Data Augmentation

Course:
INFO-6152 Deep Learning with Tensorflow & Keras 2

Developed by:
Mohammad Noorchenarboo

September 29, 2025

Current Section

1. Data Augmentation for Deep Learning
2. Introduction to OpenCV
3. Loading and Showing Images with OpenCV
4. Converting RGB Images to Grayscale
5. Resizing and Scaling Images
6. Translation
7. Flipping
8. Rotation

Data Augmentation for Deep Learning

Suppose you want to train a CNN to classify medical X-ray scans. You only have 2000 images. This is too small for a robust deep learning model. How can you expand the dataset without collecting new images?

Concept: Data Augmentation

Definition: Data augmentation artificially increases the size and diversity of training datasets by applying transformations to existing images.
Common transformations:

- Geometric: rotation, flipping, scaling, translation
- Photometric: brightness, contrast, noise addition
- Advanced: random cropping, cutout, mixup

Mathematical view: If X is an input image and T is a transformation, then the augmented dataset is:

$$X' = T(X), \quad \mathscr{D}' = \{T_i(X_j) \mid i = 1..k,\ j = 1..n\}$$

Recommendation:

- Use **TensorFlow/Keras** for training-time augmentations (easy and fast).
- Use **OpenCV** when you need advanced, customized augmentations (covered in the next section).

Data Augmentation for Deep Learning

Define Augmentation Pipeline

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Create an augmentation generator
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode="nearest"
)

# Apply augmentation on training data
# Example: x_train shape (2000, 100, 100, 3), y_train (2000,)
train_gen = datagen.flow(x_train, y_train, batch_size=32)

# Note:
# No fixed number of new images is created.
# Each epoch, generator produces a new augmented version.
```

Data Augmentation for Deep Learning

Define a Simple CNN Model

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Flatten, Dense

# Simple CNN for classification
model = Sequential([
    Conv2D(32, (3,3), activation="relu", input_shape=(100,100,3)),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(64, activation="relu"),
    Dense(2, activation="softmax")  # binary classification
])

model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
```

Data Augmentation for Deep Learning

Train Before vs After Augmentation

```
# Training WITHOUT augmentation
history_no_aug = model.fit(x_train, y_train,
                           epochs=5,
                           batch_size=32,
                           validation_split=0.2,
                           verbose=1)

# Training WITH augmentation
history_aug = model.fit(train_gen,
                        steps_per_epoch=len(x_train)//32,
                        epochs=5,
                        validation_data=(x_val, y_val),
                        verbose=1)

# Expected outcome:
# - history_no_aug: higher training accuracy but overfits quickly
# - history_aug: slower training accuracy, better validation accuracy
```

Data Augmentation for Deep Learning

Numerical Example: Dataset Expansion

Assume original dataset size = 2000 images. We apply $k = 5$ random transformations per image.

$$\text{New size} = 2000 + (2000 \times 5) = 12{,}000$$

- Original: 2000 samples
- After augmentation: 12,000 samples
- CNN has more data to learn from, reducing overfitting

Get more info $\rightarrow$ **TensorFlow Keras ImageDataGenerator Documentation**

Summary: Data Augmentation for Deep Learning

Aspect	**Explanation**
Purpose	Increase dataset size and diversity without collecting new images
Transformations	Geometric (rotate, flip), photometric (brightness, noise), advanced (mixup)
Formula	$X' = T(X)$, $\mathscr{D}' = \{T_i(X_j)\}$ expands dataset
Recommended Tools	TensorFlow/Keras (training-time), OpenCV (custom augmentations, next section)
Example	2000 images $\rightarrow$ 12,000 after augmentation

Current Section

1. Data Augmentation for Deep Learning
2. Introduction to OpenCV
3. Loading and Showing Images with OpenCV
4. Converting RGB Images to Grayscale
5. Resizing and Scaling Images
6. Translation
7. Flipping
8. Rotation

Introduction to OpenCV

Imagine you want to design a self-driving car. The system must detect lanes, traffic signs, and pedestrians in real time. To make this possible, you need a library that can efficiently handle images and video streams.

