OpenCVとPythonによる画像拡大・縮小技術：詳細解説と実践的コード例

はじめに (Introduction)

デジタル画像処理およびコンピュータビジョンの分野において、画像の拡大・縮小（スケーリングまたはリサイズ）は最も基本的な操作の一つです。この操作は、画像の寸法を変更するプロセスを指し、ピクセル数を増やす（拡大、アップスケーリング）または減らす（縮小、ダウンスケーリング）ことによって行われます [1]。画像スケーリングの主な目的は、特定の表示要件への適合、機械学習モデルへの入力データの正規化、処理速度の最適化、ストレージ容量の削減など、多岐にわたります [2]。
一般的なユースケースとしては、Webページレイアウトへの画像埋め込み、物体検出や顔認識などのコンピュータビジョンタスクの前処理、機械学習モデル（特に畳み込みニューラルネットワークなど）で要求される固定入力サイズへの画像統一などが挙げられます [2]。特に機械学習の分野では、モデルが学習するデータセット内の画像サイズを統一することで、学習効率と精度が向上し、また画像サイズを小さくすることで学習や推論の速度向上、メモリ使用量の削減に繋がります [2]。
このレポートでは、強力かつ広く利用されているオープンソースのコンピュータビジョンライブラリであるOpenCVと、汎用プログラミング言語Pythonを用いて、画像の拡大・縮小処理を実装する方法について詳細に解説します。OpenCVは、画像リサイズのための効率的で多様な機能を提供しており [4]、Pythonはその簡潔な文法と豊富なライブラリ連携により、OpenCVを用いた画像処理タスクを容易に実現可能にします [8]。本レポートを通じて、cv2.resize関数の基本的な使い方から、様々な補間アルゴリズムの選択、アスペクト比の維持といった実践的なテクニックまでを網羅的に解説し、具体的なコード例を提供します。

OpenCVにおける画像リサイズ関数 cv2.resize (The cv2.resize Function in OpenCV)

OpenCVライブラリにおいて、画像のサイズ変更を担当する主要な関数は cv2.resize です [2]。この関数は非常に柔軟性が高く、出力画像の絶対的な寸法を指定する方法と、元の画像に対する相対的なスケーリング係数を指定する方法の両方に対応しています [4]。

関数の概要と基本的なシンタックス (Function Overview and Basic Syntax)

cv2.resize 関数は、入力画像 (src) を受け取り、指定された目標サイズ (dsize) またはスケーリング係数 (fx, fy) に基づいてリサイズされた出力画像 (dst) を生成します。リサイズ処理中にピクセル値をどのように計算するかを指定するために、補間方法 (interpolation) を選択することも可能です [4]。基本的なシンタックスは以下の通りです [4]:

Python

dst = cv2.resize(src, dsize[, fx[, fy[, interpolation]]])

ここで、dst はリサイズ後の出力画像（NumPy配列）です [10]。

主要なパラメータ解説 (Explanation of Key Parameters)

cv2.resize 関数の挙動を制御する主要なパラメータは以下の通りです。

- **src (入力画像):** リサイズ対象の元画像です。通常、cv2.imread() 関数で読み込まれたNumPy配列形式の画像データを指定します [4]。
- **dsize (目標サイズ):** 出力画像の目標サイズを (幅, 高さ) のタプル形式で指定します [4]。例えば、幅300ピクセル、高さ200ピクセルにリサイズしたい場合は (300, 200) と指定します [11]。このパラメータが (0, 0) または None でない場合、fx と fy は無視され、dsize から自動的に計算されます [4]。OpenCVでは画像の形状（shape）が (高さ, 幅, チャンネル数) の順で表されるのに対し、dsize は (幅, 高さ) の順で指定する点に注意が必要です [1]。
- **fx, fy (スケーリング係数):** それぞれ水平方向（幅）と垂直方向（高さ）のスケーリング係数を指定します [4]。例えば、画像を元の半分のサイズにする場合は fx=0.5, fy=0.5 と指定します [2]。dsize が (0, 0) または None の場合に有効となり、これらの係数から出力画像のサイズが dsize = (round(fx * src.cols), round(fy * src.rows)) のように計算されます [4]。fx または fy のどちらか一方だけを指定することはできません（両方指定するか、dsize を指定する必要があります）。ただし、dsize が None で fx と fy が指定されている場合、これらが優先されます [11]。fx と fy に同じ値を指定すると、画像の縦横比（アスペクト比）が維持されます [2]。
- **interpolation (補間方法):** 画像を拡大または縮小する際に、新しいピクセル値を計算するためのアルゴリズムを指定します [4]。補間方法の選択は、リサイズ後の画質と処理速度に影響を与えます。指定しない場合のデフォルトは cv2.INTER_LINEAR です [11]。利用可能な補間方法については次節で詳しく解説します。

