dmtools

Release 0.0.1rc1

Henry Robbins

Sep 27, 2021

CONTENTS

dmtools (Digital Media Tools) is a Python package providing low-level tools for working with digital media programmatically. The netpbm module allows one to read and create Netpbm images. Color space transformations can be done with the colorspace module. Using ffmpeg, the animation module can export .mp4 videos formed from a list of images and the sound module can be used to add sound to these videos as well. Lastly, ASCII art can be produced with the ascii module.

For those experienced with installing and using Python packages, you can find brief installation instructions in the README. The installation instructions found here are aimed at beginner users. First, we will install a programming language called Python. Next, we will install dmtools, a Python package. The last section is optional and a little more intensive. It walks through the installation of a program called FFmpeg which is required if you wish to create videos with dmtools.

CHAPTER

ONE

INSTALLING PYTHON

In order to use dmtools, you will need to install the Python programming language. We preface the Python installation instructions with a breif Q&A. This section is ordered so that each answer naturally leads into the following question so it is best read in order.

1.1 Q&A

"What is Programming Language?"

The purpose of a programming language is to allow us to give instructions to a computer. At first, this may seem foreign. However, every time you interact with a computer, you are giving it instructions to do certain tasks like which website to navigate to, what document to open, etc.. The difference is in the way you are communicating that information. You are most likely familiar with Graphical User Interfaces (GUIs). These are programs which provide graphical ways of giving the computer instructions using the keyboard and mouse to point and click.

"How does a programming lanaguage let us give instructions to a computer?"

Without getting into too much detail, programming langauges are just like human languages. They have syntax which defines the structure of the language and they have semantics which define the meaning of certain structures in the language. Following these rules, we can write up a set of instructions and it off to the computer to execute.

"This sounds complicated. Why would I use this instead of a program with a nice GUI?"

There are two main reasons: humans are lazy and flexibility. Often times, there are tasks on the computer that are extremely repetitive. Unlike GUIs, programming languages don't require the human to be very involved. We only need to give the instructions once and the computer will go on chugging away until we tell it to stop. In terms of flexibility, it may seem that programs like Photoshop and Google Docs have an endless number of tabs, knobs, and dials but their flexibility pales in comparison to programming languages. With a programming language, the limit is quite literally, your imagination.

"What is Python?"

Yes, Python is a programming language. But, there are many different ways to classify programming languages. There are many characteristics of Python but the one we wish to emphasize here is that it is a general-purpose scripting language. Scripting lanaguages "automate the exectution of tasks that would otherwise be performed indiviually by a human operator." It is simple in that files written in the language can be run as scripts where the computer just goes through the file linearly, executing each task as it is given.