Concept: What is OpenCV?

Definition: OpenCV (Open Source Computer Vision) is a popular open-source library designed for real-time computer vision and image processing.

Key features:

- Provides tools for image and video input/output.
- Offers transformations such as resizing, filtering, and rotation.
- Includes advanced modules for object detection and face recognition.
- Written in C++ with bindings in Python, Java, and MATLAB.
- Optimized for real-time applications.

Mathematical foundation: An image can be represented as a matrix of pixels. Many OpenCV operations are expressed as transformations:

$$I' = T(I)$$

where I is the input image, T is a transformation, and I' is the processed image.

Introduction to OpenCV

Real-World Applications of OpenCV

- Healthcare: Detecting tumors in MRI or analyzing X-ray scans.
- Automotive: Lane detection and pedestrian tracking for self-driving cars.
- Security: Face detection in surveillance systems.
- Robotics: Object recognition and navigation.

Get more info → OpenCV Official Website

Summary: Introduction to OpenCV

Aspect	**Explanation**
Definition	OpenCV is an open-source library for computer vision and image processing
Language Support	Written in C++, with Python, Java, and MATLAB bindings
Core Features	Image/video I/O, transformations, filtering, object detection
Mathematical View	Images represented as matrices, transformations applied as $I' = T(I)$
Applications	Healthcare, automotive, security, robotics

Current Section

1 Data Augmentation for Deep Learning

2 Introduction to OpenCV

3 Loading and Showing Images with OpenCV

4 Converting RGB Images to Grayscale

5 Resizing and Scaling Images

6 Translation

7 Flipping

8 Rotation

Loading and Showing Images with OpenCV

Suppose you are analyzing X-ray images. The first step is to load the scan into memory and display it for the doctor. Without this, no further analysis can be performed.

Concept: Loading and Displaying Images

Definition: OpenCV allows you to load and display images easily.
Image as a matrix:

$$I = \begin{bmatrix} i_{11} & i_{12} & \cdots & i_{1n} \\ i_{21} & i_{22} & \cdots & i_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ i_{m1} & i_{m2} & \cdots & i_{mn} \end{bmatrix}$$

- Grayscale image: $I \in \mathbb{R}^{m \times n}$
- RGB image: $I \in \mathbb{R}^{m \times n \times 3}$

Loading and Showing Images with OpenCV

Read and Display Image in Color and Grayscale

```python
import cv2   # Import the OpenCV library

# ---- Load image in COLOR mode ----
# cv2.IMREAD_COLOR loads image in BGR (default for OpenCV)
img_color = cv2.imread("images.jpeg", cv2.IMREAD_COLOR)

# Display the color image
cv2.imshow("X-Ray Scan - Color", img_color)


# ---- Load image in GRAYSCALE mode ----
# cv2.IMREAD_GRAYSCALE loads the image in single channel
img_gray = cv2.imread("images.jpeg", cv2.IMREAD_GRAYSCALE)

# Display the grayscale image
cv2.imshow("X-Ray Scan - Grayscale", img_gray)

# ---- Wait and close windows ----
# Wait until a key is pressed (0 means indefinitely)
cv2.waitKey(0)

# Destroy all windows opened by OpenCV
cv2.destroyAllWindows()
```

Loading and Showing Images with OpenCV

Numerical Example: Pixel Access

If the image has dimensions 512×512, then the pixel at position $(100, 200)$ might have values:

$$I(100, 200) = (120, 135, 200)$$

- Red intensity = 120
- Green intensity = 135
- Blue intensity = 200

This triplet describes the pixel's color.

