```
import argparse
from distutils import dir_util
from pathlib import Path
from pytube import YouTube
import json

ROOT = Path(__file__).parent


def videos(
,    overwrite: bool = False,
,) -> None:
    """Download the train video and the test video from YouTube and save them to files.

    Args:
        overwrite (bool, optional): overwrite existing files. Defaults to False.
    """

    # Load settings.json
    with open(ROOT / "settings.json") as f:
        settings = json.load(f)

    # Create varaibles
    videos = ROOT / settings["videos"]
    urls = settings["vidoes_urls"]
    names = settings["vidoes_names"]

    # Remove all files in directory if overwrite is true
    if overwrite and videos.exists():
        dir_util.remove_tree(str(videos))

    # Create the directory if it doesn't exist
    videos.mkdir(exist_ok=True)

    # Download all vidoes
    for url, name in zip(urls, names):
        stream = YouTube(url).streams.get_highest_resolution()
        stream.download(videos, name)


def parse_opt(known: bool = False) -> argparse.Namespace:
    """Set up command line arguments

    Args:
        known (bool, optional): if arguments are known, throw an error if an unknown argument are
passed in. Defaults to False.

    Returns:
        argparse.Namespace: parsed arguments.
    """
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-o", "--overwrite", action="store_true", help="overwrite the directory"
    )
    opt = parser.parse_known_args()[0] if known else parser.parse_args()
    return opt


# Run this code if this script is called from a command line
if __name__ == "__main__":
    opt = parse_opt()
    videos(overwrite=opt.overwrite)
```

```
import argparse
import cv2
from pathlib import Path
import numpy as np
import json

ROOT = Path(__file__).parent


def extract(
,    n_frames: int = 1000,
,    overwrite: bool = False,
,) -> None:
    """Extract a certain number from frames from the train video without duplication.

    Args:
        n_frames (int, optional): the number of frames to extract. Defaults to 1000.
        overwrite (bool, optional): overwrite existing files. Defaults to False.
    """
    # Load settings.json
    with open(ROOT / "settings.json") as f:
        settings = json.load(f)

    # Create variables
    train_video = ROOT / settings["videos"] / settings["vidoes_names"][0]
    frames = ROOT / settings["frames"]

    # Remove all files in the directory if overwrite is true
    # Note: from distutils.dir_util import remove_tree cause directory creation problem due to
race condition.
    if overwrite and frames.exists():
        for f in frames.glob("*.png"):
            f.unlink()

    # Create the directory if it doesn't exist
    frames.mkdir(exist_ok=True)

    # Create video capture object
    cap = cv2.VideoCapture(str(train_video))

    # Find the number of frames in the video
    nums_frame = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    # Find all frames that have been extracted already
    existed_frames = [int(f.stem) for f in frames.glob("*.png")]

    # Find all frames that haven't been extracted yet
    new_frames = [i for i in range(nums_frame) if i not in existed_frames]

    # Randomly pick frames
    picked_frames = np.random.choice(
        new_frames, n_frames - len(existed_frames), replace=False
    )

    # Write frames to the directory
    for frame in picked_frames:
        cap.set(cv2.CAP_PROP_POS_FRAMES, frame)
        _, frame_val = cap.read()
        cv2.imwrite(str(frames / f"{frame}.png"), frame_val)
```

```python
def parse_opt(known: bool = False) -> argparse.Namespace:
    """Set up command line arguments

    Args:
        known (bool, optional): if arguments are known, throw an error if an unknown argument are
passed in. Defaults to False.

    Returns:
        argparse.Namespace: parsed arguments.
    """
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "--n_frames", "-n", default=1000, type=int, help="number of frames to extact"
    )
    parser.add_argument(
        "-o", "--overwrite", action="store_true", help="overwrite the directory"
    )
    opt = parser.parse_known_args()[0] if known else parser.parse_args()
    return opt


# Run this code if this script is called from a command line
if __name__ == "__main__":
    opt = parse_opt()
    extract(n_frames=opt.n_frames, overwrite=opt.overwrite)
```

```
from pathlib import Path
from distutils import dir_util
from zipfile import ZipFile
from ruamel.yaml import YAML
import numpy as np
import argparse
from mega import Mega
import json
import os

ROOT = Path(__file__).parent


def train_test_split(x: np.array, y: np.array, split: float = 0.7):
    """Split a given dataset into train dataset and test dataset

    Args:
        x (np.array): data.
        y (np.array): labels.
        split (float, optional): splitting ratio. Defaults to 0.7.

