Imports and Preprocessing

In [2]:
```
try:
    import tensorflow as tf
    import numpy as np
    import pandas as pd
    import os
    import cv2 as cv
    import matplotlib.pyplot as plt
    from tensorflow.keras.layers import add, Conv2D, Input
    from tensorflow.keras.models import Model
    from tensorflow.keras.optimizers import Adam
except ImportError:
    !pip install tensorflow opencv-python matplotlib pandas h5py
    import tensorflow as tf
    import numpy as np
    import pandas as pd
    import os
    import cv2 as cv
    import matplotlib.pyplot as plt
    from tensorflow.keras.layers import add, Conv2D, Input
    from tensorflow.keras.models import Model
    from tensorflow.keras.optimizers import Adam

tf.random.set_seed(1)
np.random.seed(1)

print("Environment ready for image enhancement.")
```

```
Environment ready for image enhancement.
```

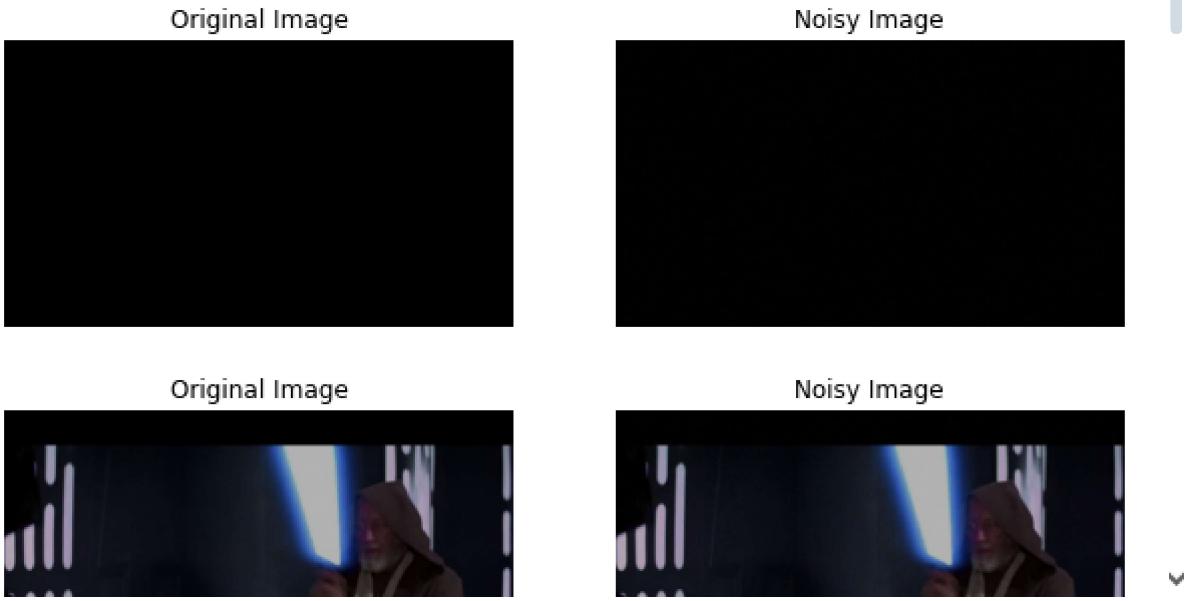
Noise Addition Function

In [3]:
```python
def add_noise(image, noise_type="s&p"):
    if noise_type == "gauss":
        mean, var = 0, 0.0001
        sigma = var ** 0.5
        gauss = np.random.normal(mean, sigma, image.shape).astype(np.float32)
        noisy = np.clip(image + gauss, 0, 1)
        return noisy
    elif noise_type == "s&p":
        prob = 0.02
        noisy = np.copy(image)
        black = np.random.rand(*image.shape) < prob / 2
        white = np.random.rand(*image.shape) < prob / 2
        noisy[black] = 0
        noisy[white] = 1
        return noisy
    return image
```

