```python
"""
EECS 445 - Introduction to Machine Learning
Fall 2022  - Project 2

Script to create an augmented dataset.
"""
from PIL import Image
from array import array
import numpy
import tensorflow as tf
import argparse
import csv
import glob
import os
import sys
import numpy as np
from scipy.ndimage import rotate
from imageio import imread, imwrite

IMAGE_SIZE = 64


def Rotate(deg=20):
    """Return function to rotate image."""

    def _rotate(img):
        """Rotate a random integer amount in the range (-deg, deg).

        Keep the dimensions the same and fill any missing pixels with black.

        :img: H x W x C numpy array
        :returns: H x W x C numpy array
        """
        # TODO
        angle = np.random.randint(-deg, deg)
        img = rotate(img, angle, reshape=False)
        return img

    return _rotate


def Grayscale():
    """Return function to grayscale image."""
    # def getRed(redVal):
    #     return '#%02x%02x%02x' % (redVal, 0, 0)

    # def getGreen(greenVal):
    #     return '#%02x%02x%02x' % (0, greenVal, 0)

    # def getBlue(blueVal):
    #     return '#%02x%02x%02x' % (0, 0, blueVal)

    # Grayscale = (R + G + B / 3)
    # For each pixel,
    # 1- Get pixels red, green, and blue
    # 2- Calculate the average value
    # 3- Set each of red, green, and blue values to average value

    def _grayscale(img):
        """Return 3-channel grayscale of image.

        Compute grayscale values by taking average across the three channels.

        Round to the nearest integer.

        :img: H x W x C numpy array
        :returns: H x W x C numpy array

        """

        # TODO
        # for p in img:
        #     red = getRed(p[0])
        #     green = getGreen(p[1])
        #     blue = getBlue(p[2])
        #     average = (red + green + blue) / 3
        #     p[0] = average
        #     p[1] = average
        #     p[2] = average
        # for p in img:
        #     gray = sum(p)/3
        #     for i in range(3):
        #         p[i] = gray

        average = np.mean(img, axis=2)
        out = np.stack((average, average, average), axis=2)
        out = np.rint(out)
        out = out.astype(np.uint8)

        # img = color.rgb2gray(img)
        return out

    return _grayscale


def flip_images_horiz():
    def _flip_images_horiz(img):
        original_img = Image.fromarray(img)
        horz_img = original_img.transpose(method=Image.FLIP_LEFT_RIGHT)
        return horz_img
    return _flip_images_horiz

    # X_flip = []
    # tf.compat.v1.disable_eager_execution()
    # tf.compat.v1.reset_default_graph()

    # image = cv2.imread(img)
    # flippedimage = cv2.flip(image, 0)

    # horz_img.save("horizontal.png")

    # X = tf.compat.v1.placeholder(
    #     tf.float32, shape=(IMAGE_SIZE, IMAGE_SIZE, 3))
    # tf_img1 = tf.compat.v1.image.flip_left_right(img)

    # tf_img2 = tf.compat.v1.image.flip_up_down(X)
    # tf_img3 = tf.compat.v1.image.transpose_image(X)
    # with tf.compat.v1.Session() as sess:
    #     sess.run(tf.compat.v1.global_variables_initializer())
    #     np.ndarray.resize(img, (64, 64, 3))
    #     img = np.ndarray.reshape(img, (64, 64, 3))
    # tf_img1 = tf.compat.v1.image.flip_left_right(img)
    #     resized = np.resize(img, (64, 192))
    #     img = np.reshape(resized, (64, 64, 3))

    # X_flip.extend(tf_img1)
    # X_flip.extend(tf_img2)
    # X_flip.extend(tf_img3)
    # X_flip = np.array(X_flip, dtype=np.float32)


def flip_images_vertical():
    def _flip_images_vertical(img):
        original_img = Image.fromarray(img)
        vertical_img = original_img.transpose(method=Image.FLIP_TOP_BOTTOM)
        return vertical_img
    return _flip_images_vertical


def add_salt_pepper_noise():
    def _add_salt_pepper_noise(X_imgs):
        # Need to produce a copy as to not modify the original image
        X_imgs_copy = X_imgs.copy()
        row, col, _ = X_imgs_copy[0].shape
        salt_vs_pepper = 0.2
        amount = 0.004
        num_salt = np.ceil(amount * X_imgs_copy[0].size * salt_vs_pepper)
        num_pepper = np.ceil(
            amount * X_imgs_copy[0].size * (1.0 - salt_vs_pepper))
```

```python
        for X_img in X_imgs_copy:
            # Add Salt noise
            coords = [np.random.randint(0, i - 1, int(num_salt))
                      for i in X_img.shape]
            X_img[coords[0], coords[1], :] = 1

            # Add Pepper noise
            coords = [np.random.randint(0, i - 1, int(num_pepper))
                      for i in X_img.shape]
            X_img[coords[0], coords[1], :] = 0
        return X_imgs_copy

    return _add_salt_pepper_noise


def augment(filename, transforms, n=1, original=True):
    """Augment image at filename.

    :filename: name of image to be augmented
    :transforms: List of image transformations
    :n: number of augmented images to save
    :original: whether to include the original images in the augmented dataset or not
    :returns: a list of augmented images, where the first image is the original

    """
    print(f"Augmenting {filename}")
    img = imread(filename)
    res = [img] if original else []
    for i in range(n):
        new = img
        for transform in transforms:
            new = transform(new)
        res.append(new)
    return res


def main(args):
    """Create augmented dataset."""
    reader = csv.DictReader(open(args.input, "r"), delimiter=",")
    writer = csv.DictWriter(
        open(f"{args.datadir}/augmented_dogs.csv", "w"),
        fieldnames=["filename", "semantic_label",
                    "partition", "numeric_label", "task"],
    )
    augment_partitions = set(args.partitions)

    # TODO: change `augmentations` to specify which augmentations to apply
    # Rotate() Grayscale() flip_images() add_salt_pepper_noise()
    # augmentations = [Grayscale()]
    # augmentations = [Rotate()]   flip_images_vertical(),flip_images_horiz()
    augmentations = [Rotate(), Grayscale(), flip_images_horiz()]

    writer.writeheader()
    os.makedirs(f"{args.datadir}/augmented/", exist_ok=True)
    for f in glob.glob(f"{args.datadir}/augmented/*"):
        print(f"Deleting {f}")
        os.remove(f)
    for row in reader:
        if row["partition"] not in augment_partitions:
            imwrite(
                f"{args.datadir}/augmented/{row['filename']}",
                imread(f"{args.datadir}/images/{row['filename']}"),
            )
            writer.writerow(row)
            continue
        imgs = augment(
            f"{args.datadir}/images/{row['filename']}",
            augmentations,
            n=1,
            original=True,  # TODO: change to False to exclude original image.
        )
        for i, img in enumerate(imgs):
            fname = f"{row['filename'][:-4]}_aug_{i}.png"
            imwrite(f"{args.datadir}/augmented/{fname}", img)
            writer.writerow(
                {
                    "filename": fname,
                    "semantic_label": row["semantic_label"],
                    "partition": row["partition"],
                    "numeric_label": row["numeric_label"],
                    "task": row["task"],
                }
            )


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("input", help="Path to input CSV file")
    parser.add_argument("datadir", help="Data directory", default="./data/")
    parser.add_argument(
        "-p",
        "--partitions",
        nargs="+",
        help="Partitions (train|val|test|challenge|none)+ to apply augmentations to. Defaults to train",
        default=["train"],
    )
    main(parser.parse_args(sys.argv[1:]))
```