    Returns:
        tuple: train_data, train_labels, test_data, test_labels
    """
    indices = np.random.choice(range(len(x)), int(split * len(x)), replace=False)

    return x[indices], y[indices], np.delete(x, indices), np.delete(y, indices)


def train_valid_test_split(wd: Path, ratio: float = 1.0):
    """Split the dataset in a given directory into three datasets: train, valid, and test.

    Args:
        wd (Path): dataset path.
        ratio (float, optional): the percentage of data to keep. Defaults to 1.0.
    """
    # Form paths to the three datasets: train, valid, test
    train = wd / "train"
    valid = wd / "valid"
    test = wd / "test"

    # Create images folders for all datasets
    (train / "images").mkdir(parents=True, exist_ok=True)
    (valid / "images").mkdir(parents=True, exist_ok=True)
    (test / "images").mkdir(parents=True, exist_ok=True)

    # Create labels folders for all datasets
    (train / "labels").mkdir(parents=True, exist_ok=True)
    (valid / "labels").mkdir(parents=True, exist_ok=True)
    (test / "labels").mkdir(parents=True, exist_ok=True)

    # Find the dataset
    images = np.array(list((train / "images").glob("*")))
    labels = np.array(list((train / "labels").glob("*")))

    # Remove data from the dataset until a certain percentage of data remained
    size = len(images)
    indices = np.random.choice(range(size), int((1.0 - ratio) * size), replace=False)
    for image, label in zip(images[indices], labels[indices]):
        image.unlink()
```

```
            label.unlink()
        images = np.delete(images, indices)
        labels = np.delete(labels, indices)

        # Split the dataset into train, valid, and test
        train_images, train_labels, images, labels = train_test_split(images, labels)
        valid_images, valid_labels, test_images, test_labels = train_test_split(
            images, labels
        )

        # Move data from the train dataset into the valid dataset
        for image, label in zip(valid_images, valid_labels):
            os.rename(str(image), str(valid / "images" / image.name))
            os.rename(str(label), str(valid / "labels" / label.name))

        # Move data from the train dataset into the test dataset
        for image, label in zip(test_images, test_labels):
            os.rename(str(image), str(test / "images" / image.name))
            os.rename(str(label), str(test / "labels" / label.name))

        # Update data.yaml
        yaml = YAML()
        yaml.width = 4096
        data = None
        with open(wd / "data.yaml", "r") as f:
            data = yaml.load(f)
        if data:
            data["train"] = str(train / "images")
            data["val"] = str(valid / "images")
            data["test"] = str(test / "images")
        with open(wd / "data.yaml", "w") as f:
            yaml.dump(data, f)


def datasets(
,    overwrite: bool = False,
,):
    """Download dataset.zip from Mega.io and split it into multiple datasets of various sizes.

    Args:
        overwrite (bool, optional): overwrite existing files. Defaults to False.
    """
    # Load settings.json
    with open(ROOT / "settings.json") as fil:
        settings = json.load(fil)

    # Create variables
    datasets = ROOT / settings["datasets"]
    url = settings["datasets_url"][settings["datasets_select"]]
    ratios = settings["ratios"]

    # Remove all files in the directory if overwrite is true
    if overwrite and datasets.exists():
        dir_util.remove_tree(str(datasets))

    # Make the directory if it doesn't exist
    datasets.mkdir(exist_ok=True)

    # Download the dataset zip file
    dataset_zip = datasets / "dataset.zip"
    if not dataset_zip.exists():
        try:
            m = Mega().login()
            m.download_url(url, str(dataset_zip.parent), dataset_zip.name)
        except:
            pass

    # Extract the zip file into a temporary folder
    temp = datasets / "temporary"
    temp.mkdir(exist_ok=True)
    zipfile = ZipFile(dataset_zip)
    zipfile.extractall(temp)
    zipfile.close()

    # Update filenames of images and labels
    for fil in temp.rglob("*"):
        new_name = fil.stem.split("_")[0]
        fil.rename(fil.parent / f"{new_name}{fil.suffix}")

    # Create new datasets from the dataset based on given ratios
    for ratio in ratios:
        src = temp
        dst = datasets / str(ratio)

        # Copy contents from the temporary folder to the new dataset folder
        dir_util.copy_tree(str(src), str(dst))

        # Split the new dataset into train, val, test dataset with a given ratio
        train_valid_test_split(dst, ratio)

    # Remove the temporary folder
    dir_util.remove_tree(temp)


def parse_opt(known: bool = False) -> argparse.Namespace:
    """Set up command line arguments

    Args:
        known (bool, optional): if arguments are known, throw an error if an unknown argument are
passed in. Defaults to False.

