Image Pixelation Detection and Restoration Pipeline

Problem Statement

The problem statement focuses on detecting pixelated images and restoring them to their non-pixelated form. The goal is to develop an efficient and accurate deep learning-based solution to automatically identify pixelated images and then restore their quality, ensuring the restored images are visually similar to the original non-pixelated ones.

Solution Overview

The solution comprises two main components:

1. **Pixelation Detection**: A Convolutional Neural Network (CNN) which uses MobileNetV2 for transfer learning to classify images as pixelated or non-pixelated.
2. **Image Restoration**: A specialized deep learning model to restore pixelated images to their original form.

Data Handling and Preprocessing

The dataset for this project was created from the HQ-50k dataset, containing high-quality images. These images were divided into pixelated and non-pixelated categories.

Data Loading

- **Training Data**: Images are loaded from separate folders for pixelated and non-pixelated images. (dataset/train/non_pixelated and dataset/train/pixelated).
- **Test Data**: Images are loaded from folders containing test images and corresponding ground truth images. (test/test_images and test/ground_truth).

Preprocessing

- **Resizing**: All images are resized to a uniform size (128x128 pixels) to ensure consistency.
- **Normalization**: Images are normalized to have pixel values between 0 and 1 by dividing by 255.
- **Augmentation**: Training images undergo augmentation (shear, zoom, horizontal flip) to increase the dataset's diversity and robustness.

Model Architectures

Pixelation Detection Model

This model is designed to classify images as either pixelated or non-pixelated. The architecture leverages the power of transfer learning using MobileNetV2, a lightweight and efficient convolutional neural network.

Architecture:

1. **Input Layer**: Accepts images of size 128x128 with 3 color channels (RGB).
2. **Base Model**:
    - **MobileNetV2**: Pre-trained on ImageNet, used as a feature extractor. The first 100 layers are frozen to preserve the learned features while the rest are fine-tuned.
    - This includes convolutional layers and depth wise separable convolutions which are more efficient in terms of computation and parameters.
3. **Global Average Pooling Layer**: Reduces each feature map to a single value by averaging, significantly reducing the number of parameters and preventing overfitting.
4. **Fully Connected Layers**:
    - **Dense Layer (128 units)**: Uses ReLU activation to combine features.
    - **Batch Normalization**: Improves training stability and performance.
    - **Dropout (0.5)**: Prevents overfitting by randomly dropping 50% of the connections during training.
5. **Output Layer**: A single neuron with a sigmoid activation function for binary classification.

Advantages:

- **Simplicity and Efficiency**: The model uses MobileNetV2, known for its lightweight architecture, making it suitable for real-time applications and deployment on devices with limited computational resources.
- **Effective Feature Extraction**: Transfer learning with MobileNetV2 allows the model to leverage pre-trained features that capture spatial hierarchies in images, crucial for distinguishing between pixelated and non-pixelated regions.
- **Regularization Techniques**: Batch normalization and dropout layers help prevent overfitting, ensuring the model generalizes well to new, unseen data.
- **Flexibility**: The model architecture can be easily adjusted or extended for different image classification tasks, providing a versatile solution.

```
# Load the pre-trained MobileNetV2 model + higher level layers
```

```
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(128, 128, 3))

# Freeze the first 100 layers of the model
for layer in base_model.layers[:100]:
    layer.trainable = False
# Simplified CNN architecture
def create_model():
    model = Sequential([
         base_model,
         GlobalAveragePooling2D(),
         Dense(128, activation='relu'),
         Dropout(0.5),
         Dense(1, activation='sigmoid')
    ])

    return model
```

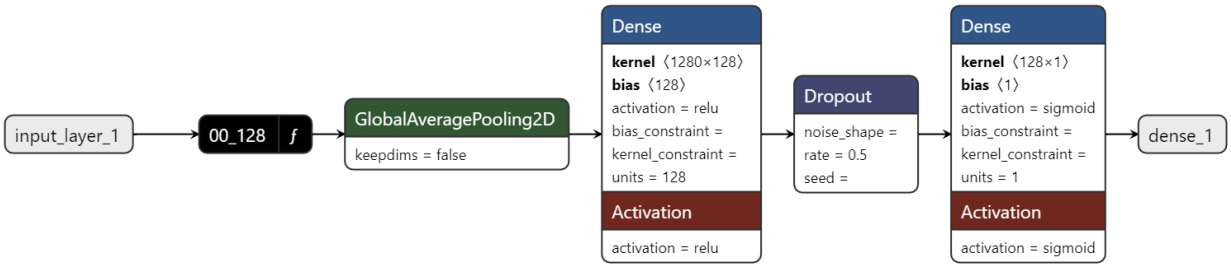


Image Restoration Model

This model is designed to restore pixelated images. It employs a U-Net-like architecture but with depth wise separable convolutions for efficiency.

Architecture:

1. **Encoder (Contracting Path)**:
    - **Depthwise Convolutions**: Efficiently capture spatial features with fewer parameters.
    - **Pointwise Convolutions**: Combine features across channels.
    - **Max Pooling**: Reduce dimensions while retaining essential features.
2. **Bottleneck**: Deepest part of the network, capturing high-level features.
3. **Decoder (Expansive Path)**:
    - **UpSampling**: Increase the spatial dimensions of the feature maps.