In [37]:
```
import os
import zipfile
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

zip_file_path = "test_A.zip"

test_dir = "extracted_images"

with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(test_dir)

print(f"Dataset extracted to: {test_dir}")

def add_noise(image, noise_type="s&p"):
    if noise_type == "gauss":
        mean, var = 0, 0.0001
        sigma = var ** 0.5
        gauss = np.random.normal(mean, sigma, image.shape).astype(np.float32)
        noisy = np.clip(image + gauss, 0, 1)
        return noisy
    elif noise_type == "s&p":
        prob = 0.02
        noisy = np.copy(image)
        black = np.random.rand(*image.shape) < prob / 2
        white = np.random.rand(*image.shape) < prob / 2
        noisy[black] = 0
        noisy[white] = 1
        return noisy
    return image

def process_images(directory, noise_type="gauss"):
    for filename in os.listdir(directory):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            image_path = os.path.join(directory, filename)

            image = cv.imread(image_path)
            image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
            image = image / 255.0

            noisy_image = add_noise(image, noise_type=noise_type)

            plt.figure(figsize=(10, 5))
            plt.subplot(1, 2, 1)
            plt.title("Original Image")
            plt.imshow(image)
            plt.axis("off")

            plt.subplot(1, 2, 2)
            plt.title("Noisy Image")
            plt.imshow(noisy_image)
            plt.axis("off")
            plt.show()
```

```
process_images(test_dir, noise_type="gauss")
```

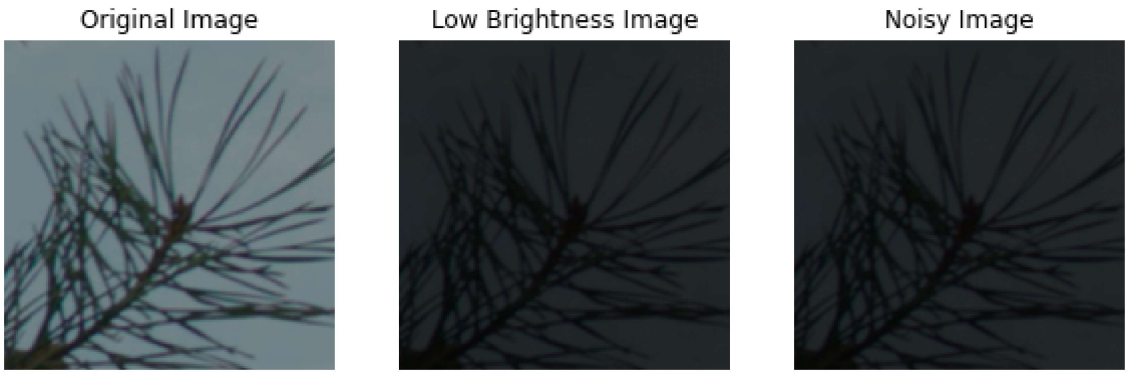
```
Dataset extracted to: extracted_images
```



Data Preprocessing

In [36]:
```
import os
import zipfile
import cv2 as cv
import numpy as np
import tempfile
import matplotlib.pyplot as plt
from pathlib import Path

def add_noise(img, noise_type="s&p"):
    return img

def preprocess_images_from_zip(zip_file, img_size=(256, 256)):
    X, y = [], []

    with tempfile.TemporaryDirectory() as temp_dir:
        with zipfile.ZipFile(zip_file, 'r') as zip_ref:
            zip_ref.extractall(temp_dir)

        for file in os.listdir(temp_dir):
            try:
                img_path = os.path.join(temp_dir, file)

                if file.lower().endswith(('.png', '.jpg', '.jpeg')):
                    img = cv.imread(img_path)
                    if img is None:
                        continue

                    img = cv.cvtColor(img, cv.COLOR_BGR2RGB) / 255.0  # Normal
                    img_resized = cv.resize(img, img_size)

                    # Apply low brightness
                    hsv = cv.cvtColor((img_resized * 255).astype('uint8'), cv.
                    hsv[..., 2] = hsv[..., 2] * 0.3  # Reduce brightness
                    img_low_light = cv.cvtColor(hsv, cv.COLOR_HSV2RGB) / 255.0

                    # Add noise
                    noisy_img = add_noise(img_low_light, noise_type="s&p")  # A
                    X.append(noisy_img)
                    y.append(img_resized)

                    # Visualizations of processed images (Original, Low Light,
                    if len(X) == 5:
                        plt.figure(figsize=(10, 4))

                        # Original image
                        plt.subplot(1, 3, 1)
                        plt.imshow(img_resized)
                        plt.title("Original Image")
                        plt.axis('off')

                        # Low light image
                        plt.subplot(1, 3, 2)
                        plt.imshow(img_low_light)
                        plt.title("Low Brightness Image")
                        plt.axis('off')

                        # Noisy image
```

```
                plt.subplot(1, 3, 3)
                plt.imshow(noisy_img)
                plt.title("Noisy Image")
                plt.axis('off')

                plt.show()

            except Exception as e:
                print(f"Error processing file {file}: {e}")

    return np.array(X), np.array(y)

# Example usage:
zip_file_path = "test_A.zip"  # Modified to use your zip file
X, y = preprocess_images_from_zip(zip_file_path)
print(f"Processed {len(X)} images.")
```