    Returns:
        argparse.Namespace: parsed arguments.
    """
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-o", "--overwrite", action="store_true", help="overwrite the directory"
    )
    opt = parser.parse_known_args()[0] if known else parser.parse_args()
    return opt


# Run this code if this script is called from a command line
if __name__ == "__main__":
    opt = parse_opt()
    datasets(overwrite=opt.overwrite)
```

```
from pathlib import Path
from urllib3 import PoolManager
from zipfile import ZipFile
from distutils import dir_util
import argparse
import json

ROOT = Path(__file__).parent


def yolov5_no_update(wd: Path):
    """Modify YOLOv5 files such that it stops checking for update with git when training.

    Args:
        wd (Path): YOLOv5 path.
    """
    with open(wd / "train.py", "r+") as f:
        lines = f.readlines()
        for i, line in enumerate(lines):
            if "check_git_status()" in line:
                lines[i] = lines[i].replace("check_git_status()", "#check_git_status()")
        f.seek(0)
        f.writelines(lines)


def yolov5_visualize_no_new_folders(wd: Path):
    """Modify YOLOv5 files such that it stops creating new folders when visualizing layers.

    Args:
        wd (Path): YOLOv5 path.
    """
    with open(wd / "detect.py", "r+") as f:
        lines = f.readlines()
        for i, line in enumerate(lines):
            if (
                "visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if visualize
else False"
                in line
            ):
                lines[i] = lines[i].replace(
                    "visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if
visualize else False",
                    "visualize = increment_path(save_dir / Path(path).stem, mkdir=True,
exist_ok=exist_ok) if visualize else False",
                )
        f.seek(0)
        f.writelines(lines)


def yolov5_add_init(wd: Path):
    """Modifying YOLOv5 files such that it can be imported as a package.

    Args:
        wd (Path): YOLOv5 path.
    """
    with open(wd / "__init__.py", "w") as f:
        f.seek(0)
        f.write(
            "from _yolov5.train import run as train\nfrom _yolov5.val import run as val\nfrom
_yolov5.detect import run as detect"
        )
```

```
def yolov5(
,    overwrite: bool = False,
,) -> None:
    """Download YOLOv5 source code from github.

    Args:
        overwrite (bool, optional): overwrite existing files. Defaults to False.
    """

    # Load settings.json
    with open(ROOT / "settings.json") as f:
        settings = json.load(f)

    # Create variables
    yolov5 = ROOT / settings["yolov5"]
    url = settings["yolov5_url"]

    # Remove all files in the directory if overwrite is true
    if overwrite and yolov5.exists():
        dir_util.remove_tree(str(yolov5))

    # Create the directory if it doesn't exist
    yolov5.mkdir(exist_ok=True)

    # Download the YOLOv5 zip file from github
    yolov5_zip = yolov5 / "yolov5.zip"
    if not yolov5_zip.exists():
        http = PoolManager()
        req = http.request("GET", url)
        with open(yolov5_zip, "wb") as f:
            f.write(req.data)

    # Extract the zip file to a temporary folder
    zipfile = ZipFile(yolov5_zip)
    zipfile.extractall(yolov5)
    zipfile.close()
    temp = yolov5 / zipfile.filelist[0].filename

    # Copy contents of the temporary folder to the YOLOv5 folder
    dir_util.copy_tree(str(temp), str(yolov5))

    # Modify YOLOv5 files for various purposes
    yolov5_no_update(yolov5)
    yolov5_visualize_no_new_folders(yolov5)
    yolov5_add_init(yolov5)

    # Remove the temporary folder
    dir_util.remove_tree(temp)


def parse_opt(known: bool = False) -> argparse.Namespace:
    """Set up command line arguments

    Args:
        known (bool, optional): if arguments are known, throw an error if an unknown argument are
passed in. Defaults to False.

    Returns:
        argparse.Namespace: parsed arguments.
    """
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-o", "--overwrite", action="store_true", help="overwrite the directory"
    )
    opt = parser.parse_known_args()[0] if known else parser.parse_args()
    return opt


# Run this code if this script is called from a command line
if __name__ == "__main__":
    opt = parse_opt()
    yolov5(overwrite=opt.overwrite)
```

```python
import sys
from pathlib import Path
from distutils import dir_util
import json
import argparse


# Add yolov5 folder to path
ROOT = Path(__file__).parent


def train(overwrite: bool = False):
    """Train models based on existing datasets.

    Args:
        overwrite (bool, optional): overwrite existing files. Defaults to False.
    """

    # Load settings.json
    with open(ROOT / "settings.json") as f:
        settings = json.load(f)

    # Create variables
    datasets = ROOT / settings["datasets"]
    models = ROOT / settings["models"]
    yolov5 = ROOT / settings["yolov5"]
    images_size = settings["datasets_images_size"][settings["datasets_select"]]

    # Add yolov5 to path and import it
    sys.path.append(str(yolov5))
    import _yolov5 as yolov5

    if overwrite and models.exists():
        dir_util.remove_tree(models)
    models.mkdir(exist_ok=True)

    datasets = list(filter(lambda dataset: dataset.is_dir(), datasets.glob("*/")))

    for dataset in datasets:
        # Hyperparameter
        # Max batch_size for 12gb vram
        # 1280   => XL: 1,  L: 4,    M: 6,      S: 14,       N: 26
        # 640    => XL: 8,  L: 16,   M: 28,     S: 54,      N: 96
        weights = "yolov5s.pt"
        epochs = 100000
        batch_size = 16
        patience = 100

        # Other paremeters
        device = 0

        # Pick the correct pretrained weights based on the dataset
        weights = (
            weights[: weights.find(".")] + "6" + weights[weights.find(".") :]
            if settings["datasets_select"] == 1
            else weights
        )

        # Check if the model is partially trained
        last_pt = models / dataset.name / "train/weights/last.pt"
        resume = last_pt.exists()

        # Try to resume training
        if resume:
            try:
                yolov5.train(resume=last_pt)
            except:
                pass
        # Train a model from scratch
        else:
            yolov5.train(
                weights=models / weights,
                data=dataset / "data.yaml",
                epochs=epochs,
                batch_size=batch_size,
                imgsz=images_size,
                device=device,
                project=models / dataset.name,
                name="train",
                exist_ok=True,
                patience=patience,
            )

        # Validate the model with test dataset
        yolov5.val(
            data=dataset / "data.yaml",
            weights=models / dataset.name / "train/weights/best.pt",
            batch_size=batch_size,
            imgsz=images_size,
            task="test",
            device=device,
            verbose=True,
            project=models / dataset.name,
            name="test",
            exist_ok=True,
        )

        # Detect bounding boxes and coffidence with the dataset
        yolov5.detect(
            weights=models / dataset.name / "train/weights/best.pt",
            source=dataset / "test/images",
            imgsz=[images_size, images_size],
            device=device,
            save_txt=True,
            save_conf=True,
            project=models / dataset.name,
            name="test",
            exist_ok=True,
        )


def parse_opt(known: bool = False) -> argparse.Namespace:
    """Set up command line arguments

    Args:
        known (bool, optional): if arguments are known, throw an error if an unknown argument are
passed in. Defaults to False.

    Returns:
        argparse.Namespace: parsed arguments.
    """
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-o", "--overwrite", action="store_true", help="overwrite the directory"
    )
    opt = parser.parse_known_args()[0] if known else parser.parse_args()
    return opt


# Run this code if this script is called from a command line
if __name__ == "__main__":
    opt = parse_opt()
    train(overwrite=opt.overwrite)
```

```python
from pathlib import Path
import argparse
import json
from distutils import dir_util
import sys

import shutil

ROOT = Path(__file__).parent


def detect(overwrite: bool = False):
    """Utilize existing models to detect characters in all vidoes.

    Args:
        overwrite (bool, optional): overwrite existing files. Defaults to False.
