Supplementary Materials: USC-DCT: A Collection of Diverse Classification Tasks

Adam M. Jones [1,†], Gozde Sahin [2,†], Zachary W. Murdock [1,†], Yunhao Ge [2], Ao Xu [2], Yuecheng Li [2], Di Wu [2], Shuo Ni [2], Po-Hsuan Huang [1], Kiran Lekkala [2] and Laurent Itti [1,2,*]

S.1. Dataset Inclusion Criteria Decision Tree

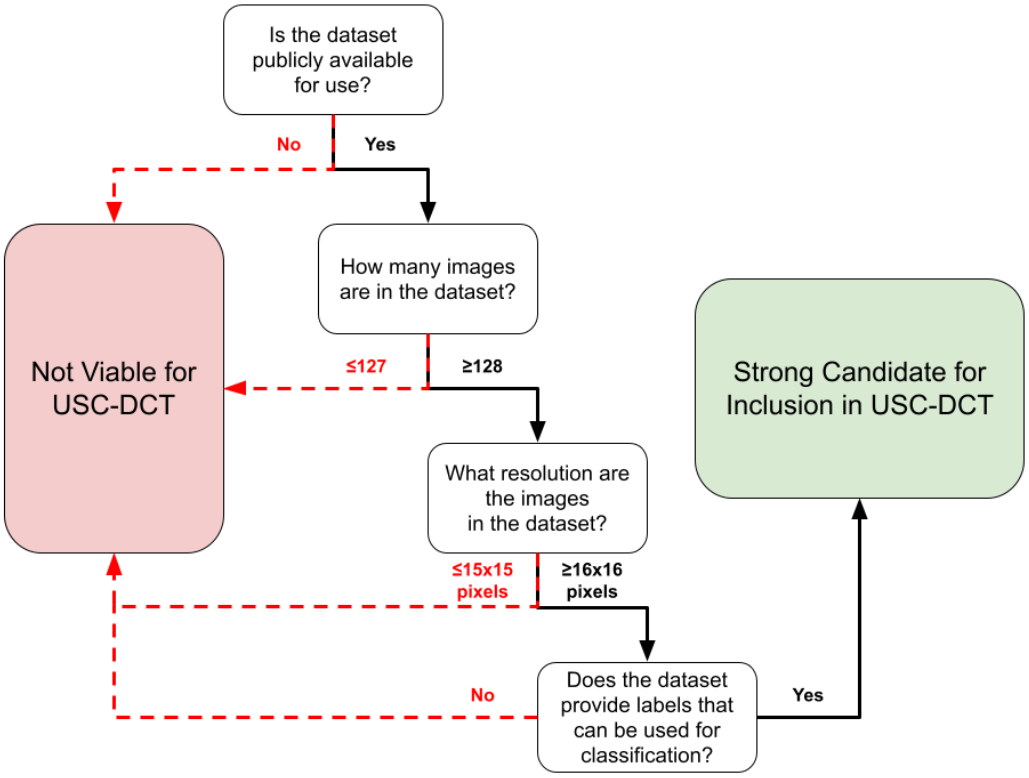


Figure S1. Visualization of the decision process used to determine if a computer vision dataset could be included in the development of USC-DCT, as mentioned in Section 3.1.1.

S.2. Dataset Preparation Examples

As mentioned above, a *prepare_dataset.py* script is created for each dataset during the inspection phase. Most of the work to "ingest" the dataset is done by a function called *parse_dataset()*. In this function, the extracted files and any other knowledge are combined to create a list of every image file with its details and a dictionary of all classes. Many of the datasets have straightforward parsing. However, some are quite complex (not including the splitting up of the super-datasets: iNaturalist and Office-Home).