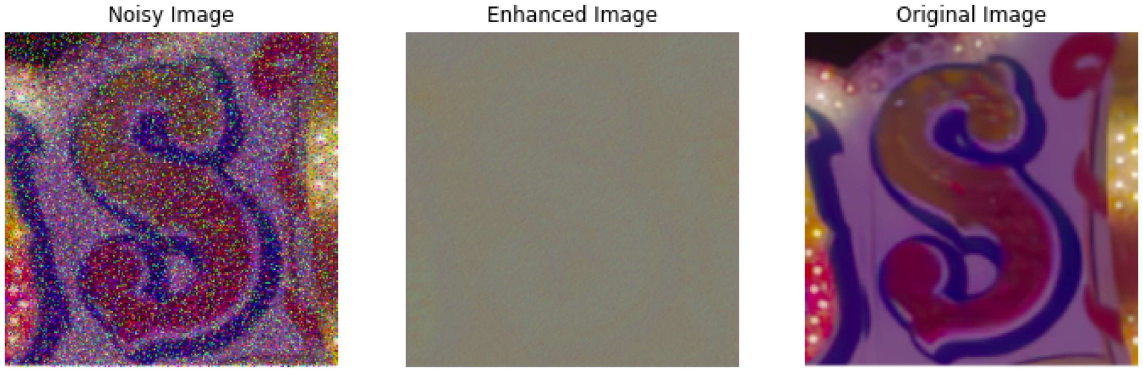



```
Processed 7 images.
```

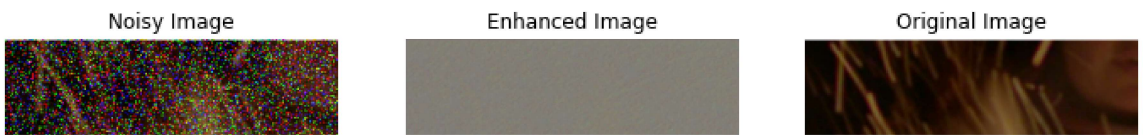
In [35]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, add
from tensorflow.keras.optimizers import Adam
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
import tempfile
import zipfile
import os

# Function to add noise (dummy function for now)
def add_noise(img, noise_type="s&p"):
    noise = np.random.rand(*img.shape)
    noisy_img = img.copy()
    noisy_img[noise < 0.1] = 0
    noisy_img[noise > 0.9] = 1
    return noisy_img

# Build the image enhancement model
def build_enhancement_model(input_shape=(256, 256, 3)):
    inp = Input(shape=input_shape)

    # Initial convolutions
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(inp)
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)
    conv3 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv2)

    conv1_1x1 = Conv2D(64, (1, 1), activation='relu', padding='same')(conv1)

    add1 = add([conv1_1x1, conv3])

    conv4 = Conv2D(32, (3, 3), activation='relu', padding='same')(add1)
    conv5 = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(conv4)

    model = Model(inputs=inp, outputs=conv5)
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model

def preprocess_images_from_zip(zip_file):
    images = []
    with tempfile.TemporaryDirectory() as temp_dir:
        with zipfile.ZipFile(zip_file, 'r') as zip_ref:
            zip_ref.extractall(temp_dir)

        for file in os.listdir(temp_dir):
            img_path = os.path.join(temp_dir, file)

            if file.lower().endswith(('.png', '.jpg', '.jpeg')):
                img = cv.imread(img_path)
                img = cv.cvtColor(img, cv.COLOR_BGR2RGB) / 255.0
                img_resized = cv.resize(img, (256, 256))
                images.append(img_resized)

    return np.array(images)


def preprocess_and_predict(zip_file, model):
```

```
    images = preprocess_images_from_zip(zip_file)

    for img_resized in images:
        # Add noise to the image
        noisy_img = add_noise(img_resized, noise_type="s&p")