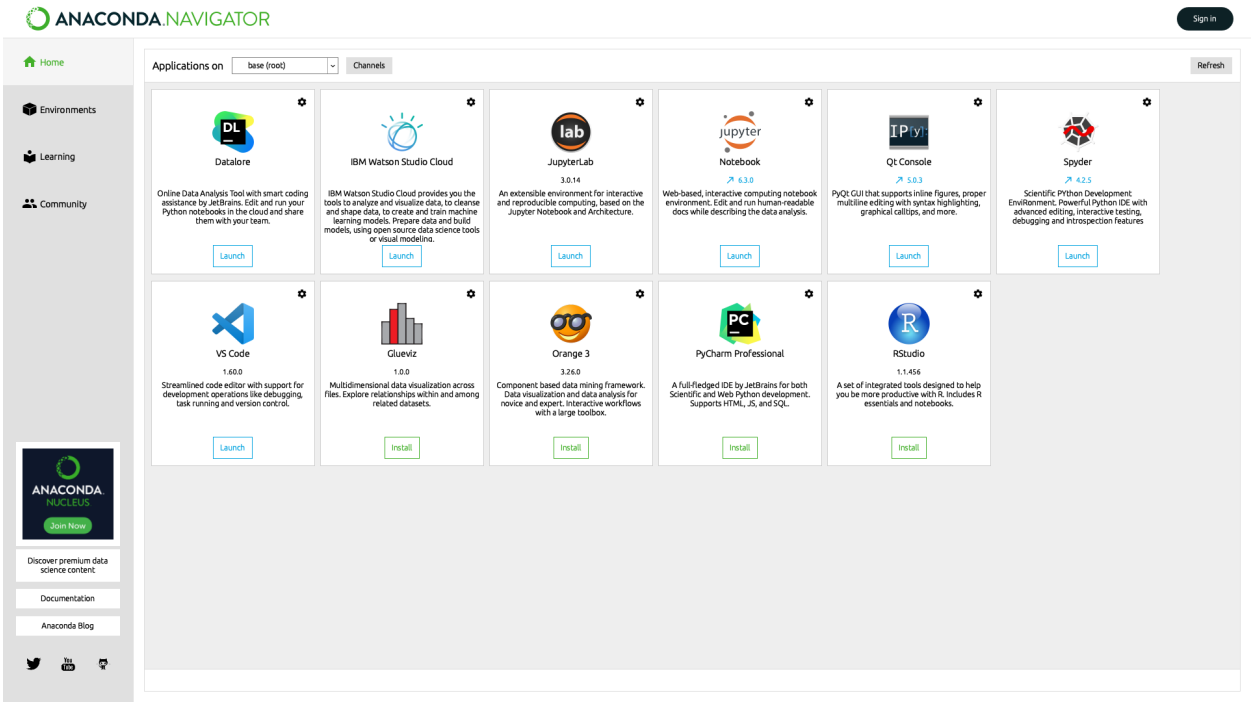
1.2 Anaconda

To install Python, we will use Anaconda which provides an extremely popular Python distribution called Anaconda Individual Edition. Navigate to the link and scroll to the bottom to select the Anaconda Installer for your operating system. Choose the Graphical Installer.

To verify you now have Python, open up a terminal (the Terminal Application on macOS) and run `python` to open up a Python prompt (a place where Python instructions can be run). The line beginning with >>> is where you can type Python code and run it. Type `print("Hello World!")` and hit Enter. It should display `Hello World!` as the result of the command! You can then type `quit()` or CTRL+D to exit the prompt.

```
python
Python 3.8.8 (default, Apr 13 2021, 12:59:45)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World!")
Hello World!
>>> quit()
```

You now have Python installed on your computer! Terminal is not a very friendly place to learn to write code. For this reason, it is recommended you install Juypter Notebook at this point. See the *Using Jupyter Notebooks* tutorial for more information. To install, navigate to the Home tab of the Anaconda Navigator application and click install under Jupyter Notebook (Not JupyterLab).



After installing, the “Install” button should become a “Launch” button.

CHAPTER

TWO

INSTALLING DMTOOLS

In this section, we will install the dmtools Python package. But first, what is a Python package? A Python package is essentially pre-bundled Python code that provides some functionality. For example, NumPy is a Python package (one you will get more familiar with in *Working with Images in NumPy*) that allows for easy manipulation of arrays. Python packages are your friend! They allow you to easily use other people's code so you never have to re-invent the wheel and can spend more time being creative.

In installing anaconda, you should now have a program called pip which stands for Pip Installs Packages. It is a Python package manager and it is the tool we will use to install dmtools. Just run the following line.

```
pip install dmtools
```

To the verify the installation worked correctly, open a Python prompt by typing `python` and then type `from dmtools import netpbm`. If you don't get any error messages, the instllation was a success!

```
python
Python 3.8.8 (default, Apr 13 2021, 12:59:45)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from dmtools import netpbm
>>> quit()
```

CHAPTER

THREE

INSTALLING FFMPEG (OPTIONAL)

1. This section is not optional of you wish to create videos with dmtools
2. Currently, these installation instructions focus on macOS users. For installation instructions on other operating systems, see Download FFmpeg.