これらのパラメータを適切に設定することで、様々な要件に応じた画像リサイズが可能になります。特に、dsize を用いた絶対的なサイズ指定と、fx, fy を用いた相対的なスケーリング指定を使い分けることが重要です。

補間アルゴリズムの比較と選択 (Comparison and Selection of Interpolation Algorithms)

画像のリサイズ処理、特にピクセル数を増やす拡大や、ピクセル数を減らす縮小においては、元のピクセル情報から新しいピクセル値をどのように計算するかが重要となります。この計算方法を補間アルゴリズムと呼びます。OpenCVの cv2.resize 関数では、interpolation パラメータを通じて複数の補間アルゴリズムを選択できます [2]。

利用可能な補間方法の紹介 (Introduction to Available Methods)

OpenCVで一般的に利用される主な補間方法は以下の通りです [2]:

- **cv2.INTER_NEAREST (最近傍補間):** 最も単純かつ高速な方法です。出力画像の各ピクセル位置に最も近い入力画像のピクセル値をそのまま使用します [10]。計算コストは低いですが、特に画像を拡大した場合にブロックノイズ（ピクセルが目立つ状態）が発生しやすい傾向があります [11]。
- **cv2.INTER_LINEAR (バイリニア補間、線形補間):** デフォルトの補間方法です [11]。出力画像の各ピクセルに対し、近傍の 2x2 = 4 ピクセルの値を用いて線形補間を行い、新しいピクセル値を計算します [12]。INTER_NEAREST より滑らかな結果が得られ、計算速度と品質のバランスが取れています [11]。画像の拡大に適しています [13]。
- **cv2.INTER_CUBIC (バイキュービック補間、3次補間):** 近傍の 4x4 = 16 ピクセルの値を用いて3次多項式で補間計算を行います [12]。INTER_LINEAR よりも計算量は増えますが、より高品質で滑らかな結果が得られ、特に画像の拡大に適しています [1]。ただし、処理速度は遅くなります [13]。
- **cv2.INTER_AREA (ピクセル領域の関係に基づいたリサンプリング):** ピクセル領域の関係性を利用してリサンプリングを行います [12]。画像の縮小（シュリンク）に最も適した方法とされています [1]。縮小時には、元の画像のピクセル領域の情報を効果的に利用してアンチエイリアス効果（ギザギザの軽減）が得られ、モアレの発生を抑制しやすいです。拡大に使用された場合は、INTER_NEAREST と同様の結果になります [12]。
- **cv2.INTER_LANCZOS4 (ランチョス補間):** 近傍の 8x8 = 64 ピクセルの値を用いて補間計算を行います [11]。INTER_CUBIC よりもさらに多くのピクセル情報を利用するため、一般的に最も高品質な結果が得られますが、計算コストも最も高くなります [2]。鮮明さを保ちつつアーティファクトを最小限に抑えたい場合に有効です。

各アルゴリズムの特性比較 (Comparison of Characteristics)

以下の表は、各補間アルゴリズムの主な特性をまとめたものです。

補間方法 (interpolation)	計算コスト	出力品質 (主観)	主な用途	特徴
cv2.INTER_NEAREST	最低	低	高速処理、単純な拡大・縮小	ブロックノイズが発生しやすい [11]
cv2.INTER_LINEAR	低	中	デフォルト、拡大、速度と品質のバランス [11]	2x2近傍ピクセルを使用 [12]
cv2.INTER_CUBIC	中	高	高品質な拡大 [1]	4x4近傍ピクセルを使用、INTER_LINEARより滑らかだが低速 [12]
cv2.INTER_AREA	可変	縮小時に高	画像の縮小に最適	ピクセル領域の関