    """

    # Load settings.json
    with open(ROOT / "settings.json", "r") as f:
        settings = json.load(f)

    # Create variables
    videos = ROOT / settings["videos"]
    models = ROOT / settings["models"]
    detect = ROOT / settings["detect"]
    yolov5 = ROOT / settings["yolov5"]
    images_size = settings["datasets_images_size"][settings["datasets_select"]]

    # Add yolov5 to path and import it
    sys.path.append(str(yolov5))
    import _yolov5 as yolov5

    # Remove all files in directory if overwrite is true
    if overwrite and detect.exists():
        dir_util.remove_tree(str(detect))
    detect.mkdir(exist_ok=True)

    # Get all vidoes and models
    videos = list(videos.glob("*"))
    models = list(filter(lambda model: model.is_dir(), models.glob("*")))

    # Pairs every vidoes with every models
    videos_and_models = [(video, model) for video in videos for model in models]

    for video, model in videos_and_models:

        # Copy video from _video into a temporary folder and rename it to video_model_extension
        (detect / "temp").mkdir(exist_ok=True)
        shutil.copy2(video, detect / f"temp/{video.stem}_{model.name}{video.suffix}")
        video = detect / f"temp/{video.stem}_{model.name}{video.suffix}"

        # Detect bounding boxes and confidences in the video
        yolov5.detect(
            weights=model / "train/weights/best.pt",
            source=video,
            imgsz=[images_size, images_size],
            device=0,
            project=detect.parent,
            name=detect.name,
            exist_ok=True,
        )

    # Remove the temporary folder
    dir_util.remove_tree(str(detect / "temp"))


def parse_opt(known: bool = False) -> argparse.Namespace:
    """Set up command line arguments

    Args:
        known (bool, optional): if arguments are known, throw an error if an unknown argument are
passed in. Defaults to False.

    Returns:
        argparse.Namespace: parsed arguments.
    """
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-o", "--overwrite", action="store_true", help="overwrite the directory"
    )
    opt = parser.parse_known_args()[0] if known else parser.parse_args()
    return opt


# Run this code if this script is called from a command line
if __name__ == "__main__":
    opt = parse_opt()
    detect(overwrite=opt.overwrite)
```

```json
{
    "videos": "_videos",
    "vidoes_urls": [
        "https://youtu.be/rilFfbm7j8k",
        "https://youtu.be/cqyziA30whE"
    ],
    "vidoes_names": [
        "train.mp4",
        "test.mp4"
    ],
    "frames": "_frames",
    "yolov5": "_yolov5",
    "yolov5_url": "https://github.com/ultralytics/yolov5/archive/refs/heads/master.zip",
    "datasets": "_datasets",
    "datasets_url": [
        "https://mega.nz/file/z3YCWBYC#n6Klmpr3XB6ula_WOSriem5W0gnNgEZk3tZBVm5wDQ8",
        "https://mega.nz/file/uyAwFZaK#9lZAk6_Pn0W9yB40KlfZx7e5WjYgTjdzIVogt6qv1jA"
    ],
    "datasets_images_size": [
        640,
        1280
    ],
    "datasets_select": 0,
    "ratios": [
        0.25,
        0.5,
        0.75,
        1.0
    ],
    "models": "_models",
    "detect": "_detect"
}
```

```
absl-py==0.15.0
black==21.10b0
cachetools==4.2.4
certifi==2021.10.8
charset-normalizer==2.0.7
click==8.0.3
colorama==0.4.4
cycler==0.11.0
Cython==0.29.24
google-auth==2.3.3
google-auth-oauthlib==0.4.6
grpcio==1.41.1
idna==3.3
kiwisolver==1.3.2
Markdown==3.3.4
matplotlib==3.4.3
mega.py==1.0.8
mypy-extensions==0.4.3
numpy==1.21.3
oauthlib==3.1.1
opencv-python==4.5.4.58
pandas==1.3.4
pathlib==1.0.1
pathspec==0.9.0
Pillow==8.4.0
platformdirs==2.4.0
protobuf==3.19.1
pyasn1==0.4.8
pyasn1-modules==0.2.8
pycocotools==2.0.2
pycryptodome==3.11.0
pyparsing==3.0.4
python-dateutil==2.8.2
pytube==11.0.1
pytz==2021.3
PyYAML==6.0
regex==2021.11.2
requests==2.26.0
requests-oauthlib==1.3.0
rsa==4.7.2
ruamel.yaml==0.17.17
ruamel.yaml.clib==0.2.6
scipy==1.7.1
seaborn==0.11.2
six==1.16.0
tenacity==5.1.5
tensorboard==2.7.0
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.0
tomli==1.2.2
torch==1.10.0+cu113
torchaudio==0.10.0+cu113
torchvision==0.11.1+cu113
tqdm==4.62.3
typing-extensions==3.10.0.2
urllib3==1.26.7
Werkzeug==2.0.2
wincertstore==0.2
```