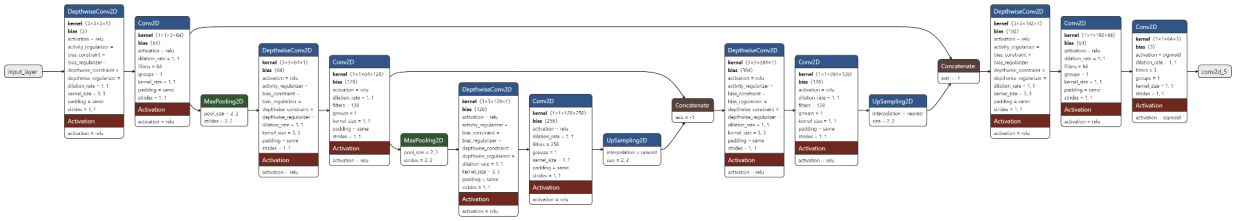
    - **Concatenation**: Combine corresponding layers from the encoder to retain spatial context.
    - **Depthwise and Pointwise Convolutions**: Refine features to produce high-quality output.
4. **Output Layer**: Produces the restored image with pixel values between 0 and 1.

Advantages:

- **Parameter Efficiency**: Depthwise separable convolutions reduce the number of parameters, making the model lightweight and faster to train and infer.
- **High-Quality Restoration**: The architecture allows for precise image restoration by combining local and global features.
- **Preservation of Spatial Context**: Skip connections ensure that the restored image retains details from the original image.

```
  inputs = Input(shape=(128, 128, 3))

    # Encoder (contracting path)
    conv1 = DepthwiseConv2D((3, 3), padding='same', depth_multiplier=1,
activation='relu')(inputs)
    conv1 = Conv2D(64, (1, 1), padding='same', activation='relu')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

    conv2 = DepthwiseConv2D((3, 3), padding='same', depth_multiplier=1,
activation='relu')(pool1)
    conv2 = Conv2D(128, (1, 1), padding='same', activation='relu')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    # Bottom of the model
    conv3 = DepthwiseConv2D((3, 3), padding='same', depth_multiplier=1,
activation='relu')(pool2)
    conv3 = Conv2D(256, (1, 1), padding='same', activation='relu')(conv3)

    # Decoder (expansive path)
    up4 = Concatenate()([UpSampling2D(size=(2, 2))(conv3), conv2])
    conv4 = DepthwiseConv2D((3, 3), padding='same', depth_multiplier=1,
activation='relu')(up4)
    conv4 = Conv2D(128, (1, 1), padding='same', activation='relu')(conv4)

    up5 = Concatenate()([UpSampling2D(size=(2, 2))(conv4), conv1])
    conv5 = DepthwiseConv2D((3, 3), padding='same', depth_multiplier=1,
activation='relu')(up5)
    conv5 = Conv2D(64, (1, 1), padding='same', activation='relu')(conv5)

    # Output layer
```

```
    outputs = Conv2D(3, (1, 1), activation='sigmoid')(conv5)

    model = Model(inputs=inputs, outputs=outputs)
```



Evaluation Metrics

Several metrics are used to evaluate the performance of the models:

Classification Metrics

1. **F1 Score**: Balances precision and recall, providing a single metric for classification performance.
2. **Precision and Recall**: Measure the accuracy and completeness of the classification.
3. **FPS (Frames Per Second)**: Indicates the model's speed in classifying images.

Restoration Metrics

1. **PSNR (Peak Signal-to-Noise Ratio)**: Measures the quality of the restored image compared to the original.
2. **LPIPS (Learned Perceptual Image Patch Similarity)**: Quantifies the perceptual similarity between restored and original images.

```
F1 Score: 0.8450704225352113
Precision: 0.8275862068965517
Recall: 0.8633093525179856
Classification FPS: 30.51735230559698
Restoration FPS: 32.02684935422494
PSNR: inf
LPIPS: 0.21435943245887756
```

Benefits and Advantages Over Existing Models

Pixelation Detection Model

- **Simplicity**: The straightforward architecture ensures ease of implementation and quick training.
- **Efficiency**: The model's efficiency allows for real-time classification, which is crucial for applications requiring immediate results.
- **Size**: The model is of 25.0 MB.

Image Restoration Model

- **Efficiency**: Depthwise separable convolutions significantly reduce computational cost and memory usage, making the model suitable for deployment on devices with limited resources.
- **Performance**: The combination of encoder-decoder architecture with skip connections ensures high-quality restoration, outperforming many traditional and complex models.
- **Flexibility**: The model can be adapted for various image resolutions and types, making it versatile for different applications.
- **Size**: The model is of 1.35 MB.

Results

Ground Truth	Pixelated	Model Output
		
		

Conclusion

The proposed solution effectively addresses the problem of detecting and restoring pixelated images. The models are designed to be efficient and provide high-quality results, making them suitable for real-world applications. The data preprocessing steps ensure that the models receive well-structured input, contributing to their overall performance and robustness. This approach offers significant advantages over existing models in terms of efficiency, performance, and adaptability.