			[1]	係性を利用、モアレ抑制効果 [12]。拡大時は INTER_NEAREST相当 [12]
cv2.INTER_LANCZOS4	高	最高	最高品質の拡大・縮小、鮮明さの維持	8x8近傍ピクセルを使用、計算負荷が大きい [2]

補間アルゴリズムの選択は、最終的な画像の品質と処理速度の間のトレードオフを考慮して行う必要があります。例えば、リアルタイム処理が要求されるアプリケーションでは INTER_LINEAR や INTER_NEAREST が適している場合がありますが、オフラインでの高品質な画像生成が目的であれば INTER_CUBIC や INTER_LANCZOS4 を選択することが考えられます。

拡大・縮小に適した補間方法 (Suitable Methods for Upscaling and Downscaling)

経験則として、以下の選択が推奨されています [1]:

- 画像を縮小する場合 **(Downscaling/Shrinking):** cv2.INTER_AREA を使用することが一般的に最適です。ピクセル情報を適切に平均化・集約することで、エイリアシング（折り返し雑音）やモアレを効果的に抑制し、鮮明さを保ちながら自然な縮小結果を得られます [1]。
- 画像を拡大する場合 **(Upscaling/Zooming):**
    - 品質重視**:** cv2.INTER_CUBIC または cv2.INTER_LANCZOS4 を使用します。これらの方法はより多くの近傍ピクセル情報を考慮するため、滑らかでアーティファクトの少ない高品質な拡大結果が得られます [1]。ただし、計算コストは高くなります。
    - 速度重視**:** cv2.INTER_LINEAR を使用します。INTER_CUBIC や INTER_LANCZOS4 に比べて品質は若干劣るものの、多くの場合で許容範囲内の結果をより高速に得ることができます [11]。デフォルトの補間方法でもあるため、手軽に利用できます。

cv2.INTER_NEAREST は、速度が最優先される場合や、ピクセルアートのような特定のスタイルを維持したい場合に限定的に使用されることがあります。

実践的なコード例 (Practical Code Examples)

ここでは、PythonとOpenCV (cv2) ライブラリを用いて、実際に画像をリサイズする具体的なコード例を示します。

準備：ライブラリのインポートと画像の読み込み (Preparation: Importing Libraries and Loading Images)

まず、必要なライブラリ（OpenCVと、画像情報の取得などに便利なNumPy）をインポートし、リサイズ対象の画像を読み込みます。

Python

```
import cv2
import numpy as np
import os # ファイル存在確認のため (オプション)

# 画像ファイルのパス
image_path = 'input.jpg' # ここに対象の画像ファイル名を入力してください

# 画像を読み込む
img = cv2.imread(image_path)

# 画像が正しく読み込めたかを確認 (推奨)
if img is None:
    print(f"Error: 画像ファイルが見つからないか、読み込めません。パスを確認してください: {image_path}")
    # 必要に応じてここでプログラムを終了するなどの処理を追加
    exit()
else:
    # 元の画像の寸法（高さ, 幅, チャンネル数）を表示
    print(f"Original Dimensions (H, W, C): {img.shape}")
```

上記のコードでは、cv2.imread() で画像を読み込み、img 変数に格納しています [3]。読み込みに失敗した場合（ファイルが存在しない、形式が非対応など）は img が None になるため、エラーチェックを行うことが推奨されます [13]。img.shape は画像の寸法を (高さ, 幅, チャンネル数) のタプルで返します [1]。カラー画像の場合はチャンネル数が3 (BGR)、グレースケール画像の場合はチャンネル数が省略されるか1となります [16]。

特定の寸法 (dsize) を指定したリサイズ (Resizing to Specific Dimensions (dsize))

出力画像の幅と高さをピクセル単位で直接指定してリサイズする方法です。cv2.resize 関数の第二引数 dsize に (幅, 高さ) のタプルを指定します [4]。