S.2.1. Simple Example

One simple example of our task-specific scripts to integrate a task into USC-DCT is the Concrete Cracks dataset. Listing 1 includes the main processing function for this dataset, and shows a straightforward example of how a user would customize *parse_dataset()* to add a task to USC-DCT. The script first creates a list of all the image files found in the *original_files* folder where the downloaded archive for the dataset was extracted to. These images are processed using the method given in Section 3 to set status codes for each image (*problem_value* and the set (train/val/test). Images with non-zero status codes are excluded from the collection by setting their class and set values to -1.

```python
def parse_dataset(
    dataset_root_path: str,
) -> tuple[list[DatasetImage], dict[str, int]]:
    """This returns a list of all images within the dataset.
    This includes **all images** in both ./original_files/ and,
    if necessary, ./converted_files/
    """

    image_list: list[DatasetImage] = []
    class_dict: dict[str, int] = {}

    # get all file paths
    files_list = get_all_files_in_directory(
        dataset_root_path + "original_files/", dataset_root_path)

    (image_path_list, _) = filter_file_list(files_list)

    # create a class dict
    class_list = os.listdir(dataset_root_path + "original_files/")
    for k, class_name in enumerate(sorted(class_list, key=str.casefold)):
        class_dict[class_name] = k

    for image_path in image_path_list:
        problem_value, set_value = 0, 0

        # first address problem_value
        if not is_image_valid(dataset_root_path + image_path):
            problem_value = 1

        # check non-dataset file

        # once problem_value != 0, set all else as -1
        if problem_value != 0:
            class_int = -1
            set_value = -1
        else:
            # finally address the class_int
            class_from_path = image_path.split("/")[-2]
            class_int = class_dict[class_from_path]

        image_list.append(
            DatasetImage(
                relative_path=image_path,
                class_id = class_int ,
                set_id = set_value ,
                problem=problem_value,
            )
        )

    return (image_list, class_dict)
```

Code Snippet. 1: A simple example of a prepare_dataset function.

In addition to the sample *parse_dataset()*, we also provide a sample output that a human validator would obtain from the automated validation script for this dataset. Listing 2 shows this output, which notes important information like total images found in the original archives, number of images assigned to each set, and class statistics. The validation script also presents statistics on status codes across images and whether certain status codes can be independently confirmed (exact duplicate folders etc.). This information can be utilized by the human validator to make sure the customized script is running as expected.

```
verify md5sums
extract dataset
verify all archive files are used (this will be a manual check):
##########  archive.zip  ##########                              #
#############################################################
  DATASET_FILE_    1     =     "    archive    .    zip    "
# #############################################################
parse dataset
validate parsed dataset
validate classes dict
validate images list
get all remaining image details
get image details using 26 workers...
took 9 sec
# ######################################################################
statistics:
  image statistics:
```

```
    images total : 40000
    images found independently : 40000
      inoriginal_files: 40000
   problem-free images: 40000
       imagesintrainingset: 40000
      imagesinvalidationset: 0
      imagesintestingset: 0
    problematic images : 0
      class_id=-1 images: 0
      set_id=-1 images: 0
      all3 (problem/class_id/set_id) images: 0
  classstatistics:
    classes ( >=0): 2
    first 2  classes:
     -id- , - name - , - count -
        0 , Negative ,   20000
        1 , Positive ,   20000
    classnamesinalphabetical order :
    True classids are contiguous : True
    shortestclassname : Negative
    longestclassname : Negative
    allclasses are the same  size:
                                20000
######################################################################
create database.sqlite
run additional checks on  hashes
image count : 40000 folder count:2
there are no folders full of images that are duplicated elsewhere
there are 1519 ( problem =0) hashes that  arenotunique
create examples
```

Code Snippet. 2: A simple example of what the output from the automated validation script looks like.