Get more info $\rightarrow$ OpenCV Documentation

Summary: Loading and Showing Images with OpenCV

Aspect	**Explanation**
Image Representation	Matrix of pixel intensities, grayscale or RGB
cv2.imread	Loads an image from a file path
cv2.imshow	Opens a window and displays the image
cv2.waitKey	Waits for key press before continuing
cv2.destroyAllWindows	Closes all OpenCV display windows

Current Section

1 Data Augmentation for Deep Learning

2 Introduction to OpenCV

3 Loading and Showing Images with OpenCV

4 Converting RGB Images to Grayscale

5 Resizing and Scaling Images

6 Translation

7 Flipping

8 Rotation

Converting RGB Images to Grayscale

Suppose you are working with surveillance video. Processing in full color is computationally expensive. Often, grayscale is enough for tasks like motion detection or face recognition. How can we convert RGB images to grayscale mathematically and in OpenCV?

Concept: RGB to Grayscale Conversion

Definition: Grayscale is a single-channel image where each pixel represents brightness intensity, instead of color information.
Mathematical formula: A grayscale pixel value Y is computed from the RGB components as a weighted sum:

$$Y = 0.299R + 0.587G + 0.114B$$

- $R, G, B \in [0, 255]$ are the red, green, and blue intensities.
- The weights reflect human visual perception (we see green more strongly, blue less strongly).
- Result $Y \in [0, 255]$ is a single intensity value.

Converting RGB Images to Grayscale

Convert RGB to Grayscale with OpenCV

```python
import cv2   # Import the OpenCV library

# Step 1: Load the original color image
img_color = cv2.imread("images.jpeg", cv2.IMREAD_COLOR)

# Step 2: Convert the color image to grayscale
# cv2.COLOR_BGR2GRAY automatically applies the weighted formula
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

# Step 3: Display both images for comparison
cv2.imshow("Original - Color", img_color)
cv2.imshow("Converted - Grayscale", img_gray)

# Step 4: Wait for a key press and close windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Converting RGB Images to Grayscale

Numerical Example: Pixel Conversion

Consider a pixel with values:

$$R = 120, \quad G = 200, \quad B = 150$$

Applying the formula:

$$Y = 0.299(120) + 0.587(200) + 0.114(150)$$

$$Y = 35.88 + 117.4 + 17.1 = 170.38 \approx 170$$

- Original pixel: $(120, 200, 150)$
- Converted grayscale pixel: 170

Thus the pixel is replaced by a single brightness value.

Get more info → **OpenCV Documentation - Color conversions**

Summary: Converting RGB Images to Grayscale

Aspect	**Explanation**
Purpose	Reduce computation by converting 3-channel RGB to 1-channel grayscale
Formula	$Y = 0.299R + 0.587G + 0.114B$
OpenCV Function	`cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`
Output	Single-channel image where each pixel is intensity only
Example Conversion	$(120, 200, 150) \rightarrow 170$

Current Section

1 Data Augmentation for Deep Learning

2 Introduction to OpenCV

3 Loading and Showing Images with OpenCV

4 Converting RGB Images to Grayscale

5 Resizing and Scaling Images

6 Translation

7 Flipping

8 Rotation

Resizing and Scaling Images

Suppose your model accepts input images of size 224×224. You have an image of size 640×480. You need to resize it without distortion or artifacts. How?

Concept: Resize / Scale with OpenCV

Definition: Resizing changes image width and height. Scaling uses multiplicative factors. OpenCV provides:

```
cv2.resize(src, dsize, fx, fy, interpolation)
```

where:

- `src` = source image
- `dsize` = target size $(\text{width}, \text{height})$
- `fx`, `fy` = scale factors in horizontal and vertical directions
- `interpolation` = method such as `INTER_LINEAR`, `INTER_AREA`, `INTER_CUBIC`

If $\texttt{dsize} = (0, 0)$, then (ignore the fixed size and instead use your scale factors):

$$\texttt{dsize} = (\text{round}(fx \cdot \text{src.cols}), \text{round}(fy \cdot \text{src.rows}))$$

Resizing and Scaling Images

INTER_NEAREST: Nearest Neighbor Resizing

Formula:
For a target pixel at (x', y'), map it back to source coordinates:

$$x = \frac{x'}{s_x}, \quad y = \frac{y'}{s_y}$$

where s_x, s_y are scaling factors.
The new pixel value is taken from the nearest source pixel:

$$I'(x', y') = I(\text{round}(x), \text{round}(y))$$

Explanation:

- Each new pixel copies the value of the closest pixel in the original image.
- No averaging or smoothing is done.
- Fastest method, but produces blocky results when enlarging.

Resizing and Scaling Images

Resize with INTER_NEAREST