In order to install FFmpeg, we will first need to install a package manager. A package manager functions similarly to an app store–it provides a way of installing and managing computer programs "in a consistent manner." Homebrew is a package manager for macOS. It is the one we will use to install FFmpeg. To install it, paste the following line in macOS Terminal.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/
↪install.sh)"
```

When running the above line, you will likley be prompted to install Command Line Tools (CLT) for Xcode. This can be installed with

```
xcode-select --install
```

To verify Homebrew was installed properly, run `brew` in Terminal and you should recieive a help page on various Homebrew commands. With Homebrew now installed, you can easily install FFmpeg with

```
brew install ffmpeg
```

This installation may take some time. Once complete, verify it was installed properly by running `ffmpeg` in Terminal. It should return some FFmpeg version information.

Congratulations! You have now installed a package manager and FFmpeg. You will now be able to create videos using dmtools.

CHAPTER

FOUR

TUTORIALS

This section includes a few tutorials to get you up and running and using dmtools effectivley. The first tutorial is an introduction to Jupyter Notebooks which are a tool for writing and exectuting Python code. It is highly recommended that you follow this tutorial before proceeding to the Python tutorial as the Python introduction will utilize Jupyter Notebooks and Following along is the best way to learn. Similarly, the introduction to NumPy will use Jupyter Notebooks.

4.1 Using Jupyter Notebooks

This tutorial will walk through a short introduction to Jupyter Notebooks with emphasis on basics needed to follow along to the following tutorials.

4.2 Introduction to Python

This tutorial will walk through a short introduction to Python with emphasis on the neccessary basics for using dmtools and working with images.

4.3 Working with Images in NumPy

This tutorial will walk through a short introduction to NumPy with emphasis on the tools that can be used for working with images.

CHAPTER

FIVE

DOCUMENTATION

5.1 dmtools package

5.1.1 dmtools.io module

dmtools.io.**read_png**(*path: str*) → numpy.ndarray
Read a png file into a NumPy array.

Parameters **path** (*str*) – String file path.

Returns NumPy array representing the image.

Return type np.ndarray

dmtools.io.**write_png**(*image: numpy.ndarray*, *path: str*)
Write NumPy array to a png file.

The NumPy array should have integer values in the range [0, 255]. Otherwise, this function has undefined behavior.

Parameters

- **image** (*np.ndarray*) – NumPy array representing image.
- **path** (*str*) – String file path.

5.1.2 dmtools.netpbm module

class dmtools.netpbm.**Netpbm**(*P: int*, *k: int*, *M: numpy.ndarray*)
Bases: object

An object representing a Netpbm image.

Netpbm is a package of graphics programs and a programming library. These programs work with a set of graphics formats called the "netpbm" formats. Each format is identified by a "magic number" which is denoted as P followed by the number identifier. This class works with the following formats.

- pbm: Pixels are black or white (P1 and P4).
- pgm: Pixels are shades of gray (P2 and P5).
- ppm: Pixels are in full color (P3 and P6).

Each of the formats has two "magic numbers" associated with it. The lower number corresponds to the ASCII (plain) format while the higher number corresponds to the binary (raw) format. This class can handle reading both the plain and raw formats though it can only export Netpbm images in the plain formats (P1, P2, and P3).

The plain formats for all three of pbm, pgm, and ppm are quite similar. Here is an example pgm format.