S.2.2. Complex Example

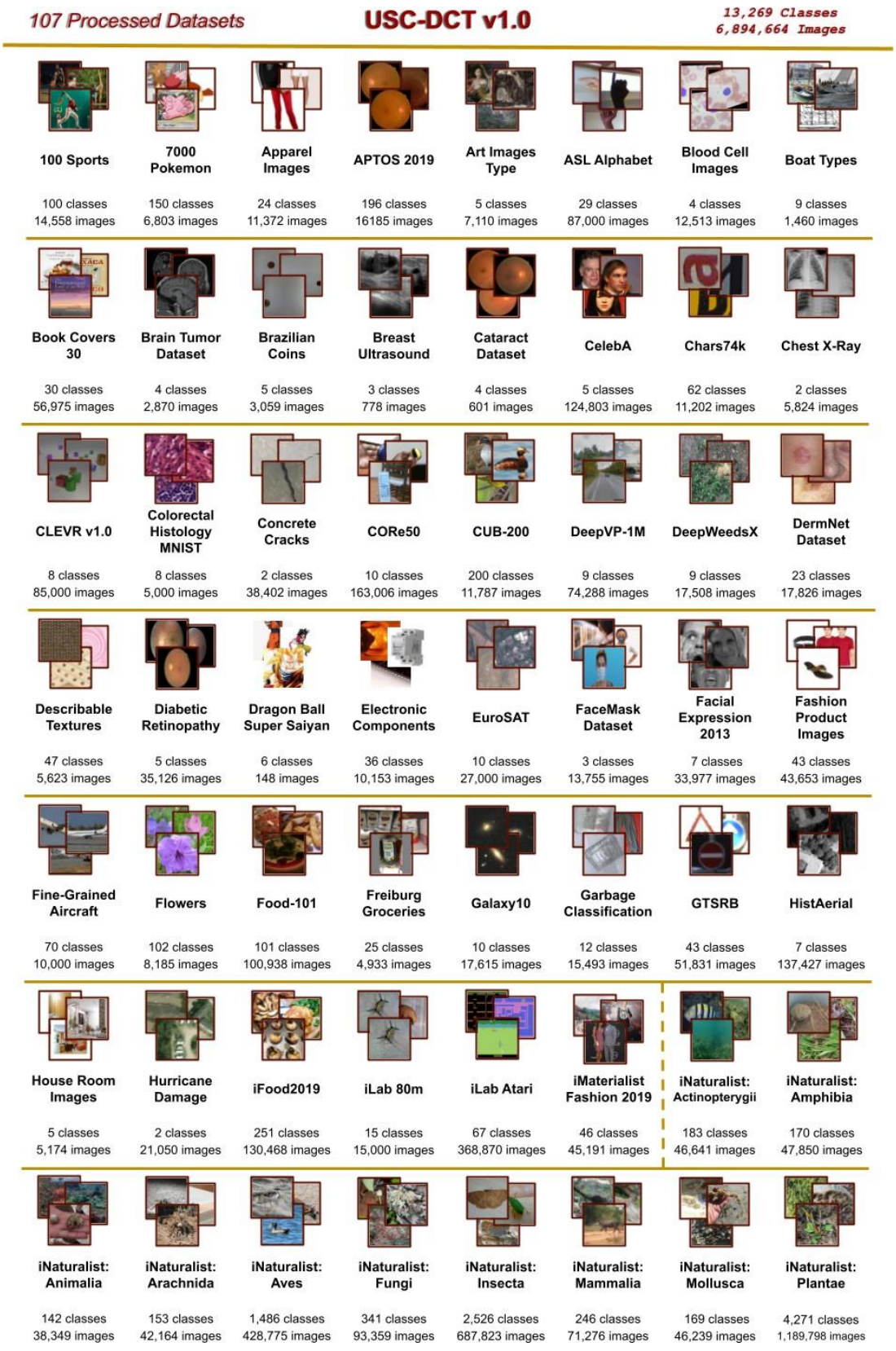
A slightly more complex (but not too complex) example is from the Oxford Buildings dataset. Here, the folder structure was less straightforward, which necessitated some additional code. It bears mentioning that for some of the datasets, the *parse_dataset()* function runs into hundreds of lines, and sometimes requires the use of additional files that act as keys for the class structure.

```
def parse_dataset(
    dataset_root_path: str,
) -> tuple[list[DatasetImage], dict[str, int]]:
    """ This returns a list of all images within the dataset.
    This includes ** all images ** in both ./original_files/ and,
    if necessary, ./converted_files/
    """

    image_list: list[DatasetImage] = []
    class_dict: dict[str, int] = {}

    # get all file paths
    files_list = get_all_files_in_directory(
        dataset_root_path + "original_files/", dataset_root_path
    )

    # splitting the file list into images and non-images(
    image_path_list, text_path_list) = filter_file_list(files_list)

    # generate list of classes AND list of file labels
    class_list = []
    text2img_dict: dict[str, int] = {}
    for filePath in text_path_list:
        # remove all filepaths, break at counting number
        # (pattern: .../.../LABEL_#_...)
        className = re.split(r"\d+", filePath.replace("original_files/", ""))[0][:-1]
        # Collect Unique Class Names
        if className not in class_list:
            class_list.append(className)

    # creating class_dict, and text2img_dict
    for id, classStr in enumerate(sorted(class_list, key=str.casefold)):
        class_dict.update({classStr: id})

        # Create list of files denoted for the given classStr
        textLabels = []
        for fileText in sorted(text_path_list):
            # If class is in file name (and not a "Query" file),
```

```
            # add to collection, else ignore
            if classStr in fileText and "query" not in fileText:
                for x in (
                    open(str(dataset_root_path + fileText), "r").read().split("\n")[:-1]
                ):
                    textLabels.append(x)
            else:
                pass
        # Generate dict with the image name: id#
        n = 0
        for txtImg in textLabels:
            n += 1
            text2img_dict.update({"original_files/" + txtImg + ".jpg": id})

    # go through each image, ensure valid, flag any problems, else find label
    for image_path in sorted(image_path_list):
        problem_value, set_value = 0, 0

        # first address problem_value
        if not is_image_valid ( dataset_root_path + image_path ):
            problem_value = 1
            class_int = -1

        # If not in listed named files
        elif image_path not in text2img_dict.keys():
            class_int = -1
            problem_value = 4
        else:
            class_int = text2img_dict[image_path]

        # once problem_value != 0, set all else as -1
        if problem_value != 0:
            class_int = -1
            set_value = -1

        assert isinstance(class_int, int)

        image_list.append(
            DatasetImage(
                relative_path=image_path,
                class_id=class_int,
                set_id=set_value,
                problem=problem_value,
            )
        )

    return (image_list, class_dict)
```

Code Snippet. 3: A more-complex example example of a prepare_dataset function.