Python

```
# (上記の準備コードに続けて実行)

# 新しい寸法を指定 (例: 幅=500ピクセル, 高さ=400ピクセル)
new_width = 500
new_height = 400
```

```
dsize = (new_width, new_height) # (幅, 高さ) の順で指定

# 画像をリサイズ (補間方法はデフォルトの INTER_LINEAR を使用)
resized_img_dsize = cv2.resize(img, dsize, interpolation=cv2.INTER_LINEAR) # interpolation は省略可能

# リサイズ後の画像の寸法を表示
print(f"Resized Dimensions (dsize): {resized_img_dsize.shape}")

# --- 結果の表示・保存 (オプション) ---
# cv2.imshow("Original Image", img)
# cv2.imshow("Resized Image (dsize)", resized_img_dsize)
# cv2.waitKey(0)
# cv2.destroyAllWindows()
# cv2.imwrite('resized_dsize.jpg', resized_img_dsize)
```

注意点: この方法を用いる際、指定した dsize のアスペクト比（幅と高さの比率）が元の画像のアスペクト比と異なる場合、リサイズ後の画像は歪んで表示されます（縦長または横長に引き伸ばされたり、押し潰されたりします）[1]。歪みを避けたい場合は、元の画像のアスペクト比を維持するように dsize を計算するか、次に説明するスケーリング係数を使用する方法が適しています [4]。アスペクト比を維持しつつ特定の幅または高さに合わせるには、以下のように計算します。

Python

```
# アスペクト比を維持して特定の幅にリサイズする例
target_width = 300
original_height, original_width = img.shape[:2]
aspect_ratio = original_height / original_width
target_height = int(target_width * aspect_ratio)
dsize_aspect_preserved = (target_width, target_height)
resized_aspect_preserved = cv2.resize(img, dsize_aspect_preserved,
interpolation=cv2.INTER_AREA) # 縮小を想定
print(f"Resized Dimensions (Aspect Preserved, Width={target_width}):
{resized_aspect_preserved.shape}")
```

スケーリング係数 **(fx, fy)** を指定したリサイズ **(Resizing Using Scaling Factors (fx, fy))**

元の画像の幅と高さに対する倍率（スケールファクター）を指定してリサイズする方法です。dsize パラメータを None または (0, 0) に設定し、fx (水平方向の倍率) と fy (垂直方向の倍率) を指定します [2]。

Python

```
# (上記の準備コードに続けて実行)

# スケーリング係数を指定 (例: 縦横ともに0.75倍に縮小)
scale_percent_x = 0.75 # 水平方向(幅)の倍率
scale_percent_y = 0.75 # 垂直方向(高さ)の倍率

# 画像をリサイズ (dsize=None を指定)
resized_img_fx_fy = cv2.resize(img, None, fx=scale_percent_x, fy=scale_percent_y,
interpolation=cv2.INTER_AREA) # 縮小なので INTER_AREA を推奨

# リサイズ後の画像の寸法を表示
print(f"Resized Dimensions (fx, fy): {resized_img_fx_fy.shape}")

# --- 結果の表示・保存 (オプション) ---
# cv2.imshow("Original Image", img)
# cv2.imshow("Resized Image (fx, fy)", resized_img_fx_fy)
# cv2.waitKey(0)
# cv2.destroyAllWindows()
# cv2.imwrite('resized_fx_fy.jpg', resized_img_fx_fy)
```

この方法の大きな利点は、fx と fy に同じ値を設定するだけで、自動的に元画像のアスペクト比が維持される点です [2]。dsize を使用する場合のように、アスペクト比を維持するために別途計算を行う必要がありません。そのため、「画像を半分にしたい」「2倍に拡大したい」といった相対的なサイズ変更を行いたい場合に非常に直感的で便利です。元の画像の正確な寸法を事前に知らなくても、あるいは気にする必要がない場合にも容易に適用できます。これは、バッチ処理などで様々なサイズの画像を一定の比率でスケーリングする際に特に有効です。

推奨される補間方法の適用例 (Applying Recommended Interpolation Methods)

