```
import cv2
import os
import numpy as np
from tensorflow.keras.datasets import cifar10
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

(train_images, train_labels), (_, _) = cifar10.load_data()

fixed_image_folder = "cifar10_fixed_images"
os.makedirs(fixed_image_folder, exist_ok=True)

moving_image_folder = "cifar10_moving_images"
os.makedirs(moving_image_folder, exist_ok=True)

def create_image_pair(image, fixed_image_path, moving_image_path):
    cv2.imwrite(fixed_image_path, cv2.cvtColor(image,
cv2.COLOR_RGB2BGR))

    rows, cols, _ = image.shape
    rotation_matrix = cv2.getRotationMatrix2D((cols / 2, rows / 2),
180, 1)
    rotated_image = cv2.warpAffine(image, rotation_matrix, (cols,
rows))

    cv2.imwrite(moving_image_path, cv2.cvtColor(rotated_image,
cv2.COLOR_RGB2BGR))

for i, image in enumerate(train_images):
    fixed_image_path = os.path.join(fixed_image_folder,
f"fixed_image_{i}.jpg")
    moving_image_path = os.path.join(moving_image_folder,
f"moving_image_{i}.jpg")
    create_image_pair(image, fixed_image_path, moving_image_path)

fixed_images = [cv2.imread(os.path.join(fixed_image_folder,
f"fixed_image_{i}.jpg"), cv2.IMREAD_GRAYSCALE) for i in
range(len(train_images))]
moving_images = [cv2.imread(os.path.join(moving_image_folder,
f"moving_image_{i}.jpg"), cv2.IMREAD_GRAYSCALE) for i in
range(len(train_images))]
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-
python.tar.gz
170498071/170498071 [==============================] - 4s 0us/step
```

```
# Prepare data for training
fixed_images = np.array(fixed_images) / 255.0
moving_images = np.array(moving_images) / 255.0
```

```
print(fixed_images.shape)
print(moving_images.shape)
```

```
(50000, 32, 32)
(50000, 32, 32)
```

```
import os
import numpy as np
import cv2
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt

# Function to load images from a folder
def load_images(folder, num_samples=None):
    images = []
    filenames = sorted(os.listdir(folder))[:num_samples] if
num_samples else sorted(os.listdir(folder))
    for filename in filenames:
        img = cv2.imread(os.path.join(folder, filename))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        img = cv2.resize(img, (256, 256))
        images.append(img)
    return np.array(images)

# Load fixed and moved images
fixed_images = load_images("/content/cifar10_fixed_images")
moved_images = load_images("/content/cifar10_moving_images")

# Combine fixed and moved images
all_images = np.concatenate((fixed_images, moved_images), axis=0)

# Create labels for translations and rotation
labels = np.zeros((len(all_images), 3))
labels[len(fixed_images):, 0] = 256 / 2
labels[len(fixed_images):, 1] = 256 / 2
labels[len(fixed_images):, 2] = 180

# Shuffle the data
random_index = np.random.permutation(len(all_images))
all_images = all_images[random_index]
labels = labels[random_index]

# Save the fixed images to a folder
save_folder = "/content/fixed_images"
os.makedirs(save_folder, exist_ok=True)
for i, img in enumerate(fixed_images):
    cv2.imwrite(os.path.join(save_folder, f"fixed_image_{i}.jpg"),
img)
```

```
# Save the fixed images to a folder
save_folder = "/content/moved_images"
os.makedirs(save_folder, exist_ok=True)
for i, img in enumerate(moved_images):
    cv2.imwrite(os.path.join(save_folder, f"moved_image_{i}.jpg"),
img)

# Create a CNN model for image registration
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(256,
256, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(256, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(3)  # Output layer with 3 units for x and y
translations and rotation angle
])

model.compile(optimizer='adam', loss='mse')  # Mean Squared Error loss
for regression

# Train the model
model.fit(np.expand_dims(all_images, axis=-1), labels, epochs=5,
batch_size=32)

# Predict translations and rotation for moved images
predictions = model.predict(np.expand_dims(moved_images, axis=-1))
```

```
Epoch 1/5
3125/3125 [==============================] - 129s 39ms/step - loss:
250.9030
Epoch 2/5
3125/3125 [==============================] - 124s 40ms/step - loss:
42.6192
Epoch 3/5
3125/3125 [==============================] - 124s 40ms/step - loss:
34.0700
Epoch 4/5
3125/3125 [==============================] - 123s 39ms/step - loss:
25.7433
Epoch 5/5
3125/3125 [==============================] - 122s 39ms/step - loss:
21.1949
1563/1563 [==============================] - 22s 14ms/step
```

```
# Create a folder to save aligned images if it doesn't exist
save_folder = "/content/aligned_images"
os.makedirs(save_folder, exist_ok=True)

# Apply translations and rotation to align moved images with fixed
images
aligned_images = []
for i in range(len(moved_images)):
    translation_x, translation_y, rotation_angle = predictions[i]
    M = cv2.getRotationMatrix2D((256 / 2, 256 / 2), rotation_angle, 1)
# Get rotation matrix
    M[:, 2] += [translation_x, translation_y]  # Apply translation to
the rotation matrix
    aligned_img = cv2.warpAffine(moved_images[i], M, (512, 512))  #
Apply transformation with larger output size
    aligned_images.append(aligned_img)
    # Save aligned image
    cv2.imwrite(os.path.join(save_folder, f"aligned_image_{i}.jpg"),
aligned_img)

# Save the model
model.save("/content/image_registration_model.h5")
```

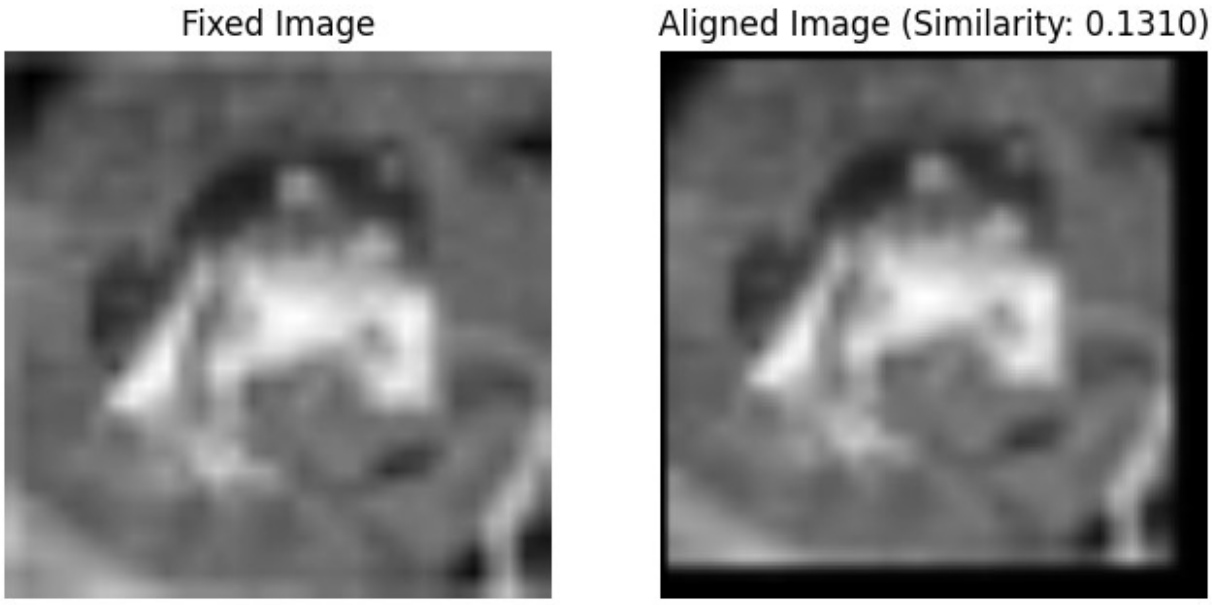
```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/
training.py:3103: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