```
P2
5 3
4
1 1 0 1 0
2 0 3 0 1
2 2 3 1 0
```

The first row of the file contains the "magic number". In this example, the file is a grayscale pgm image. The second row gives the file dimensions (width by height) separated by whitespace. The third row gives the maximum gray/color value. In this case, it is the maximum gray value since this is a grayscale pgm image. Essentially, this number encodes how many different gradients there are in the image. Lastly, the remaining lines of the file encode the actual pixels of the image. In a pbm image, the third line is not needed since pixels have binary (black or white) values. In a ppm full-color image, each pixels has three values represeting it–the values of the red, green, and blue channels.

This descriptions serves as a brief overview of the Netpbm formats with the relevant knowledge for using this class. For more information about Netpbm, see the Netpbm Home Page.

extension_to_magic_number = {'pbm': 1, 'pgm': 2, 'ppm': 3}

magic_number_to_extension = {1: 'pbm', 2: 'pgm', 3: 'ppm'}

rescale(*k: int*)
Rescale the image by the desired scaling factor.

Uses Nearest-neighbor interpolation as the image scaling algorithm. Read more about image scaling algorithms at Image scaling.

Parameters **k** (*int*) – Scale factor

set_max_color_value(*k: int*)
Set the maximum gray/color value of this Netpbm image.

Parameters **k** (*int*) – Maximum gray/color value.

to_netpbm(*path: str*, *comment: List[str] = []*)
Write object to a Netpbm file (pbm, pgm, ppm).

Uses the ASCII (plain) magic numbers.

Parameters

- **path** (*str*) – String file path.
- **comment** (*str*) – List of comment lines to include in the file.

to_png(*path: str*)
Write object to a png file.

Parameters **path** (*str*) – String file path.

dmtools.netpbm.**read_netpbm**(*path: str*) → *dmtools.netpbm.Netpbm*
Read Netpbm file (pbm, pgm, ppm) into Netpbm.

Parameters **path** (*str*) – String file path.

Returns A Netpbm image

Return type *Netpbm*

5.1.3 dmtools.transform module

dmtools.transform.**box_resize_weighting_function**(*x: float*) → float
Box filter's weighting function.

For more information about the Box filter, see Box.

Parameters **x** (*float*) – distance to source pixel.

Returns weight on the source pixel.

Return type float

dmtools.transform.**catmull_rom_resize_weighting_function**(*x: float*) → float
Catmull-Rom filter's weighting function.

For more information about the Catmull-Rom filter, see Cubic Filters.

Parameters **x** (*float*) – distance to source pixel.

Returns weight on the source pixel.

Return type float

dmtools.transform.**rescale**(*image: numpy.ndarray*, *k: int*, *filter: str = 'point'*, *weighting_function: Optional[Callable] = None*, *support: Optional[Callable] = None*, *clip: bool = True*) → numpy.ndarray
Rescale the image by the given scaling factor.

Parameters

- **image** (*np.ndarray*) – Image to rescale.
- **k** (*int*) – Scaling factor.
- **filter** (*str*) – {point, box, triangle, catrom}. Defaults to point.
- **weighting_function** (*Callable*) – Weighting function to use.
- **support** (*float*) – Support of the provided weighting function.
- **clip** (*bool*) – Clip values into [0,255] if True. Defaults to true.

Returns Rescaled image.

Return type np.ndarray

dmtools.transform.**triangle_resize_weighting_function**(*x: float*) → float
Triangle filter's weighting function.

For more information about the Triangle filter, see Triangle.

Parameters **x** (*float*) – distance to source pixel.

Returns weight on the source pixel.

Return type float

5.1.4 dmtools.colorspace module

dmtools.colorspace.**Lab_to_RGB**(*image: numpy.ndarray*, *illuminant: str = 'D65'*) → numpy.ndarray
Convert an image in Lab space to CIE RGB space.

For details about the implemented conversion, see CIE 1931 color space and CIELAB color space.

Parameters

- **image** (*np.ndarray*) – Image in Lab space.
- **illuminant** (*str*) – Standard illuminant {D65, D50}

Returns Image in CIE RGB space.

Return type np.ndarray

dmtools.colorspace.**Lab_to_XYZ**(*image: numpy.ndarray*, *illuminant: str = 'D65'*) → numpy.ndarray
Convert an image in Lab space to CIE XYZ space.

For details about the implemented conversion, see CIELAB color space.

Parameters

- **image** (*np.ndarray*) – Image in Lab space.
- **illuminant** (*str*) – Standard illuminant {D65, D50}

Returns Image in CIE XYZ space.

Return type np.ndarray

dmtools.colorspace.**RGB_to_Lab**(*image: numpy.ndarray*, *illuminant: str = 'D65'*) → numpy.ndarray
Convert an image in CIE RGB space to Lab space.

For details about the implemented conversion, see CIE 1931 color space and CIELAB color space.

Parameters

- **image** (*np.ndarray*) – Image in CIE RGB space.
- **illuminant** (*str*) – Standard illuminant {D65, D50}

Returns Image in Lab space.

Return type np.ndarray

dmtools.colorspace.**RGB_to_XYZ**(*image: numpy.ndarray*) → numpy.ndarray
Convert an image in CIE RGB space to XYZ space.

For details about the implemented conversion, see CIE 1931 color space.

Parameters **image** (*np.ndarray*) – Image in CIE RGB space.

Returns Image in CIE XYZ space.

Return type np.ndarray

dmtools.colorspace.**RGB_to_YUV**(*image: numpy.ndarray*) → numpy.ndarray
Convert an image in CIE RGB space to YUV space.

For details about the implemented conversion, see YUV.

Parameters **image** (*np.ndarray*) – Image in CIE RGB space.

Returns Image in YUV space.

Return type np.ndarray

dmtools.colorspace.**RGB_to_gray**(*image: numpy.ndarray*) → numpy.ndarray
Convert an image in CIE RGB space to grayscale.

For details about the implemented conversion, see FAQs about Color.

Parameters **image** (*np.ndarray*) – Image in CIE RGB space.

Returns Image in grayscale.

Return type np.ndarray

dmtools.colorspace.**XYZ_to_Lab**(*image: numpy.ndarray*, *illuminant: str = 'D65'*) → numpy.ndarray
Convert an image in CIE XYZ space to Lab space.

For details about the implemented conversion, see CIELAB color space.

Parameters

- **image** (*np.ndarray*) – Image in CIE XYZ space.
- **illuminant** (*str*) – Standard illuminant {D65, D50}

Returns Image in Lab space.

Return type np.ndarray

dmtools.colorspace.**XYZ_to_RGB**(*image: numpy.ndarray*) → numpy.ndarray
Convert an image in CIE XYZ space to RGB space.

For details about the implemented conversion, see CIE 1931 color space.

Parameters **image** (*np.ndarray*) – Image in CIE XYZ space.

Returns Image in CIE RGB space.

Return type np.ndarray

dmtools.colorspace.**YUV_to_RGB**(*image: numpy.ndarray*) → numpy.ndarray
Convert an image in YUV space to CIE RGB space.

For details about the implemented conversion, see YUV.

Parameters **image** (*np.ndarray*) – Image in YUV space.

Returns Image in CIE RGB space.

Return type np.ndarray

dmtools.colorspace.**apply_to_channels**(*image: numpy.ndarray*, *f_1: Callable*, *f_2: Callable*, *f_3: Callable*) → numpy.ndarray
Return the image with the functions applied to each channel.

Parameters

- **image** (*np.ndarray*) – Image (recommended to be normalized).
- **f_1** (*Callable*) – Function to apply to the first channel.
- **f_2** (*Callable*) – Function to apply to the second channel.
- **f_3** (*Callable*) – Function to apply to the third channel.

Returns Pixel matrix with functions applied to each channel.

Return type np.ndarray

dmtools.colorspace.**denormalize**(*image: numpy.ndarray*, *color_space: str*) → numpy.ndarray
Denormalize the image in the given color space.

Parameters

- **image** (*np.ndarray*) – Normalized image in the given color space.
- **color_space** (*str*) – Color space {RGB, Lab, YUV}.

Returns Denormalized image in the given color space.

Return type np.ndarray

dmtools.colorspace.**gray_to_RGB**(*image: numpy.ndarray*) → numpy.ndarray
Convert an image in grayscale to CIE RGB space.

Parameters **image** (*np.ndarray*) – Image in grayscale.

Returns Image in CIE RGB space.

Return type np.ndarray

dmtools.colorspace.**normalize**(*image: numpy.ndarray*, *color_space: str*) → numpy.ndarray
Normalize the image in the given color space.

Parameters

- **image** (*np.ndarray*) – Image in the given color space.
- **color_space** (*str*) – Color space {RGB, Lab, YUV}.

