8023-4f2f-896d-0006ee09bd97",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"<tf.Tensor: shape=(300,), dtype=float32, numpy=\n",

"array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,\n",

" 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)>"

]

},

"execution\_count": 98,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"tf.zeros(len(anchor))"

]

},

{

"cell\_type": "code",

"execution\_count": 103,

"id": "b84581cf-3a37-4575-bb85-07f9666bdc0b",

"metadata": {},

"outputs": [],

"source": [

"class\_labels = tf.data.Dataset.from\_tensor\_slices(tf.zeros(len(anchor)))"

]

},

{

"cell\_type": "code",

"execution\_count": 104,

"id": "129d9e7a-fa6e-44aa-8e6b-04cd1ae93df0",

"metadata": {},

"outputs": [],

"source": [

"iterator\_labs = class\_labels.as\_numpy\_iterator()"

]

},

{

"cell\_type": "code",

"execution\_count": 105,

"id": "b3ad2b61-b84a-49a3-a31b-25ca28ef48a2",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"np.float32(0.0)"

]

},

"execution\_count": 105,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"iterator\_labs.next()"

]

},

{

"cell\_type": "code",

"execution\_count": 96,

"id": "daf5d3bb-5ced-4195-9540-47b2cdf1fd1d",

"metadata": {},

"outputs": [],

"source": [

"positives = tf.data.Dataset.zip((anchor, positive, tf.data.Dataset.from\_tensor\_slices(tf.ones(len(anchor)))))\n",

"negatives = tf.data.Dataset.zip((anchor, negative, tf.data.Dataset.from\_tensor\_slices(tf.zeros(len(anchor)))))\n",

"data = positives.concatenate(negatives)"

]

},

{

"cell\_type": "code",

"execution\_count": 108,

"id": "52592280-1867-4b1e-af3d-2e58a7fa66a8",

"metadata": {},

"outputs": [],

"source": [

"samples = data.as\_numpy\_iterator()"

]

},

{

"cell\_type": "code",

"execution\_count": 109,

"id": "a4ffdd13-1cd2-4a14-b325-9cc9145a8cc0",

"metadata": {},

"outputs": [

{

"data": {

"text/plain": [

"(b'data\\\\anchor\\\\c676f9a9-8a1b-11f0-b068-e59bf518a7e5.jpg',\n",

" b'data\\\\positive\\\\1c6f98a7-8a1b-11f0-8e88-e59bf518a7e5.jpg',\n",

" np.float32(1.0))"

]

},

"execution\_count": 109,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"samples.next()"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "43019ea9-4790-456f-a8fc-bb89d8f32762",

"metadata": {},

"outputs": [],

"source": []

}

],

"metadata": {

"kernelspec": {

"display\_name": "Python 3 (ipykernel)",

"language": "python",

"name": "python3"

},

"language\_info": {

"codemirror\_mode": {

"name": "ipython",

"version": 3

},

"file\_extension": ".py",

"mimetype": "text/x-python",

"name": "python",

"nbconvert\_exporter": "python",

"pygments\_lexer": "ipython3",

"version": "3.13.7"

}

},

"nbformat": 4,

"nbformat\_minor": 5

}