```
import os
import numpy as np
from skimage.io import imread
from skimage.transform import resize
from skimage.metrics import structural_similarity as ssim
import matplotlib.pyplot as plt

def load_and_prepare_images(folder, target_shape, num_samples=10):
    images = []
    filenames = sorted(os.listdir(folder))[:num_samples]
    for filename in filenames:
        img = imread(os.path.join(folder, filename))
        # Resize image
        resized_img = resize(img, target_shape, anti_aliasing=True)
        images.append(resized_img)
    return images

def evaluate_alignment(fixed_images, aligned_images):
    similarities = []
    for fixed_img, aligned_img in zip(fixed_images, aligned_images):
```

```
        similarity = ssim(fixed_img, aligned_img)
        similarities.append(similarity)
    return similarities

def crop_image(img):
    # Get coordinates of non-black pixels
    coords = np.argwhere(img != 0)
    # Get bounding box of non-black region
    x0, y0 = coords.min(axis=0)
    x1, y1 = coords.max(axis=0) + 1
    # Crop image to bounding box
    cropped_img = img[x0:x1, y0:y1]
    return cropped_img

# Define folder paths
fixed_images_folder = "/content/fixed_images"
aligned_images_folder = "/content/aligned_images"

# Define target shape for resizing
target_shape = (256, 256)

# Load and prepare images for the first 10 samples
fixed_images = load_and_prepare_images(fixed_images_folder,
target_shape)
aligned_images = load_and_prepare_images(aligned_images_folder,
target_shape)

# Evaluate alignment using SSIM
similarities = evaluate_alignment(fixed_images, aligned_images)

# Print similarity scores and display images
for i, (fixed_img, aligned_img, similarity) in
enumerate(zip(fixed_images, aligned_images, similarities), start=1):
    print(f"Similarity for image {i}: {similarity}")

    # Display fixed image
    plt.figure(figsize=(8, 4))
    plt.subplot(1, 2, 1)
    plt.imshow(fixed_img, cmap='gray')
    plt.title('Fixed Image')
    plt.axis('off')

    # Crop aligned image to remove black background
    cropped_aligned_img = crop_image(aligned_img)

    # Display cropped aligned image
    plt.subplot(1, 2, 2)
    plt.imshow(cropped_aligned_img, cmap='gray')
    plt.title(f'Aligned Image (Similarity: {similarity:.4f})')
```

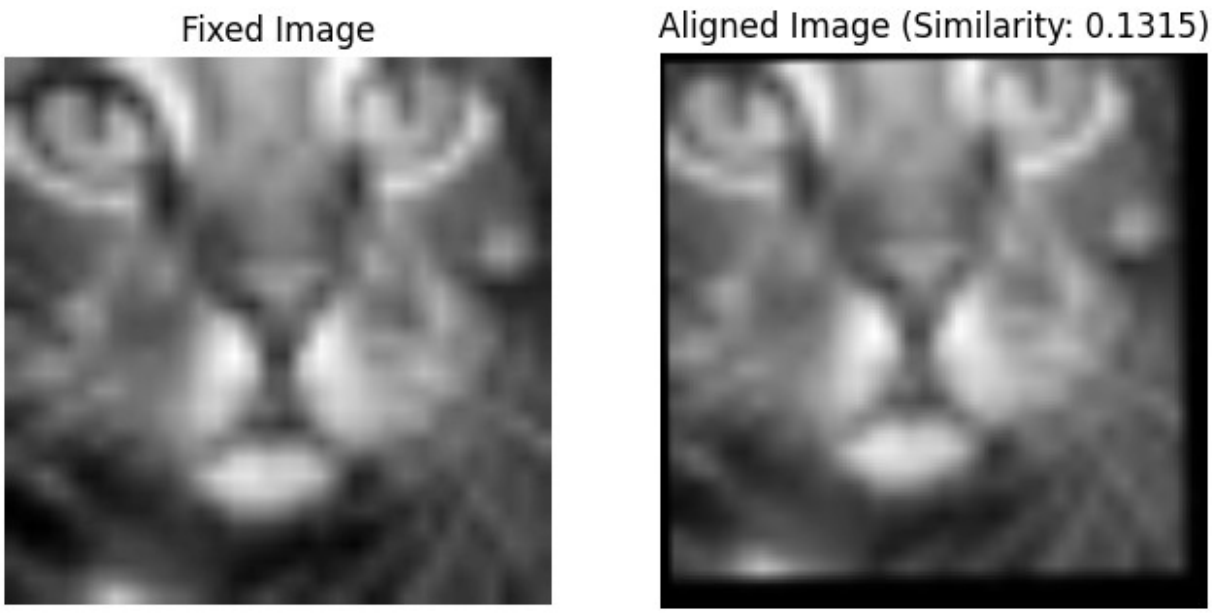
```
    plt.axis('off')
    plt.show()
```