Returns Normalized image with values in [0,1].

Return type np.ndarray

5.1.5 dmtools.animation module

dmtools.animation.**clip**(*path: str*, *start: int = 0*, *end: int = - 1*) → List[numpy.ndarray]
Return a list of images in the given directory.

Images are ordered according to their name. Hence, the following naming convention is recommend.

name0000.png, name0001.png, ...

Parameters

- **path** (*str*) – String directory path.
- **start** (*int, optional*) – Starting frame. Defaults to 0.
- **end** (*int, optional*) – Ending frame. Defaults to -1.

Returns List of NumPy arrays representing images.

Return type List[np.ndarray]

dmtools.animation.**to_mp4**(*frames: List[numpy.ndarray]*, *path: str*, *fps: int*, *s: int = 1*, *audio: Optional[*dmtools.sound.WAV*] = None*)
Write an animation as a .mp4 file using ffmpeg through imageio.mp4

Parameters

- **frames** (*List[np.ndarray]*) – List of frames in the animation.
- **audio** (*sound.WAV*) – Audio for the animation (None if no audio).
- **path** (*str*) – String file path.
- **fps** (*int*) – Frames per second.

- **s** (*int, optional*) – Multiplier for scaling. Defaults to 1.

5.1.6 dmtools.ascii module

class dmtools.ascii.**Ascii**(*M: numpy.ndarray*)
Bases: object

An object representing an ASCII image.

For more information about ASCII, see ASCII

to_png(*path: str*)
Write object to a png file.

Parameters **path** (*str*) – String file path.

to_txt(*path: str*)
Write object to a txt file.

Parameters **path** (*str*) – String file path.

dmtools.ascii.**netpbm_to_ascii**(*image:* dmtools.netpbm.Netpbm) → *dmtools.ascii.Ascii*
Return an ASCII representation of the given image.

This function uses a particular style of ASCII art in which "symbols with various intensities [are used for] creating gradients or contrasts."

Parameters **image** (netpbm.Netpbm) – Netpbm image.

Returns ASCII representation of image.

Return type *Ascii*

5.1.7 dmtools.sound module

class dmtools.sound.**WAV**(*r: numpy.ndarray*, *l: numpy.ndarray*, *sample_rate: int = 44100*)
Bases: object

An object representing a WAV audio file.

For more information about the audio file format, see WAV

to_wav(*path*)
Write object to a WAV audio file (wav)

Parameters **path** (*str*) – String file path.

dmtools.sound.**wave**(*f: float*, *a: float*, *t: float*) → numpy.ndarray
Generate the samples of a sound wave.

Parameters

- **f** (*float*) – Frequency of the sound wave.
- **a** (*float*) – Amplitude of the sound wave.
- **t** (*float*) – Duration (seconds) of the sound wave.

Returns NumPy array with sample points of wave.

Return type np.ndarray

dmtools.sound.**wave_sequence**(*frequencies: numpy.ndarray*, *t*) → *dmtools.sound.WAV*
Return a Wav sound which iterates through the given frequencies.

Parameters

- **frequencies** (*np.ndarray*) – frequencies to iterate through.
- **t** (*[type]*) – duration of iteration.

Returns Wav file.

Return type *WAV*

5.1.8 dmtools.arrange module

dmtools.arrange.**border**(*image: numpy.ndarray*, *b: int*, *color: int = 'white'*, *k: int = 255*) → numpy.ndarray
Add a border of width b to the image.

Parameters

- **image** (Netpbm) – Netpbm image to add a border to
- **b** (*int*) – width of the border/margin.
- **color** (*int*) – color of border {'white', 'black'} (defaults to white).
- **k** (*int*) – white point (defaults to 255).

Returns Image with border added.

Return type np.ndarray

dmtools.arrange.**image_grid**(*images: List[numpy.ndarray]*, *w: int*, *h: int*, *b: int*, *color: int = 'white'*, *k: int = 255*) → numpy.ndarray
Create a w * h grid of images with a border of width b.

Parameters

- **images** (*List[np.ndarray]*) – images (of same dimension) for grid.
- **w** (*int*) – number of images in each row of the grid.
- **h** (*int*) – number of images in each column of the grid.
- **b** (*int*) – width of the border/margin.
- **color** (*int*) – color of border {'white', 'black'} (defaults to white).
- **k** (*int*) – white point (defaults to 255).

Returns grid layout of the images.

Return type np.ndarray

PYTHON MODULE INDEX

d

INDEX

W

X

Y