        noisy_img_input = np.expand_dims(noisy_img, axis=0)

        enhanced_img = model.predict(noisy_img_input)

        enhanced_img_rescaled = np.clip(enhanced_img[0] * 255, 0, 255).astype(

        plt.figure(figsize=(12, 4))

        plt.subplot(1, 3, 1)
        plt.imshow(noisy_img)
        plt.title("Noisy Image")
        plt.axis('off')

        plt.subplot(1, 3, 2)
        plt.imshow(enhanced_img_rescaled)
        plt.title("Enhanced Image")
        plt.axis('off')

        plt.subplot(1, 3, 3)
        plt.imshow(img_resized)
        plt.title("Original Image")
        plt.axis('off')

        plt.show()

model = build_enhancement_model(input_shape=(256, 256, 3))

zip_file_path = 'test_A.zip'
preprocess_and_predict(zip_file_path, model)
```

```
1/1 [==============================] - 0s 294ms/step
```



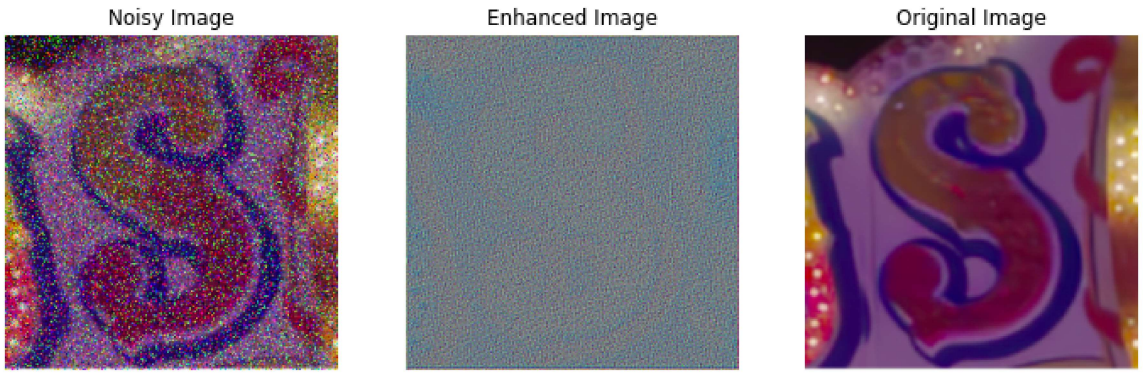

```
1/1 [==============================] - 0s 169ms/step
```



In [26]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, add
from tensorflow.keras.optimizers import Adam
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
import tempfile
import zipfile
import os

def add_noise(img, noise_type="s&p"):
    noise = np.random.rand(*img.shape)
    noisy_img = img.copy()
    noisy_img[noise < 0.1] = 0  # Black pixel
    noisy_img[noise > 0.9] = 1  # White pixel
    return noisy_img

def sharpen_image(image):
    kernel = np.array([[-1, -1, -1],
                       [-1,  9, -1],
                       [-1, -1, -1]])
    return cv.filter2D(image, -1, kernel)

def build_enhancement_model(input_shape=(256, 256, 3)):
    inp = Input(shape=input_shape)

    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(inp)
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)
    conv3 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv2)

    conv1_1x1 = Conv2D(64, (1, 1), activation='relu', padding='same')(conv1)

    add1 = add([conv1_1x1, conv3])

    conv4 = Conv2D(32, (3, 3), activation='relu', padding='same')(add1)
    conv5 = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(conv4)

    model = Model(inputs=inp, outputs=conv5)
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model

def preprocess_images_from_zip(zip_file):
    images = []
    with tempfile.TemporaryDirectory() as temp_dir:
        with zipfile.ZipFile(zip_file, 'r') as zip_ref:
            zip_ref.extractall(temp_dir)
        for file in os.listdir(temp_dir):
            img_path = os.path.join(temp_dir, file)
            if file.lower().endswith(('.png', '.jpg', '.jpeg')):
                img = cv.imread(img_path)
                img = cv.cvtColor(img, cv.COLOR_BGR2RGB) / 255.0
                img_resized = cv.resize(img, (256, 256))
                images.append(img_resized)

    return np.array(images)

def preprocess_and_predict(zip_file, model):
```

```
    images = preprocess_images_from_zip(zip_file)

    for img_resized in images:
        noisy_img = add_noise(img_resized, noise_type="s&p")

        noisy_img_input = np.expand_dims(noisy_img, axis=0)

        enhanced_img = model.predict(noisy_img_input)

        enhanced_img_sharpened = sharpen_image(enhanced_img[0])

        enhanced_img_rescaled = np.clip(enhanced_img_sharpened * 255, 0, 255).

        plt.figure(figsize=(12, 4))

        plt.subplot(1, 3, 1)
        plt.imshow(noisy_img)
        plt.title("Noisy Image")
        plt.axis('off')

        plt.subplot(1, 3, 2)
        plt.imshow(enhanced_img_rescaled)
        plt.title("Enhanced Image")
        plt.axis('off')

        plt.subplot(1, 3, 3)
        plt.imshow(img_resized)
        plt.title("Original Image")
        plt.axis('off')

        plt.show()

model = build_enhancement_model(input_shape=(256, 256, 3))

zip_file_path = 'test_A.zip'
preprocess_and_predict(zip_file_path, model)
```

```
1/1 [==============================] - 0s 279ms/step
```