```python
import cv2
import numpy as np

# Simple 3x3 grayscale "image" with values 1 to 9
img = np.arange(1, 10, dtype=np.uint8).reshape((3,3))
print("Original 3x3:\n", img)
# [[1 2 3]
#  [4 5 6]
#  [7 8 9]]

# Resize from 3x3 -> 6x6 using INTER_NEAREST
nearest = cv2.resize(img, (6,6), interpolation=cv2.INTER_NEAREST)

print("\nResized 6x6 with INTER_NEAREST:\n", nearest)
# Explanation:
# Each pixel from original is duplicated into larger blocks.
# For example, top-left corner (value 1) expands into a 2x2 block of 1s.
# Result is blocky, but very fast to compute.
# Expected top-left corner of output (first 3 rows, 3 cols):
# [[1 1 2]
#  [1 1 2]
#  [4 4 5]]
# Shows nearest neighbor replication.
```

Resizing and Scaling Images

INTER_LINEAR: Bilinear Interpolation

Formula:

$$I'(x', y') = (1-a)(1-b)Q_{11} + a(1-b)Q_{21} + (1-a)bQ_{12} + abQ_{22}$$

Explanation: - Each new pixel is a weighted average of 4 nearest neighbors. - Produces smoother transitions than nearest neighbor. - May blur sharp edges slightly.

Resize with INTER_LINEAR

```
import cv2
import numpy as np
# Simple 3x3 grayscale "image"
img = np.arange(1, 10, dtype=np.uint8).reshape((3,3))
print("Original 3x3:\n", img)
# Resize 3x3 -> 6x6 using bilinear interpolation
linear = cv2.resize(img, (6,6), interpolation=cv2.INTER_LINEAR)
print("\nResized 6x6 with INTER_LINEAR:\n", linear)
# Expected output:
# [[1 1 2 2 3 3]
#  [2 2 2 3 3 4]
#  [3 3 4 4 5 5]
#  [5 5 5 6 6 7]
#  [6 6 7 7 8 8]
#  [7 7 8 8 9 9]]
```

Resizing and Scaling Images

Interpolation Methods in OpenCV

- **INTER_NEAREST** — Nearest-neighbor interpolation. Each output pixel takes the value of the closest input pixel. Fastest but blocky.
- **INTER_LINEAR** — Bilinear interpolation. Each output pixel is a weighted average of the 4 nearest neighbors. Default in OpenCV.
- **INTER_CUBIC** — Bicubic interpolation over a 4×4 neighborhood (16 pixels). Uses Keys' cubic convolution kernel with $a = -0.5$ (Catmull–Rom spline). Produces smoother results than bilinear but is slower and may overshoot.
- **INTER_LANCZOS4** — Lanczos interpolation over an 8×8 neighborhood using a sinc-based kernel. High-quality for enlargement, sharper details, more computation.
- **INTER_AREA** — Pixel area relation. For shrinking, it computes resampled values by averaging input pixel areas. Produces very good results when reducing image size.

Get more info $\rightarrow$ OpenCV resize documentation – geometric transforms

Summary: Resizing and Scaling Images

Aspect	**Explanation**
Resize vs Scale	Resize uses explicit dimensions, scale uses factors fx, fy
Function	`cv2.resize(src, dsize, fx, fy, interpolation)`
dsize = (0,0)	New size computed from fx, fy and original dimensions
INTER_AREA	Pixel area relation, best for shrinking
INTER_LINEAR	Bilinear interpolation, default, general purpose
INTER_CUBIC	Bicubic interpolation, smoother enlargements, slower

Current Section

1 Data Augmentation for Deep Learning

2 Introduction to OpenCV

3 Loading and Showing Images with OpenCV

4 Converting RGB Images to Grayscale

5 Resizing and Scaling Images

6 Translation

7 Flipping

8 Rotation

Translation

Suppose you train a traffic sign recognition model. A camera mounted on a moving car may capture signs slightly shifted left, right, up, or down. The model should still detect the signs. Translation helps simulate such shifts.

Concept: Image Translation

Definition: Translation shifts every pixel of the image by a displacement vector (t_x, t_y). The affine transformation matrix for translation is:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

So the new coordinates are:

$$x' = x + t_x, \quad y' = y + t_y$$

Properties: - $t_x > 0$ shifts right, $t_x < 0$ shifts left. - $t_y > 0$ shifts down, $t_y < 0$ shifts up. - New empty regions are filled with zeros (black).

Translation

Numerical Example: Translation

Original 3x3 image:

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Shift by $t_x = 1, t_y = 1$:

$$I' = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 4 & 5 \end{bmatrix}$$

Bottom row shifts out of bounds, new top/left entries become 0.

Translation

Translate Image with OpenCV

```python
import cv2
import numpy as np

# Simple 3x3 grayscale matrix
img = np.arange(1, 10, dtype=np.uint8).reshape((3,3))
print("Original:\n", img)

# Translation matrix: shift right 1, down 1
M = np.float32([[1, 0, 1],
                [0, 1, 1]])
translated = cv2.warpAffine(img, M, (3,3))

print("\nTranslated:\n", translated)
# Expected:
# [[0 0 0]
#  [0 1 2]
#  [0 4 5]]
```

Translation

Translate Image with TensorFlow/Keras

```
import tensorflow as tf

# 3x3 grayscale image
img = tf.reshape(tf.range(1, 10, dtype=tf.float32), (1,3,3,1))

# Translate: shift right 1, down 1
translated = tf.keras.preprocessing.image.apply_affine_transform(
    img[0,...,0].numpy(), tx=1, ty=1, fill_mode='constant'
)

print("\nTranslated TensorFlow:\n", translated)
# Similar output, shifted with 0 padding
```

Get more info → OpenCV geometric transforms

Get more info → TensorFlow affine transform

Summary: Translation

Aspect	**Explanation**
Definition	Shift pixels by (t_x, t_y) without altering values
Formula	$x' = x + t_x$, $y' = y + t_y$
OpenCV	Use `warpAffine` with translation matrix
TensorFlow/Keras	Use `apply_affine_transform` with tx, ty
Applications	Data augmentation, robustness to camera shifts

Current Section

1 Data Augmentation for Deep Learning

2 Introduction to OpenCV

3 Loading and Showing Images with OpenCV

4 Converting RGB Images to Grayscale

5 Resizing and Scaling Images

6 Translation

7 Flipping

8 Rotation

Flipping

Suppose you train a face recognition system. A person might face left or right. Your model should recognize the face in both cases. Flipping is used in data augmentation to simulate mirrored views.

Concept: Image Flipping

Definition: Flipping is a reflection of an image across a chosen axis.
- Horizontal flip (mirror left–right)

$$(x', y') = (w - 1 - x,\ y)$$

- Vertical flip (mirror top–bottom)

$$(x', y') = (x,\ h - 1 - y)$$

- Both axes (180° rotation)

$$(x', y') = (w - 1 - x,\ h - 1 - y)$$

Here w, h are image width and height.
Properties: - Preserves distances and shapes. - Does not alter pixel values, only repositions them.
- Used to increase data diversity.

Flipping

Numerical Example: Flipping

Original 3x3 image:

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Horizontal flip:

$$I' = \begin{bmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \\ 9 & 8 & 7 \end{bmatrix}$$

Vertical flip:

$$I'' = \begin{bmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix}$$

Flipping

Flip Image with OpenCV

```python
import cv2
import numpy as np

img = np.arange(1, 10, dtype=np.uint8).reshape((3,3))
print("Original:\n", img)