S.3. USC-DCT Visual Overview



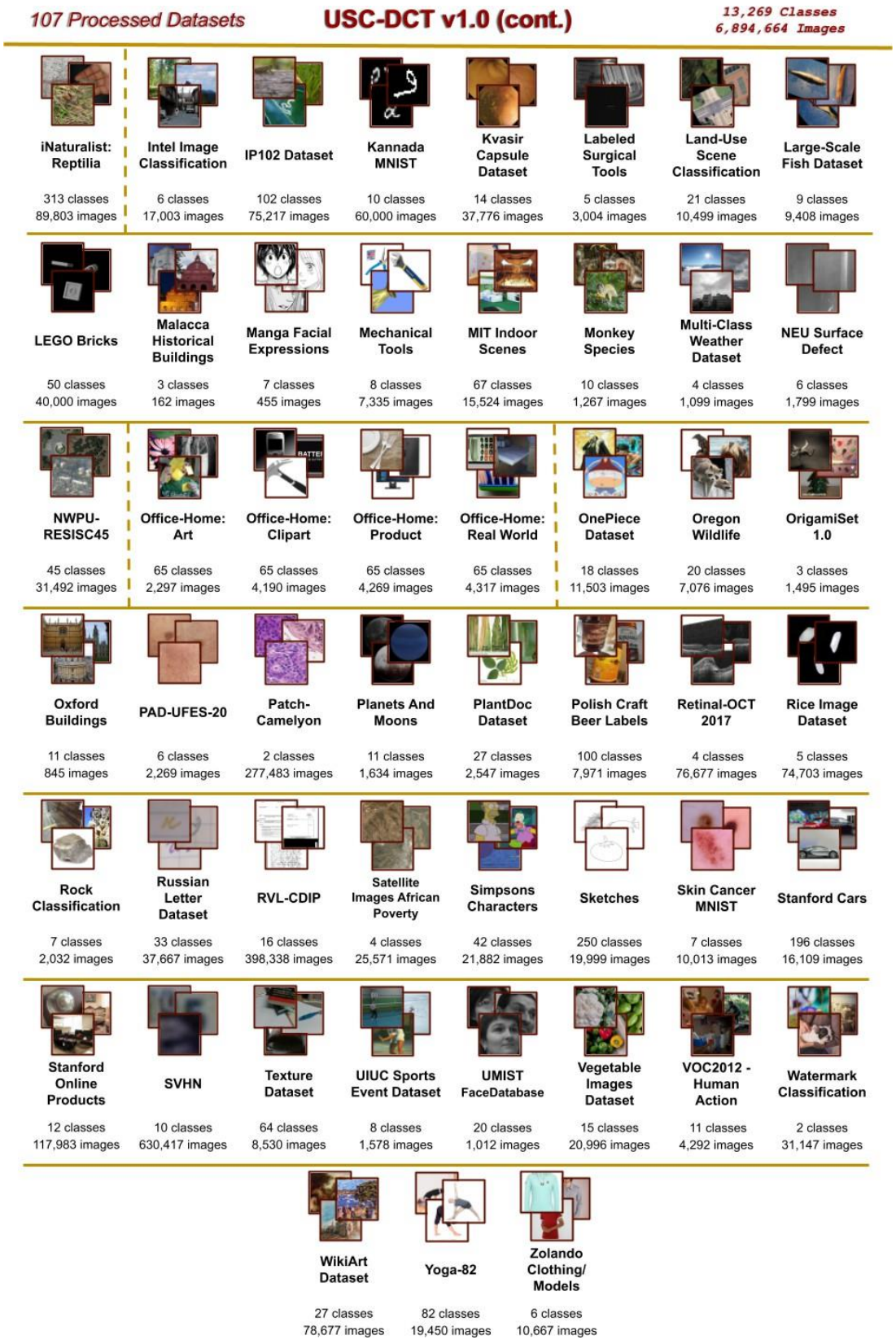


Figure S2. Visual Overview of the Datasets used within USC-DCT. Dashed-Lines represent data subsets created by splitting one larger dataset (e.g. iNaturalist). See also: Table 4 in the main paper.