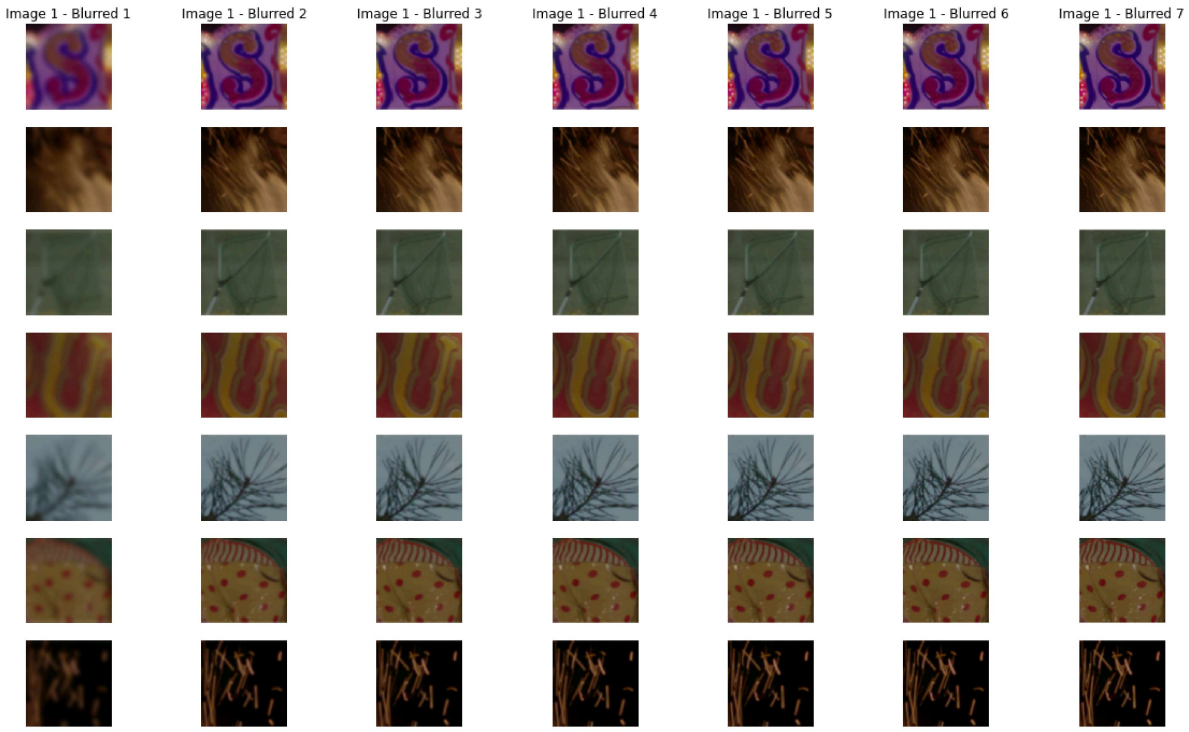

```
1/1 [==============================] - 0s 170ms/step
```

In [25]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, Add, Concatenate, Layer
from tensorflow.keras.optimizers import Adam
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
import tempfile
import zipfile
import os