前述の通り、縮小と拡大では推奨される補間方法が異なります。以下にそれぞれの適用例を示します。

Python

```
# (上記の準備コードに続けて実行)
```

```
# --- 縮小する場合 (Downscaling) ---
# 画像を元のサイズの30%に縮小 (アスペクト比維持)
scale_factor_down = 0.3
resized_down = cv2.resize(img, None, fx=scale_factor_down, fy=scale_factor_down,
interpolation=cv2.INTER_AREA) # 縮小には INTER_AREA
print(f"Resized Down (INTER_AREA): {resized_down.shape}")
# cv2.imshow("Resized Down (INTER_AREA)", resized_down)


# --- 拡大する場合 (Upscaling) ---
# 画像を元のサイズの1.5倍に拡大 (アスペクト比維持)
scale_factor_up = 1.5

# 高品質な拡大 (INTER_CUBIC)
resized_up_cubic = cv2.resize(img, None, fx=scale_factor_up, fy=scale_factor_up,
interpolation=cv2.INTER_CUBIC) # 高品質拡大には INTER_CUBIC
print(f"Resized Up (INTER_CUBIC): {resized_up_cubic.shape}")
# cv2.imshow("Resized Up (INTER_CUBIC)", resized_up_cubic)

# 速度重視の拡大 (INTER_LINEAR)
resized_up_linear = cv2.resize(img, None, fx=scale_factor_up, fy=scale_factor_up,
interpolation=cv2.INTER_LINEAR) # 速度重視なら INTER_LINEAR
print(f"Resized Up (INTER_LINEAR): {resized_up_linear.shape}")
# cv2.imshow("Resized Up (INTER_LINEAR)", resized_up_linear)


# --- 表示・終了処理 (オプション) ---
# cv2.waitKey(0) # 何かキーが押されるまで待機
# cv2.destroyAllWindows() # 開いている全てのウィンドウを閉じる
```

これらの例のように、リサイズの目的（縮小か拡大か）と要求される品質・速度に応じて interpolation パラメータを適切に選択することが重要です。

リサイズ後の画像の表示と保存 (Displaying and Saving Resized Images)

リサイズ処理を行った結果を確認したり、ファイルとして保存したりするには、以下のOpenCV関数を使用します。

- **表示:** cv2.imshow('Window Name', image_variable) を使用して、指定した名前のウィンドウに画像を表示します [3]。
- **待機:** cv2.waitKey(0) は、キーボードからの入力があるまでプログラムの実行を一時停止し、ウィンドウを表示し続けます。引数をミリ秒で指定すると、その時間だけ表示します [3]。
- **ウィンドウ破棄:** cv2.destroyAllWindows() は、cv2.imshow で開かれたすべてのウィンドウを

閉じます [3]。スクリプト終了時に呼び出すのが一般的です。
- 保存: cv2.imwrite('output_filename.jpg', image_variable) を使用して、指定したファイル名（拡張子で形式が決まる）で画像をファイルシステムに保存します [1]。

Python

```
# (リサイズ処理後のコード例)

# リサイズした画像を表示
cv2.imshow("Resized Image", resized_down) # 例として縮小画像を表示

# キー入力を待つ
key = cv2.waitKey(0)

# 's' キーが押されたら画像を保存する例
if key == ord('s'):
    output_filename = 'resized_output.png' # 保存ファイル名 (PNG形式)
    cv2.imwrite(output_filename, resized_down)
    print(f"画像を {output_filename} として保存しました。")

# すべてのウィンドウを閉じる
cv2.destroyAllWindows()
```

これらの関数を組み合わせることで、リサイズ処理の結果を柔軟に扱えます。

まとめとベストプラクティス (Conclusion and Best Practices)

主要なテクニックの要約 (Summary of Key Techniques)

本レポートでは、OpenCVとPythonを用いた画像リサイズ技術について解説しました。主要なポイントは以下の通りです。

- 画像リサイズは、OpenCVの cv2.resize 関数を用いて容易に実装できます [3]。
- リサイズ方法として、出力画像の絶対的な寸法 (dsize) を指定する方法と、相対的なスケーリング係数 (fx, fy) を指定する方法があります [4]。
- 補間アルゴリズム (interpolation パラメータ) の選択は、リサイズ後の画質と処理速度に大きく影響します。主な選択肢として INTER_NEAREST, INTER_LINEAR, INTER_CUBIC, INTER_AREA, INTER_LANCZOS4 があります [2]。

ベストプラクティスと考慮事項 (Best Practices and Considerations)

効果的かつ高品質な画像リサイズを行うためには、以下のベストプラクティスと考慮事項が重要です。

- アスペクト比の維持: 画像の歪みを防ぐためには、可能な限り元画像のアスペクト比を維持することが推奨されます。fx と fy に同じ値を指定するか [2]、dsize を指定する場合は元のアスペクト比に基づいて計算する必要があります [1]。
- 適切な補間方法の選択:
    - 縮小には、アンチエイリアス効果が高くモアレを抑制しやすい cv2.INTER_AREA を使用します [1]。
    - 拡大には、品質を最優先するなら cv2.INTER_CUBIC または cv2.INTER_LANCZOS4 を、速度とのバランスを取るなら cv2.INTER_LINEAR (デフォルト) を選択します [1]。
- コードの明確性: リサイズの目的（絶対的な目標サイズか、相対的なスケール変更か）に応じて、dsize を使うか fx/fy を使うかを明確に選択し、コードの意図を分かりやすくします。
- 画像品質の確認: リサイズ処理は、特に拡大時に情報の損失やアーティファクト（偽のパターンやノイズ）を発生させる可能性があります [1]。処理後の画像を目視で確認し、意図した品質が得られているか、予期せぬ問題が発生していないかを検証することが重要です。
- バッチ処理: 複数の画像に対して同じリサイズ処理を適用する場合は、Pythonの os モジュールなどを用いてディレクトリ内のファイルを反復処理するスクリプトを作成すると、作業を効率化できます [1]。
- **cv2.resize** の限界: cv2.resize は基本的に、画像全体のピクセルを一様に幾何学的に変形する操作です。これは、画像内のコンテンツ（写っているオブジェクトの意味や重要度）を考慮しないことを意味します。そのため、単純なリサイズによって重要な特徴が失われたり、不自然な変形が生じたりする可能性があります。例えば、画像内の小さな文字が縮小によって読めなくなったり、顔の一部が拡大によって不自然に引き伸ばされたりすることが考えられます。より高度な要求、例えば画像内の重要なオブジェクトのサイズや形状を保護しつつ背景領域を調整するような「コンテンツに応じたリサイズ」（Content-aware resizing）や、AIを用いた超解像技術などが必要な場合は、cv2.resize 以外のアルゴリズム（例：Seam Carving、ただしOpenCVの標準機能ではない）や、深層学習ベースの手法を検討する必要があります [1]。

これらの点を考慮することで、OpenCVとPythonを用いた画像リサイズをより効果的に活用し、様々なアプリケーションの要求に応えることが可能になります。

引用文献

1. Python Image Resize With Pillow and OpenCV - Cloudinary, 4月 15, 2025にアクセス、https://cloudinary.com/guides/bulk-image-resize/python-image-resize-with-pillow-and-opencv
2. What is Image Resizing? A Computer Vision Guide. - Roboflow Blog, 4月 15, 2025にアクセス、https://blog.roboflow.com/image-resizing/
3. Image Resizing using OpenCV in Python - Analytics Vidhya, 4月 15, 2025にアクセス、https://www.analyticsvidhya.com/blog/2024/01/image-resizing-using-opencv-in-

python/
4. Resizing and Rescaling Images with OpenCV, 4月 15, 2025にアクセス、https://opencv.org/blog/resizing-and-rescaling-images-with-opencv/
5. リアルタイムビデオ処理への AI の適用: 基本とその他 - Unite.AI, 4月 15, 2025にアクセス、https://www.unite.ai/ja/AI-%E3%82%92%E3%83%AA%E3%82%A2%E3%83%AB%E3%82%BF%E3%82%A4%E3%83%A0%E3%83%93%E3%83%87%E3%82%AA%E5%87%A6%E7%90%86%E3%81%AB%E5%BF%9C%E7%94%A8%E3%81%99%E3%82%8B%E5%9F%BA%E7%A4%8E%E3%81%AA%E3%81%A9/
6. YoloV5使ったフレーム画像のオブジェクト検出チュートリアル - 株式会社ひけしや, 4月 15, 2025にアクセス、https://hikesiya.co.jp/kawaraban/machine-learning-yolov5
7. OpenCV CV2 Resize Image (cv2.resize) - PyImageSearch, 4月 15, 2025にアクセス、https://pyimagesearch.com/2021/01/20/opencv-resize-image-cv2-resize/
8. 【Pythonで画像加工】OpenCVの基本を解説！〜 プログラミング初心者 向け 〜 - YouTube, 4月 15, 2025にアクセス、https://m.youtube.com/watch?v=SNTkI7aYzYE&pp=ygUTI-eUu-WDj-iqjeitmHB5dGhvbg%3D%3D
9. 【たったの7行！】Pythonで画像を自動で一括リサイズ！ - YouTube, 4月 15, 2025にアクセス、https://www.youtube.com/watch?v=hBUcbvz1m9g
10. OpenCV cv2.resize() Function - Scaler Topics, 4月 15, 2025にアクセス、https://www.scaler.com/topics/cv2-resize/
11. Effective Image Resizing with cv2 Resize Image in Python - Python Central, 4月 15, 2025にアクセス、https://www.pythoncentral.io/effective-image-resizing-with-cv2-resize-image-in-python/
12. Image Resizing with OpenCV | LearnOpenCV #, 4月 15, 2025にアクセス、https://learnopencv.com/image-resizing-with-opencv/
13. Geometric Transformations of Images - OpenCV Documentation, 4月 15, 2025にアクセス、https://docs.opencv.org/4.x/da/d6e/tutorial_py_geometric_transformations.html
14. Geometric Image Transformations - OpenCV Documentation, 4月 15, 2025にアクセス、https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html
15. How to use cv2.resize (v4.2.0) - Stack Overflow, 4月 15, 2025にアクセス、https://stackoverflow.com/questions/61089969/how-to-use-cv2-resize-v4-2-0
16. Basic Operations on Images - OpenCV Documentation, 4月 15, 2025にアクセス、https://docs.opencv.org/4.x/d3/df2/tutorial_py_basic_ops.html
17. OpenCVでリサイズ・モザイク・顔検出、ときどき象印 - Qiita, 4月 15, 2025にアクセス、https://qiita.com/mo256man/items/c570c645153cae122196
18. 【09.基礎4】OpenCVを学ぶ - クラゲのIoTテクノロジー, 4月 15, 2025にアクセス、https://jellyware.jp/kurage/movidius/c09_opencv.html
19. OpenCV-python:リサイズする方法（cv2.resize) - Qiita, 4月 15, 2025にアクセス、https://qiita.com/JarvisSan22/items/e335c9c814948e06d1b3
20. Geometric Transformations of Images - OpenCV Documentation, 4月 15, 2025にアクセス、https://docs.opencv.org/3.4/da/d6e/tutorial_py_geometric_transformations.html

21. 【Python・OpenCV】画像の拡大・縮小をマスターする(cv2.resize) - codevace, 4月 15, 2025にアクセス、https://www.codevace.com/py-opencv-resize/
22. How to resize an image with OpenCV2.0 and Python2.6 - Stack Overflow, 4月 15, 2025にアクセス、https://stackoverflow.com/questions/4195453/how-to-resize-an-image-with-opencv2-0-and-python2-6
23. 【Pythonで画像処理をはじめよう】OpenCVの使い方を解説 - Udemy メディア, 4月 15, 2025にアクセス、https://udemy.benesse.co.jp/development/python-work/opencv.html
24. Resize an image without distortion OpenCV - python - Stack Overflow, 4月 15, 2025にアクセス、https://stackoverflow.com/questions/44650888/resize-an-image-without-distortion-opencv
25. OpenCVで画像サイズの変更をしてみた - Qiita, 4月 15, 2025にアクセス、https://qiita.com/kenfukaya/items/dfa548309c301c7087c4
26. 【Python/OpenCV】バイリニア補間法で画像の拡大・縮小（cv2.resize）, 4月 15, 2025にアクセス、https://python.joho.info/opencv/opencv-resize-bilinear-interpolation-py/
27. 【Python/OpenCV】cv2.resizeの最近傍補間法(cv2.INTER_NEAREST)で画像の拡大, 4月 15, 2025にアクセス、https://python.joho.info/opencv/opencv-resize-nearest-interpolation-py/