```
Similarity for image 1: 0.13095753324101694
```

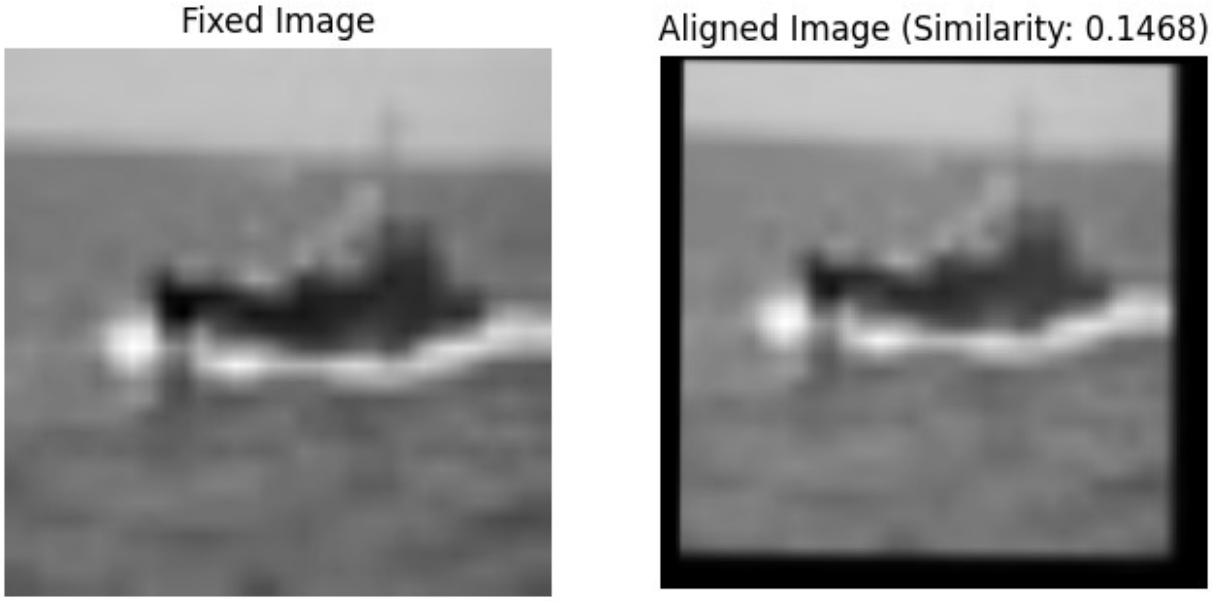



```
Similarity for image 2: 0.09576537355053705
```

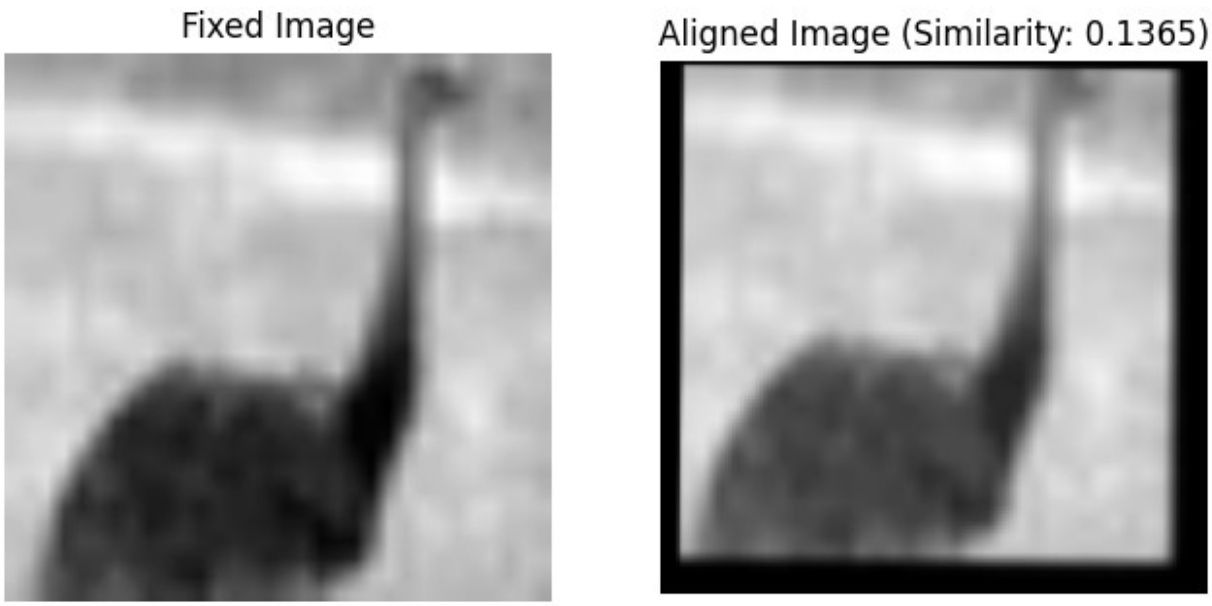

```
Similarity for image 3: 0.13148609154876115
```



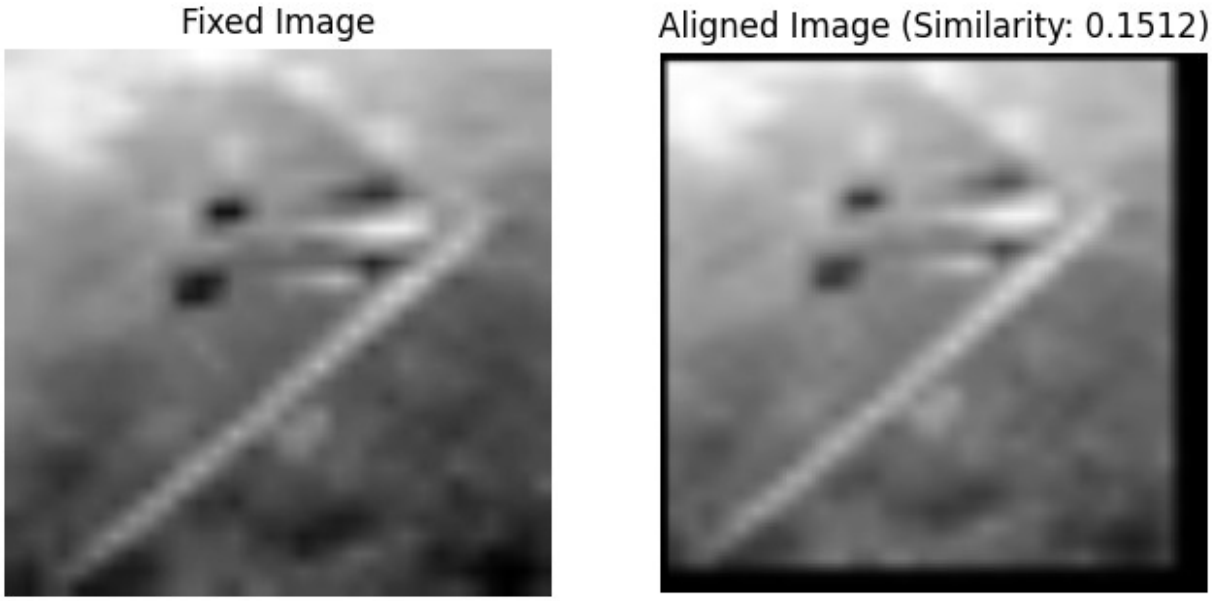