# Add noise to images (Salt & Pepper noise for demonstration)
def add_noise(img, noise_type="s&p"):
    noise = np.random.rand(*img.shape)
    noisy_img = img.copy()
    noisy_img[noise < 0.1] = 0  # Add black pixels
    noisy_img[noise > 0.9] = 1  # Add white pixels
    return noisy_img

# Apply Gaussian blur with much stronger blur for the first image (extra blur)
def apply_blur(img, num_stages=7):
    blurred_images = []

    # Apply an extremely strong blur for the first image (making it very misty
    first_image_blur_strength = 2 * num_stages + 15  # Very large kernel size
    first_blurred_img = cv.GaussianBlur(img, (first_image_blur_strength, first

    # Apply an additional blur pass to make it even more blurry (extreme blur
    first_blurred_img = cv.GaussianBlur(first_blurred_img, (first_image_blur_s
    blurred_images.append(first_blurred_img)

    # Apply decreasing blur for the rest of the images
    for i in range(1, num_stages):
        blur_strength = 2 * (num_stages - i) + 1  # Gradually reduce the blur
        blurred_img = cv.GaussianBlur(img, (blur_strength, blur_strength), 0)
        blurred_images.append(blurred_img)

    return blurred_images

# Preprocess images taken from multiple angles
def preprocess_multi_angle_images(zip_file):
    angles = []
    with tempfile.TemporaryDirectory() as temp_dir:
        with zipfile.ZipFile(zip_file, 'r') as zip_ref:
            zip_ref.extractall(temp_dir)

        for file in sorted(os.listdir(temp_dir)):  # Sorting ensures consisten
            img_path = os.path.join(temp_dir, file)
            if file.lower().endswith(('.png', '.jpg', '.jpeg')):
                img = cv.imread(img_path, cv.IMREAD_UNCHANGED)
                img = cv.cvtColor(img, cv.COLOR_BGR2RGB) / 255.0
                img_resized = cv.resize(img, (256, 256))
                angles.append(img_resized)

    return np.array(angles)  # Shape: (num_angles, height, width, channels)

# Custom Layer for Feature Aggregation
```

```
class FeatureAggregationLayer(Layer):
    def __init__(self, num_angles, **kwargs):
        super(FeatureAggregationLayer, self).__init__(**kwargs)
        self.num_angles = num_angles

    def build(self, input_shape):
        # Create a learnable weight matrix for feature aggregation
        self.conv = Conv2D(64, (1, 1), activation='relu', padding='same', use_
        super(FeatureAggregationLayer, self).build(input_shape)

    def call(self, inputs):
        # Concatenate features from different angles
        aggregated = Concatenate(axis=-1)(inputs)
        return self.conv(aggregated)

# Bracketing image restoration and enhancement model for multi-angle inputs
def build_multi_angle_model(input_shape=(256, 256, 3), num_angles=7):
    inputs = [Input(shape=input_shape) for _ in range(num_angles)]

    angle_features = []
    for inp in inputs:
        conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(inp)
        conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)
        angle_features.append(conv2)

    # Use the custom feature aggregation layer
    aggregated = FeatureAggregationLayer(num_angles)(angle_features)

    # Enhancement and restoration with residual connections
    conv3 = Conv2D(64, (3, 3), activation='relu', padding='same')(aggregated)
    conv4 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv3)
    restored_output = Conv2D(3, (3, 3), activation='relu', padding='same')(con

    model = Model(inputs=inputs, outputs=restored_output)
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model

# Process multi-angle images and predict
def predict_multi_angle_images(zip_file, model, num_angles=7):
    angles = preprocess_multi_angle_images(zip_file)

    if len(angles) < num_angles:
        raise ValueError(f"The model expects {num_angles} angles, but fewer we
    if len(angles) > num_angles:
        angles = angles[:num_angles]  # Use only the first `num_angles` inputs

    noisy_inputs = [add_noise(angle) for angle in angles]  # Add noise to inpu

    # Expand dimensions for each input to match the model's expected shape
    inputs = [np.expand_dims(noisy, axis=0) for noisy in noisy_inputs]

    # Generate blurred versions of each image in the dataset (starting with st
    all_blurred_images = []  # To store blurred images for each of the 7 image
    for angle in angles:
        blurred_images = apply_blur(angle, num_stages=num_angles)  # Apply to
        all_blurred_images.append(blurred_images)
```

```
    # Model prediction (restoration of the image)
    restored_image = model.predict(inputs)

