

Machine Problem 1

Affine Transformations

Directions

The goal of the machine problem is to apply the concepts of affine transformations, specifically using geometric transformations. You are to submit two files for this activity: (1) a Jupyter notebook containing the solutions to the action items. Ensure you provide comments, discussions, and proper section divisions for your code. Please also include your answer to the Guide Questions in the Jupyter Notebook; (2) a PDF version of your Jupyter Notebook. You can provide a link to your submission resources or a zip file. The instructor will run it on their local machine, so make sure the codes and files are accessible and functional.

```
In [1]: import os
import cv2 as cv
import numpy as np
```

[Data Formatting] Given the image dataset:

- Reshape the images to (100,100,3)
- Save the transformed images as JPEG files in a separate directory.

```
In [2]: input_directory = '../media/dataset2'
output_directory = 'dataset2_resized'

if not os.path.exists(output_directory):
    os.makedirs(output_directory)

for filename in os.listdir(input_directory):
    if filename.endswith('.png') or filename.endswith('.jpg'):
        img_path = os.path.join(input_directory, filename)
        img = cv.imread(img_path)

        # resize the image to 100x100 pixels
        img_resized = cv.resize(img, (100, 100))

        # save the resized image as JPEG in a separate directory
        output_filename = os.path.join(output_directory, filename)
        cv.imwrite(output_filename, img_resized)
```

[Data Augmentation] Given the previous dataset:

- Create individual parametrized functions that can:
- Randomly put a black patch over a portion of the image

```
In [3]: def random_black_patch(img):
        h, w, _ = img.shape
        patch_size = np.random.randint(10, 30)
        x1 = np.random.randint(0, w - patch_size)
        y1 = np.random.randint(0, h - patch_size)
        img[y1:y1+patch_size, x1:x1+patch_size] = 0
        return img
```

- Shift an image sideward or upwards.

```
In [4]: def shift_image(img, shift_x, shift_y):
        h, w = img.shape[:2]
        M = np.float32([[1, 0, shift_x], [0, 1, shift_y]])
        shifted_img = cv.warpAffine(img, M, (w, h))
        return shifted_img
```

- Rotate an image either for

```
In [5]: def rotate_image(img, angle):
        h, w = img.shape[:2]
        center = (w // 2, h // 2)
        M = cv.getRotationMatrix2D(center, angle, 1.0) # positive angle -> counter clockwise, negative angle -> clockwise
        rotated_img = cv.warpAffine(img, M, (w, h))
        return rotated_img
```

- Flip an image either vertically or horizontally.

```
In [6]: def flip_image(image, value):
        return cv.flip(image, value) # 0 -> vertical, 1 -> horizontal
```

Produce a new augmented dataset with at least 100 images (original images included) using the functions made in the previous action item.

```
In [7]: augmented_output_dir = 'dataset2_augmented'
        if not os.path.exists(augmented_output_dir):
            os.makedirs(augmented_output_dir)
```

```
In [8]: image_count = 0
        for filename in os.listdir(output_directory):
            if filename.endswith('.jpg'):
                img_path = os.path.join(output_directory, filename)
                image = cv.imread(img_path)

                # augmentations with description to include on filenames
                augmentations = [
                    (random_black_patch(image.copy()), 'random_black_patch'),
                    (shift_image(image.copy(), 20, 0), 'shift_right_20px'),
```

```

        (shift_image(image.copy(), 0, -20), 'shift_up_20px'),
        (rotate_image(image.copy(), 45), 'rotate_45_degrees'),
        (flip_image(image.copy(), 0), 'flip_vertically'),
        (flip_image(image.copy(), 1), 'flip_horizontally'),
    ]

    # saving the augmented images with appropriate filenames
    for aug_img, aug_desc in augmentations:
        output_filename = f'{filename.split(".")[0]}_{aug_desc}.jpg'
        cv.imwrite(os.path.join(augmented_output_dir, output_filename), aug_img)
        image_count += 1

    # include the original resized image
    cv.imwrite(os.path.join(augmented_output_dir, filename), image)
    image_count += 1

if image_count >= 100:
    break

```

Guide Questions:

- **Define Data Augmentation and discuss its importance and the importance of understanding digital image processing for such an activity.**
 - Data augmentation is a method used to increase the variety of a dataset without needing to gather new data. This is achieved by applying different transformations to existing data, particularly images. Common transformations include rotating, flipping, scaling, cropping, and adjusting brightness. The goal of these changes is to produce diverse versions of the original dataset, which helps strengthen machine learning models by making them less prone to overfitting and better at generalizing to new, unseen data.
 - The significance of data augmentation is evident in its ability to boost model performance, especially when working with small datasets. By artificially enlarging the dataset, models can learn from a wider range of examples, which decreases the likelihood of overfitting to specific patterns. This is especially useful in deep learning tasks like image classification and object detection.
 - Having a solid understanding of digital image processing is essential for effective data augmentation. Knowledge of image processing techniques enables one to manipulate and modify images in a thoughtful and purposeful manner. A good grasp of these concepts ensures that the augmentations keep the data's integrity while adding variations that enhance learning. For instance, applying transformations such as changing color spaces, adding noise, or performing histogram equalization requires familiarity with image processing basics. This understanding helps ensure that the augmented data remains realistic and improves model performance without compromising the original content.
- What other data augmentation techniques are applicable and not applicable to the dataset you have produced? Why?
 - Other applicable data augmentation techniques for this dataset include scaling, brightness adjustment, and adding noise. Scaling the image by zooming in or out can provide variations in size and context, helping models recognize objects at different scales. Brightness adjustment simulates different lighting conditions, allowing the model to handle variations in exposure. Adding noise introduces random distortions, which can improve the model's robustness to imperfections in real-world data.
 - However, some techniques may not be as applicable. Convolution, for instance, applies filters to highlight edges or textures and is better suited for feature extraction or pre-processing. Convolution alters the image by emphasizing specific features such as edges, patterns, or textures, which may not add the kind of variability needed for data augmentation. Instead of diversifying the dataset, it risks making the model too focused on certain image characteristics,

potentially leading to reduced generalization. Similarly, shearing, which skews the image along the x or y axis, may distort the overall structure and integrity of objects in the image, making them less recognizable. Shearing can result in unrealistic transformations that might confuse the model, especially if the objects being recognized rely on a more accurate spatial relationship. Therefore, while some augmentations like scaling and brightness adjustment help improve model generalization, convolution and shearing could introduce distortions that aren't as beneficial for creating varied, yet realistic, training data.