```
Similarity for image 4: 0.1467853858022824
```



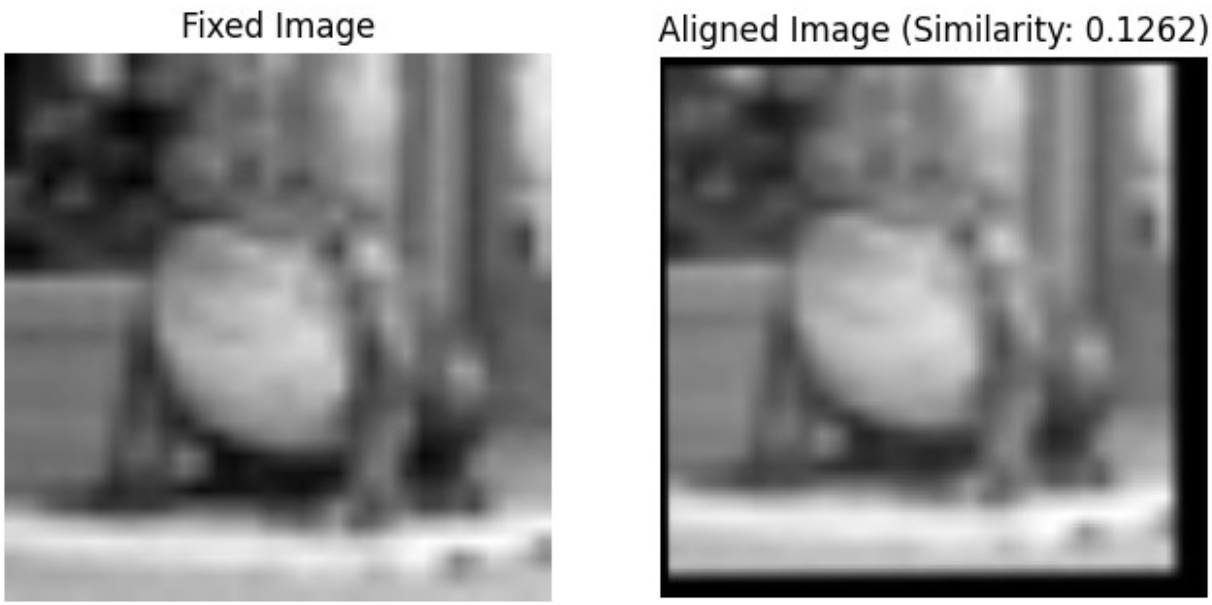

```
Similarity for image 5: 0.136459695765713
```



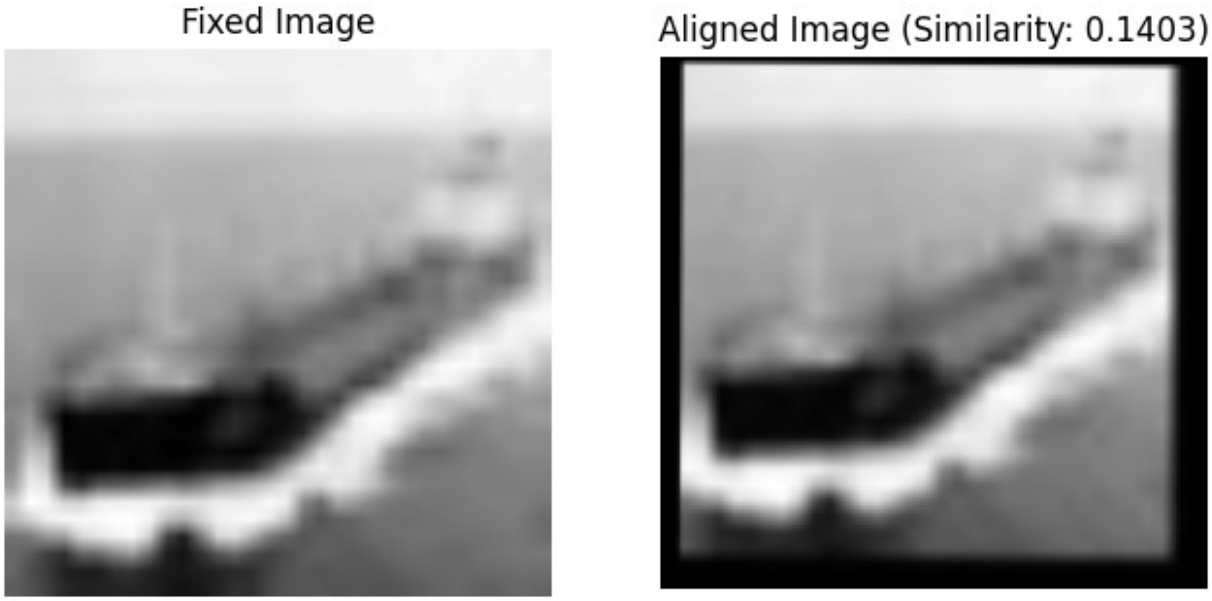

```
Similarity for image 6: 0.15120833440921289
```



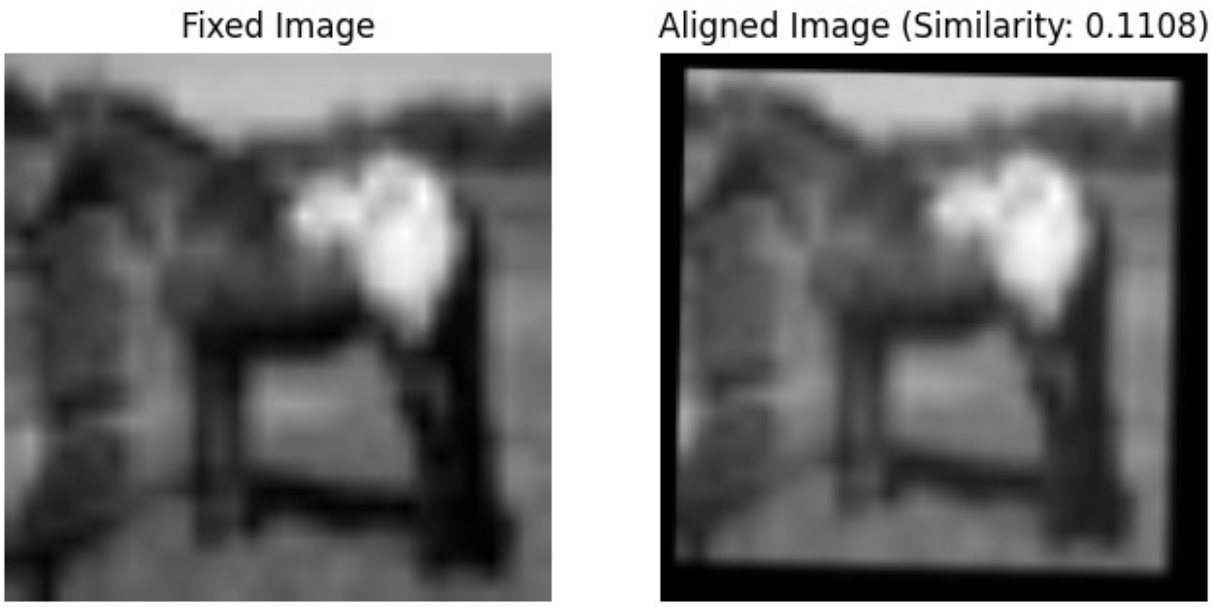