    # Ensure proper scaling for the final restored image
    restored_image_rescaled = np.clip(restored_image[0] * 255, 0, 255).astype(

    # Plot results for each of the 7 images
    plt.figure(figsize=(20, 12))

    # Show the blurred transitions for each image (starting with heavy blur, g
    for img_idx, blurred_images in enumerate(all_blurred_images):
        for i, blurred_img in enumerate(blurred_images):
            plt.subplot(len(angles), num_angles, img_idx * num_angles + i + 1)
            plt.imshow(blurred_img)
            if img_idx == 0:
                plt.title(f"Image {img_idx+1} - Blurred {i+1}")
            plt.axis('off')

    plt.show()

# Build and use the model
multi_angle_model = build_multi_angle_model()
zip_file_path = 'test_A.zip'  # Replace with the actual zip file path
predict_multi_angle_images(zip_file_path, multi_angle_model)
```

```
1/1 [==============================] - 1s 688ms/step
```



Training The Model

In [31]:
```
import zipfile
import os
import cv2 as cv
import numpy as np
import tempfile

# Preprocess images without returning labels
def preprocess_images_from_zip(zip_file, img_size=(256, 256)):
    images = []
    with tempfile.TemporaryDirectory() as temp_dir:
        with zipfile.ZipFile(zip_file, 'r') as zip_ref:
            zip_ref.extractall(temp_dir)

        for file in sorted(os.listdir(temp_dir)):  # Sorting ensures consisten
            img_path = os.path.join(temp_dir, file)
            if file.lower().endswith(('.png', '.jpg', '.jpeg')):
                img = cv.imread(img_path, cv.IMREAD_UNCHANGED)
                img = cv.cvtColor(img, cv.COLOR_BGR2RGB) / 255.0  # Normalize
                img_resized = cv.resize(img, img_size)  # Resize image to the
                images.append(img_resized)

    return np.array(images)  # Return only images (no labels)
train_dir = "test_A"

# Only return images
X_train = preprocess_images_from_zip("test_A.zip", img_size=(256, 256))

# Assuming you don't need y_train (labels), you can proceed with just the imag
model = build_enhancement_model(input_shape=(256, 256, 3))
model.summary()

# If you don't need labels, simply pass X_train (images) to the model
model.fit(X_train, X_train, batch_size=16, epochs=20, validation_split=0.2)
```

```
Model: "model_25"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected
to
==================================================================================
 input_165 (InputLayer)      [(None, 256, 256, 3)]     0         []

 conv2d_432 (Conv2D)         (None, 256, 256, 32)      896       ['input_1
65[0][0]']

 conv2d_433 (Conv2D)         (None, 256, 256, 64)      18496     ['conv2d_
432[0][0]']

 conv2d_435 (Conv2D)         (None, 256, 256, 64)      2112      ['conv2d_
432[0][0]']

 conv2d_434 (Conv2D)         (None, 256, 256, 64)      36928     ['conv2d_
433[0][0]']

 add_2 (Add)                 (None, 256, 256, 64)      0         ['conv2d_
435[0][0]',
                                                                  'conv2d_
434[0][0]']

 conv2d_436 (Conv2D)         (None, 256, 256, 32)      18464     ['add_2
[0][0]']

 conv2d_437 (Conv2D)         (None, 256, 256, 3)       867       ['conv2d_
436[0][0]']

==================================================================================
Total params: 77763 (303.76 KB)
Trainable params: 77763 (303.76 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/20
1/1 [==============================] - 6s 6s/step - loss: 0.0669 - val_loss:
0.1279
Epoch 2/20
1/1 [==============================] - 2s 2s/step - loss: 0.0587 - val_loss:
0.1197
Epoch 3/20
1/1 [==============================] - 2s 2s/step - loss: 0.0513 - val_loss:
0.1101
Epoch 4/20
1/1 [==============================] - 2s 2s/step - loss: 0.0448 - val_loss:
0.1002
Epoch 5/20
1/1 [==============================] - 2s 2s/step - loss: 0.0414 - val_loss:
0.0934
Epoch 6/20
1/1 [==============================] - 2s 2s/step - loss: 0.0413 - val_loss:
0.0893
```