# Horizontal flip
flipped_h = cv2.flip(img, 1)

# Vertical flip
flipped_v = cv2.flip(img, 0)

# Both axes
flipped_b = cv2.flip(img, -1)

print("\nHorizontal Flip:\n", flipped_h)
print("\nVertical Flip:\n", flipped_v)
print("\nBoth Axes Flip:\n", flipped_b)
```

Flipping

Flip Image with TensorFlow/Keras

```
import tensorflow as tf

# 3x3 grayscale image
img = tf.reshape(tf.range(1, 10, dtype=tf.float32), (1,3,3,1))

# Flip horizontally
flip_h = tf.image.flip_left_right(img)

# Flip vertically
flip_v = tf.image.flip_up_down(img)

print("\nHorizontal Flip TensorFlow:\n", flip_h.numpy()[0,:,:,0])
print("\nVertical Flip TensorFlow:\n", flip_v.numpy()[0,:,:,0])
```

Get more info → OpenCV flip documentation

Get more info → TensorFlow image flipping

Summary: Flipping

Aspect	**Explanation**
Definition	Reflects image across chosen axis (horizontal, vertical, or both)
Horizontal Flip	$(x', y') = (w - 1 - x, y)$
Vertical Flip	$(x', y') = (x, h - 1 - y)$
Both Axes	Equivalent to 180° rotation
OpenCV	Use `cv2.flip(image, flipCode)`
TensorFlow/Keras	Use `tf.image.flip_left_right` or `tf.image.flip_up_down`

Current Section

1 Data Augmentation for Deep Learning

2 Introduction to OpenCV

3 Loading and Showing Images with OpenCV

4 Converting RGB Images to Grayscale

5 Resizing and Scaling Images

6 Translation

7 Flipping

8 Rotation

Rotation

Suppose you train a handwritten digit recognizer (MNIST). A digit “6” rotated slightly might still appear in real-world data. To make the model robust, you apply image rotation during preprocessing.

Concept: Image Rotation

Definition: Rotation turns an image around a point (usually its center) by an angle θ.
The rotation matrix (around origin) is:

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Applied to a point (x, y):

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R(\theta) \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

For rotation around the image center (c_x, c_y), we adjust by translation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R(\theta) \cdot \begin{bmatrix} x - c_x \\ y - c_y \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix}$$

Rotation

Numerical Example: Rotation 90°

Original 3x3 image:

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Rotate 90° counter-clockwise around center:

$$I' = \begin{bmatrix} 3 & 6 & 9 \\ 2 & 5 & 8 \\ 1 & 4 & 7 \end{bmatrix}$$

Rotation

Rotate Image with OpenCV

```python
import cv2
import numpy as np

img = np.arange(1, 10, dtype=np.uint8).reshape((3,3))
print("Original:\n", img)

# Get rotation matrix: rotate 90 degrees around the center
(h, w) = img.shape
M = cv2.getRotationMatrix2D((w/2, h/2), 90, 1)
rotated = cv2.warpAffine(img, M, (w,h))

print("\nRotated 90  :\n", rotated)
# Expected:
# [[3 6 9]
#  [2 5 8]
#  [1 4 7]]
```

Rotation

Rotate Image with TensorFlow/Keras

```python
import tensorflow as tf

# 3x3 grayscale image
img = tf.reshape(tf.range(1, 10, dtype=tf.float32), (1,3,3,1))

# Rotate 90   counter-clockwise
rotated = tf.image.rot90(img, k=1)

print("\nRotated TensorFlow 90  :\n", rotated.numpy()[0,:,:,0])
```

Get more info → OpenCV rotation documentation

Get more info → TensorFlow rotation function

Summary: Rotation

Aspect	**Explanation**
Definition	Turn image around a point by angle θ
Rotation Matrix	$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$
Center Rotation	Translate to center, apply $R(\theta)$, translate back
OpenCV	Use `getRotationMatrix2D` + `warpAffine`
TensorFlow/Keras	Use `tf.image.rot90` for multiples of 90°
Applications	Data augmentation, robustness to orientation changes