```
Similarity for image 7: 0.12623001431741598
```



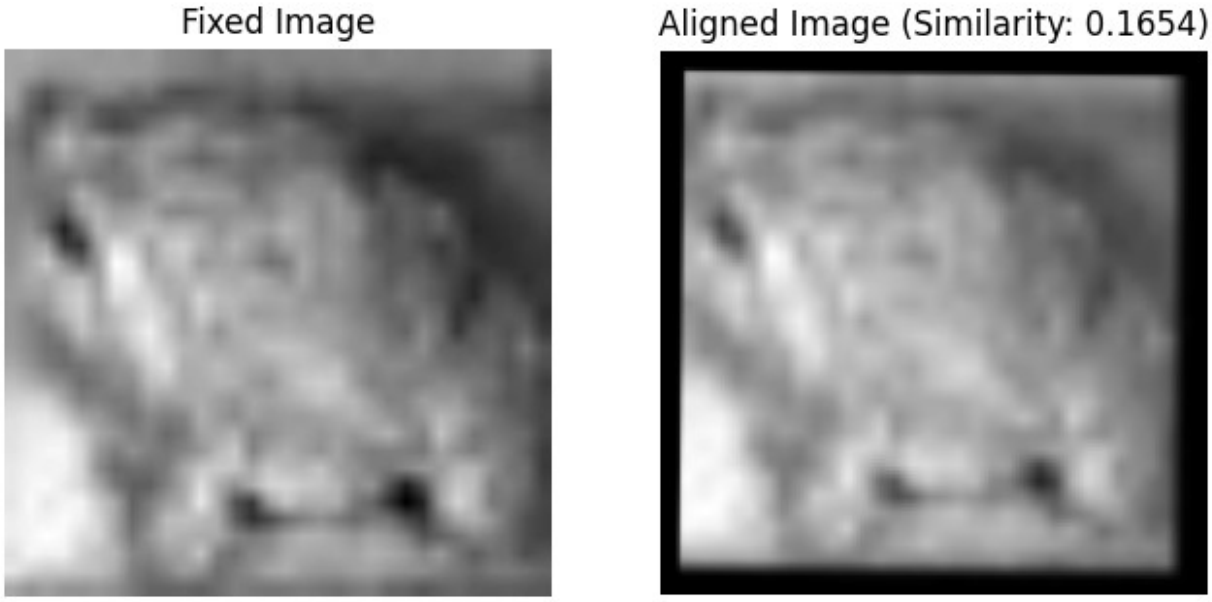

```
Similarity for image 8: 0.1403053815213098
```




```
Similarity for image 9: 0.11078208958096666
```




```
Similarity for image 10: 0.1654300060524377
```




```
import os
import numpy as np
from skimage.io import imread
from skimage.transform import resize
import matplotlib.pyplot as plt

def load_and_prepare_images(folder, target_shape, num_samples=10):
    images = []
    filenames = sorted(os.listdir(folder))[:num_samples]
```

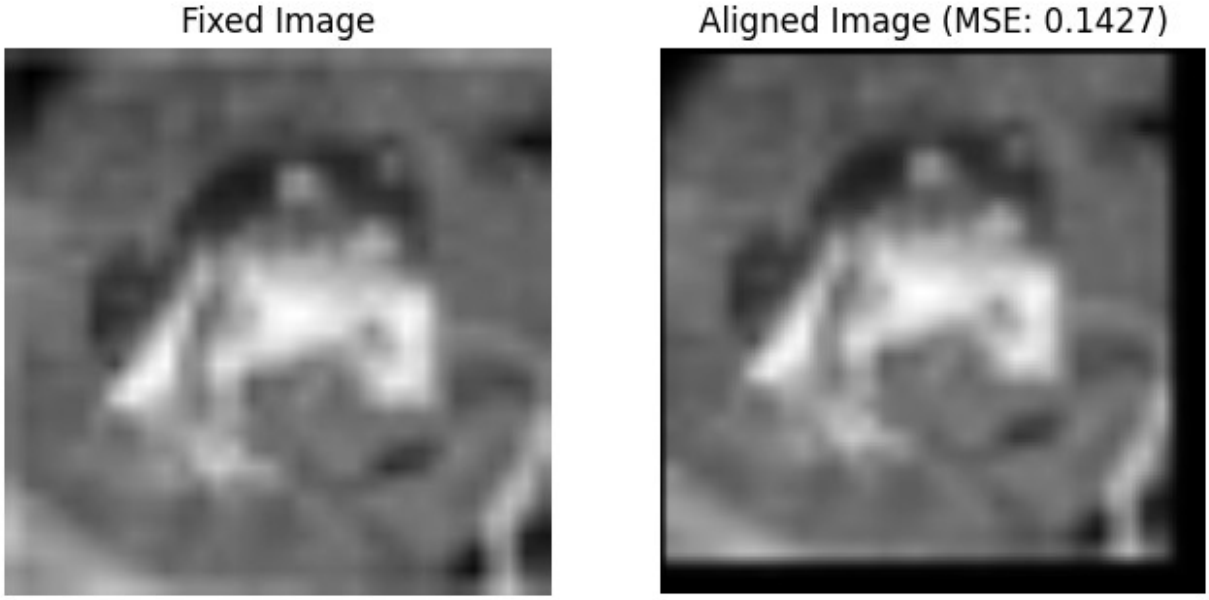
```
    for filename in filenames:
        img = imread(os.path.join(folder, filename))
        # Resize image
        resized_img = resize(img, target_shape, anti_aliasing=True)
        images.append(resized_img)
    return images

def calculate_mse(image1, image2):
    return np.mean((image1 - image2) ** 2)

# Define folder paths
fixed_images_folder = "/content/fixed_images"
aligned_images_folder = "/content/aligned_images"

# Define target shape for resizing
target_shape = (256, 256)

# Load and prepare images for the first 10 samples
fixed_images = load_and_prepare_images(fixed_images_folder,
target_shape)
aligned_images = load_and_prepare_images(aligned_images_folder,
target_shape)

# Calculate MSE similarity
similarities = []
for fixed_img, aligned_img in zip(fixed_images, aligned_images):
    mse = calculate_mse(fixed_img, aligned_img)
    similarities.append(mse)

# Print similarity scores
for i, similarity in enumerate(similarities, start=1):
    print(f"Similarity for image {i}: {similarity}")

# Optionally, calculate the average similarity score
average_similarity = np.mean(similarities)
print("Average similarity:", average_similarity)
```

```
Similarity for image 1: 0.14265190500385905
Similarity for image 2: 0.22561889087811457
Similarity for image 3: 0.12119851906996837
Similarity for image 4: 0.2458878929127758
Similarity for image 5: 0.3597774060902889
Similarity for image 6: 0.24070269589699972
Similarity for image 7: 0.21137740857364012
Similarity for image 8: 0.350642398953217
Similarity for image 9: 0.15300223792783846
Similarity for image 10: 0.19129248551013836
Average similarity: 0.2242151840816841
```

```python
import os
import numpy as np
from skimage.io import imread
from skimage.transform import resize
from skimage.metrics import structural_similarity as ssim
import matplotlib.pyplot as plt

def load_and_prepare_images(folder, target_shape, num_samples=10):
    images = []
    filenames = sorted(os.listdir(folder))[:num_samples]
    for filename in filenames:
        img = imread(os.path.join(folder, filename))
        # Resize image
        resized_img = resize(img, target_shape, anti_aliasing=True)
        images.append(resized_img)
    return images

def crop_image(img):
    # Get coordinates of non-black pixels
    coords = np.argwhere(img != 0)
    # Get bounding box of non-black region
    x0, y0 = coords.min(axis=0)
    x1, y1 = coords.max(axis=0) + 1
    # Crop image to bounding box
    cropped_img = img[x0:x1, y0:y1]
    return cropped_img

def calculate_mse(image1, image2):
    return np.mean((image1 - image2) ** 2)

# Define folder paths
fixed_images_folder = "/content/fixed_images"
aligned_images_folder = "/content/aligned_images"

# Define target shape for resizing
target_shape = (256, 256)

# Load and prepare images for the first 10 samples
fixed_images = load_and_prepare_images(fixed_images_folder,
target_shape)
aligned_images = load_and_prepare_images(aligned_images_folder,
target_shape)

# Calculate MSE similarity
similarities = []
for fixed_img, aligned_img in zip(fixed_images, aligned_images):
    mse = calculate_mse(fixed_img, aligned_img)
    similarities.append(mse)

# Print MSE values and display images
```

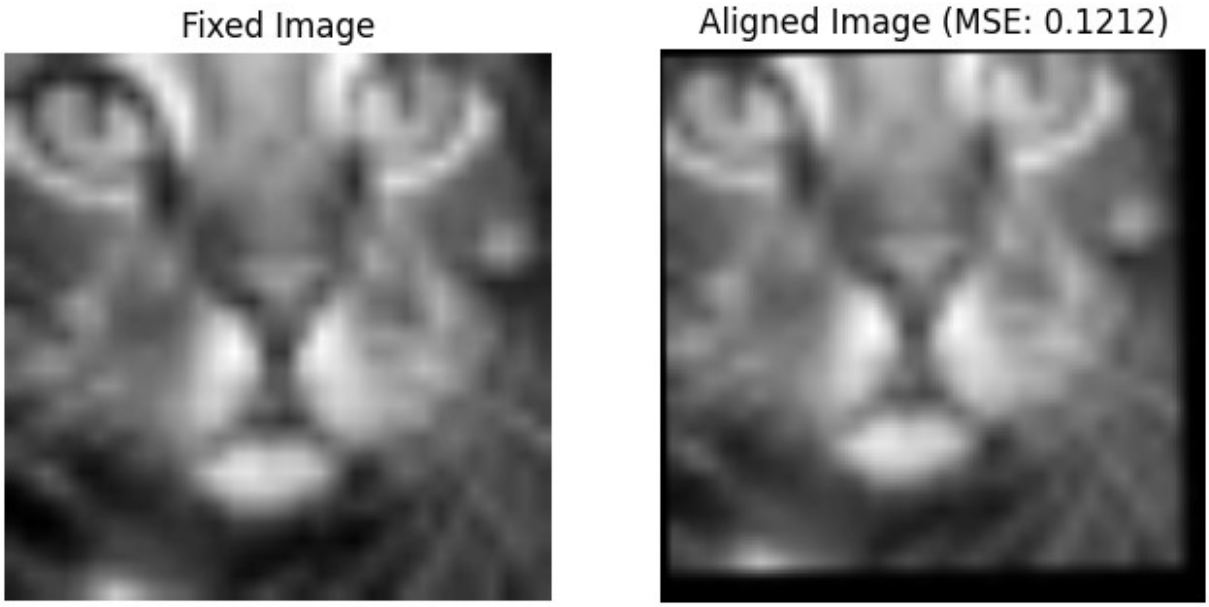
```
for i, (fixed_img, aligned_img, similarity) in
enumerate(zip(fixed_images, aligned_images, similarities), start=1):
    print(f"MSE for image {i}: {similarity}")

    # Display fixed image
    plt.figure(figsize=(8, 4))
    plt.subplot(1, 2, 1)
    plt.imshow(fixed_img, cmap='gray')
    plt.title('Fixed Image')
    plt.axis('off')

    # Crop aligned image to remove black background
    cropped_aligned_img = crop_image(aligned_img)

    # Display cropped aligned image with zoom
    plt.subplot(1, 2, 2)
    plt.imshow(cropped_aligned_img, cmap='gray')
    plt.title(f'Aligned Image (MSE: {similarity:.4f})')
    plt.axis('off')
    plt.xlim(0, cropped_aligned_img.shape[1])  # Set x-axis limits to
zoom in
    plt.ylim(cropped_aligned_img.shape[0], 0)  # Set y-axis limits to
invert the image
    plt.gca().set_aspect('equal', adjustable='box')  # Maintain aspect
ratio
    plt.show()
```

```
MSE for image 1: 0.14265190500385905
```

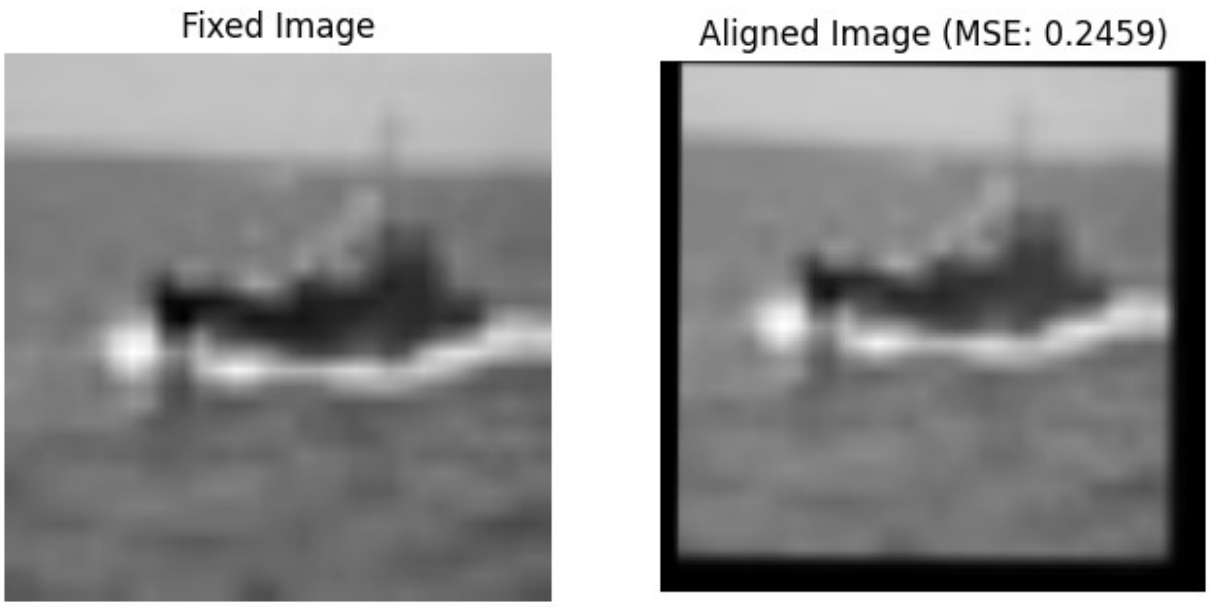



```
MSE for image 2: 0.22561889087811457
```

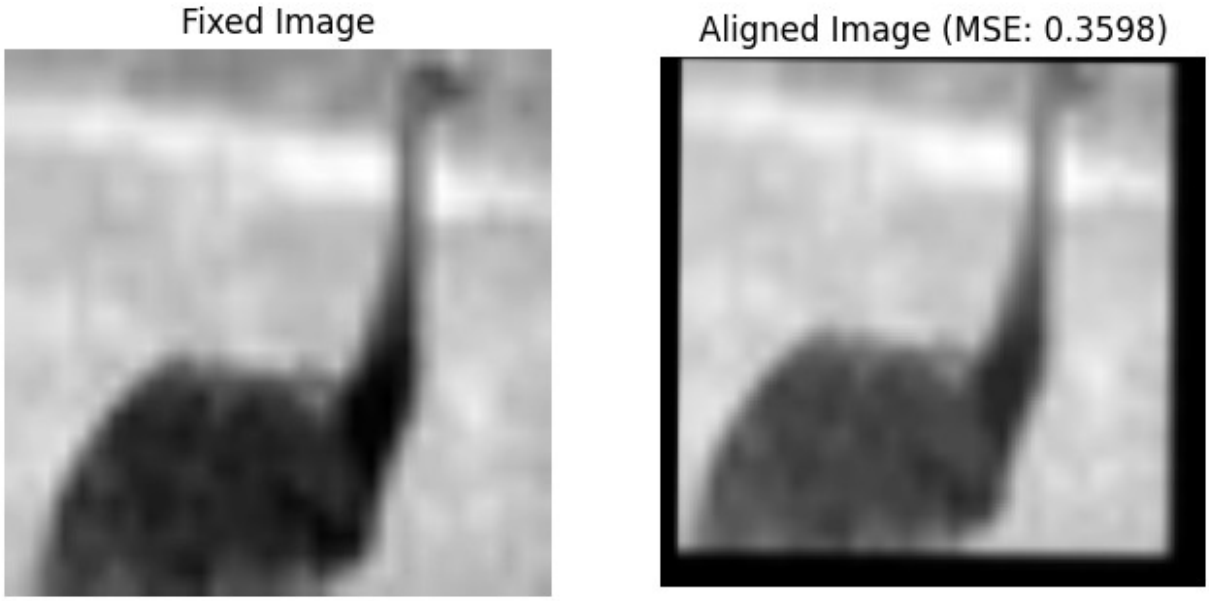

```
MSE for image 3: 0.12119851906996837
```



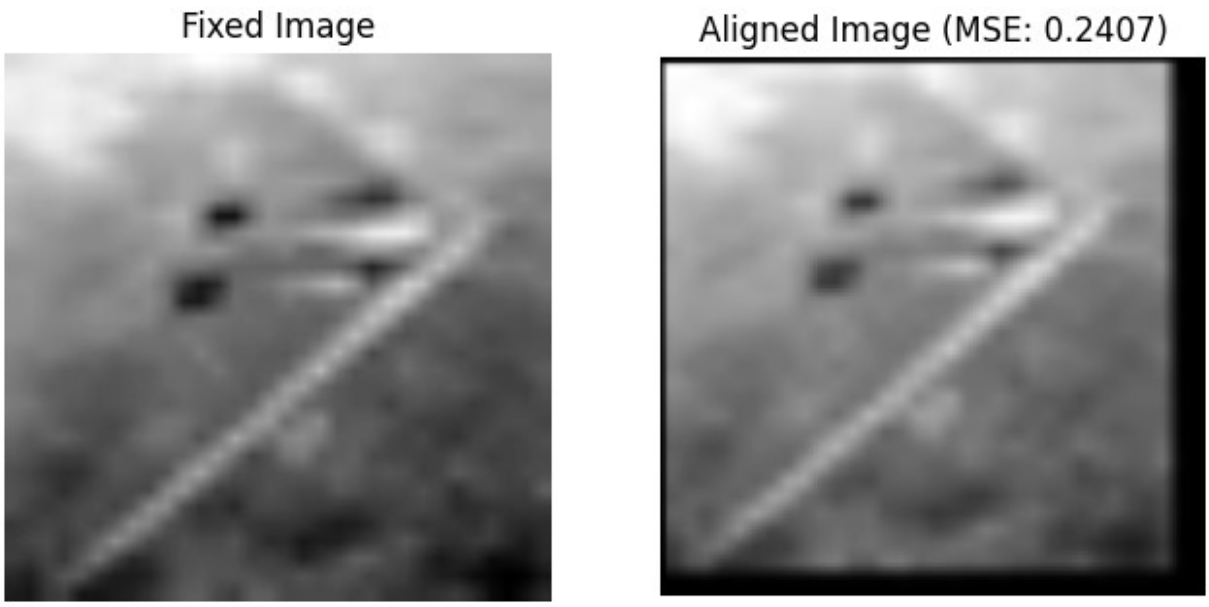

```
MSE for image 4: 0.2458878929127758
```



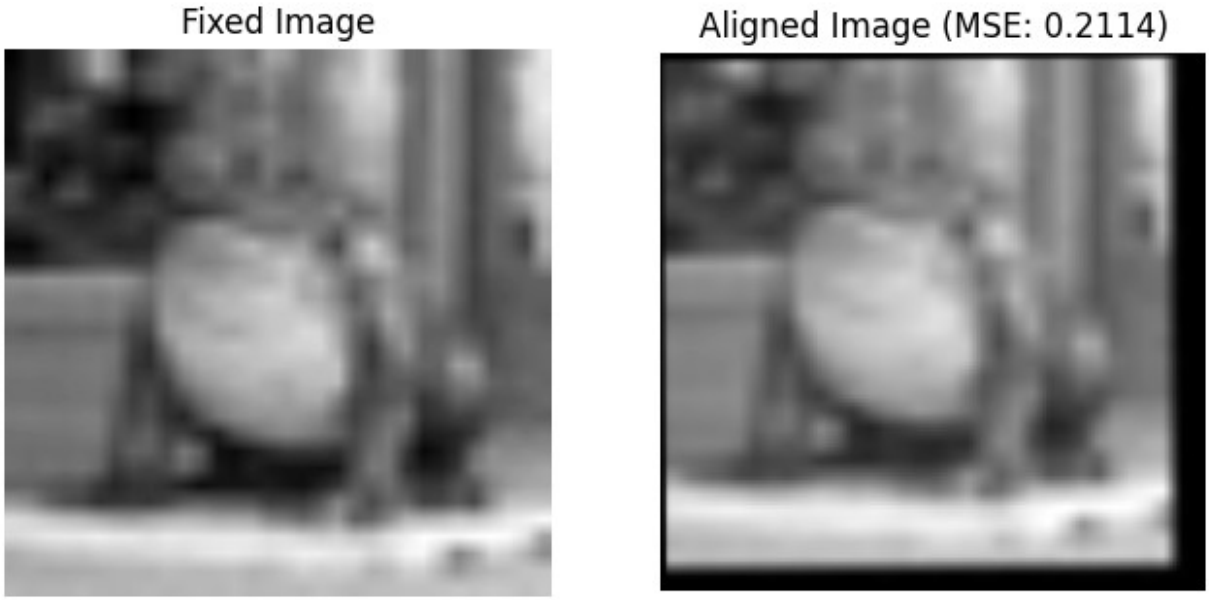

```
MSE for image 5: 0.3597774060902889
```



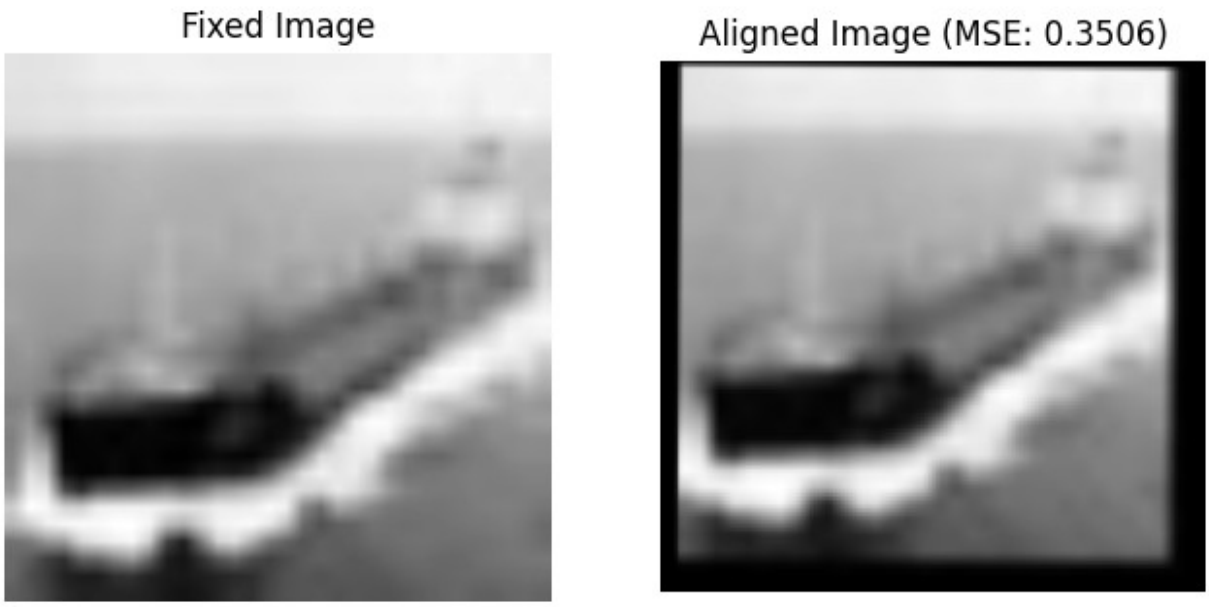

```
MSE for image 6: 0.24070269589699972
```



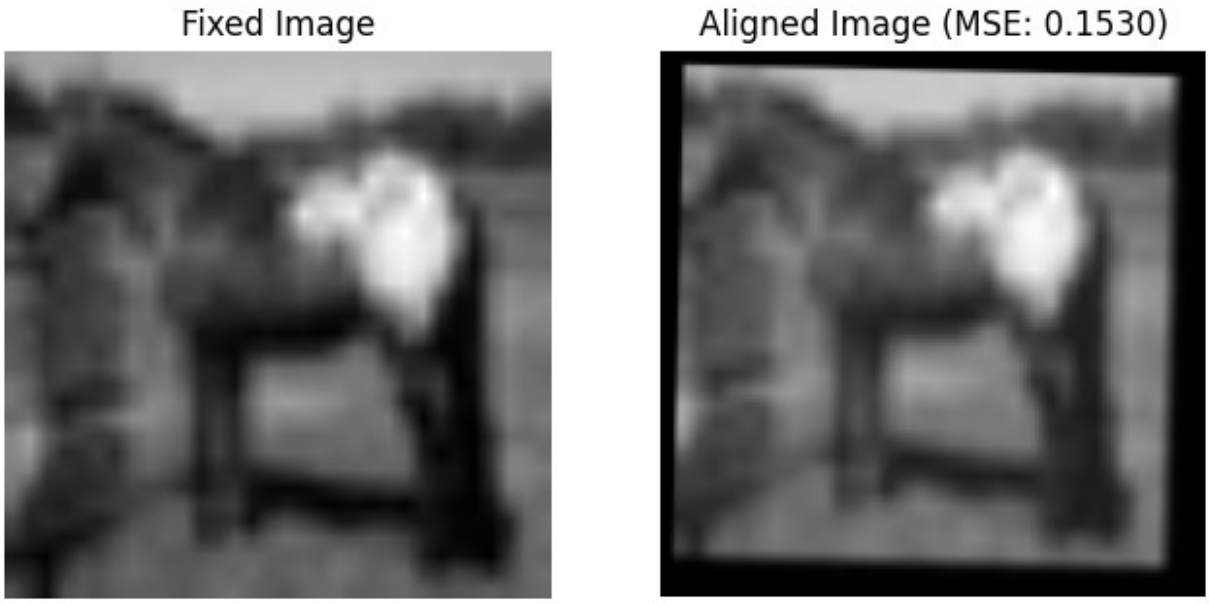

```
MSE for image 7: 0.21137740857364012
```




```
MSE for image 8: 0.350642398953217
```




```
MSE for image 9: 0.15300223792783846
```




```
MSE for image 10: 0.19129248551013836
```

Fixed Image

Aligned Image (MSE: 0.1913)