```
Epoch 7/20
1/1 [==============================] - 2s 2s/step - loss: 0.0409 - val_loss:
0.0870
Epoch 8/20
1/1 [==============================] - 2s 2s/step - loss: 0.0390 - val_loss:
0.0861
Epoch 9/20
1/1 [==============================] - 2s 2s/step - loss: 0.0368 - val_loss:
0.0861
Epoch 10/20
1/1 [==============================] - 2s 2s/step - loss: 0.0353 - val_loss:
0.0861
Epoch 11/20
1/1 [==============================] - 2s 2s/step - loss: 0.0341 - val_loss:
0.0854
Epoch 12/20
1/1 [==============================] - 2s 2s/step - loss: 0.0327 - val_loss:
0.0838
Epoch 13/20
1/1 [==============================] - 2s 2s/step - loss: 0.0310 - val_loss:
0.0811
Epoch 14/20
1/1 [==============================] - 2s 2s/step - loss: 0.0291 - val_loss:
0.0776
Epoch 15/20
1/1 [==============================] - 2s 2s/step - loss: 0.0273 - val_loss:
0.0739
Epoch 16/20
1/1 [==============================] - 2s 2s/step - loss: 0.0258 - val_loss:
0.0705
Epoch 17/20
1/1 [==============================] - 2s 2s/step - loss: 0.0246 - val_loss:
0.0673
Epoch 18/20
1/1 [==============================] - 2s 2s/step - loss: 0.0230 - val_loss:
0.0642
Epoch 19/20
1/1 [==============================] - 2s 2s/step - loss: 0.0216 - val_loss:
0.0608
Epoch 20/20
1/1 [==============================] - 3s 3s/step - loss: 0.0203 - val_loss:
0.0572
```

Out[31]: `<keras.src.callbacks.History at 0x2262fab3eb0>`

Testing The Model

In [34]:
```
import os
import zipfile
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

def extract_zip(zip_path, extract_to):
    """Extract images from the zip file."""
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_to)
    print(f"Extracted files to {extract_to}")

def add_noise(image, noise_type="s&p"):
    """Function to add salt and pepper noise (example, you can modify it)."""
    if noise_type == "s&p":
        row, col, _ = image.shape
        s_vs_p = 0.5
        amount = 0.02
        noisy_image = np.copy(image)
        # Salt mode
        num_salt = int(amount * image.size * s_vs_p)
        salt_coords = [np.random.randint(0, i - 1, num_salt) for i in image.sh
        noisy_image[salt_coords[0], salt_coords[1]] = 1
        # Pepper mode
        num_pepper = int(amount * image.size * (1.0 - s_vs_p))
        pepper_coords = [np.random.randint(0, i - 1, num_pepper) for i in imag
        noisy_image[pepper_coords[0], pepper_coords[1]] = 0
        return noisy_image
    return image

def test_model(image_dir, filename, model):
    img_path = os.path.join(image_dir, filename)

    if not os.path.exists(img_path):
        print(f"Error: Image file '{filename}' not found at '{img_path}'")
        return

    img = cv.imread(img_path)

    if img is None:
        print(f"Error: Failed to load image at '{img_path}'")
        return

    img = cv.cvtColor(img, cv.COLOR_BGR2RGB) / 255.0
    img_resized = cv.resize(img, (256, 256))

    hsv = cv.cvtColor((img_resized * 255).astype('uint8'), cv.COLOR_RGB2HSV)
    hsv[..., 2] = hsv[..., 2] * 0.3
    low_light = cv.cvtColor(hsv, cv.COLOR_HSV2RGB) / 255.0
    noisy = add_noise(low_light, noise_type="s&p")

    noisy_input = np.expand_dims(noisy, axis=0)
    enhanced_img = model.predict(noisy_input)[0]

    plt.figure(figsize=(15, 5))

    plt.subplot(1, 3, 1)
```

```
    plt.title("Original")
    plt.imshow(img_resized)
    plt.axis("off")

    plt.subplot(1, 3, 2)
    plt.title("Low Light + Noise")
    plt.imshow(noisy)
    plt.axis("off")

    plt.subplot(1, 3, 3)
    plt.title("Enhanced")
    plt.imshow(enhanced_img)
    plt.axis("off")

    plt.show()

zip_file_path = "test_A.zip"
extract_dir = "test_A"


extract_zip(zip_file_path, extract_dir)

test_model(extract_dir, "Image_2", model)
```

```
Extracted files to test_A
Error: Image file 'Image_2' not found at 'test_A\Image_2'
```

In [ ]: