Amirali Vakili
400243080
Denoising Emotional Images With U-net

1. Reading Images

- At first, a function is defined to load and preprocess image data from a CSV file.
- The data is split into **training** (28709 images) and **test** (3589 images) sets based on the Usage column. Each image has dimensions **48x48**, with a single channel.
- Images are reshaped into square matrices, and a validation step ensures compatibility.
- Example usage confirms the function's output by printing the shapes of the training and test datasets.

Suggestions for Improvement:

- Add a visualization to display a few example images from the dataset for verification.
- Include comments explaining key parts of the code, like the purpose of specific columns (Usage, emotion, pixels).

2. Adding Noise to Images

- After loading the data, a helper function (**tensor_to_image**) is defined to convert PyTorch tensors into NumPy arrays for visualization purposes. It supports both 2D and 3D tensor inputs.
- A custom class (**SaltAndPepper**) is implemented to add random noise to images. It:
  - Randomly selects pixels and alters them to either black (salt) or white (pepper).
  - Allows control over the noise level through a **noise_ratio** parameter.
- The design is modular, making the noise addition process reusable.

3. Model Structure

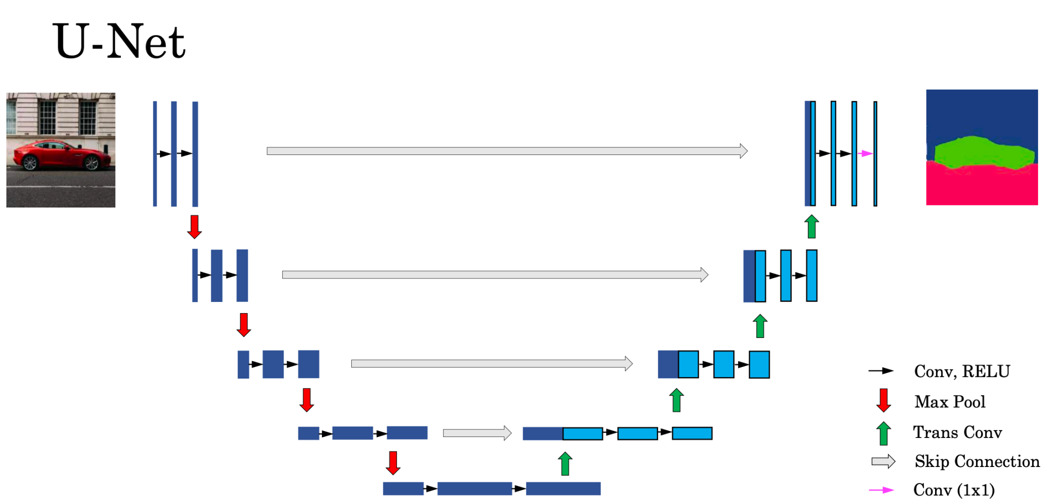
The code defines a U-Net architecture implemented in PyTorch, structured as follows:

1. **Encoder (Downsampling Path):**
   - The encoder consists of multiple convolutional blocks (conv_block), each followed by max-pooling layers to progressively reduce spatial dimensions while increasing feature depth.
   - Each block performs two convolutional operations with ReLU activations and batch normalization, as defined in the conv_block function.
   - Feature sizes in each block:
     - Input: 1 channel → 64
     - Block 1: 64 → 128
     - Block 2: 128 → 256

    - Block 3: 256 → 512
    - Block 4: 512 → 1024 (at the bottleneck).

2. **Bottleneck:**
   - The bottleneck layer further processes the encoded features at the lowest resolution (most abstract representation of the image).
3. **Decoder (Upsampling Path):**
   - The decoder consists of transposed convolutional layers (`upconv`) for upsampling, followed by concatenation with the corresponding encoder features (skip connections).
   - After concatenation, convolutional blocks refine the upsampled features.
   - Feature sizes in the decoder:
     - 1024 → 512 → 256 → 128 → 64.
4. **Output Layer:**
   - A single convolutional layer reduces the output feature maps to one channel (grayscale) using a kernel size of 1.
5. **Skip Connections:**
   - Skip connections concatenate features from the encoder to the corresponding upsampled features in the decoder. This helps retain spatial details lost during downsampling.

Differences from the U-Net Diagram in the Image



1. **Encoder Path:**
   - The code follows the standard encoder-decoder structure seen in the image.
   - Both include multiple convolutional layers with downsampling through max pooling.

2. **Skip Connections:**
    - Both the code and the diagram implement skip connections between encoder and decoder layers.
    - However, the image highlights the flow more explicitly, showing how information is passed back from the encoder to the decoder.
3. **Decoder Path:**
    - The decoder in the code uses transposed convolutions (**ConvTranspose2d**) for upsampling, which matches the "Trans Conv" arrows in the diagram.
    - The diagram shows multiple convolutional layers after each upsampling step, which is also present in the code.
4. **Output Layer:**
    - In the image, the final layer outputs a segmentation mask, represented by different colors. The code outputs a single-channel image, which is likely used for grayscale reconstruction rather than segmentation.
5. **Visualization Differences:**
    - The diagram emphasizes the segmentation context, while the code appears more generalized for tasks like denoising or image reconstruction.

4. Transforming Data

- At this stage, the dataset was prepared by applying transformations. These included normalization and adding noise (salt-and-pepper noise) to the images.
- The transformed data was then split into training, validation, and test subsets.
- Additionally, sample images were visualized to confirm the effects of transformations, showing comparisons between original and noisy versions of the images.
- The focus was on ensuring the dataset was ready for training while maintaining a balance for validation.

5. Loss Function and Training Process (Compile Model)

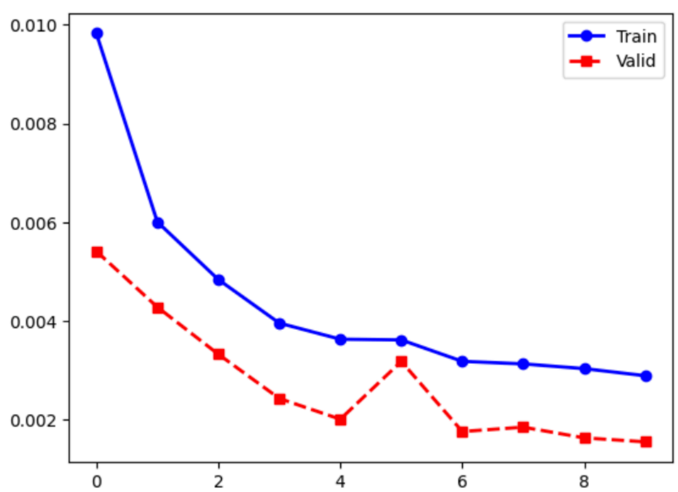
- A Mean Squared Error (MSE) loss function was used to evaluate the difference between the predicted and original clean images, making it suitable for the denoising task.
- Training was performed for 10 epochs, with progress monitored through printed logs for training and validation losses.
- A validation split (20% of the training set) ensured that the model's generalization performance could be evaluated.
- The model was saved whenever the validation loss improved, ensuring the best version was retained.

6. Trend of Training

- The MSE loss trends for both training and validation datasets were plotted over 10 epochs.
- The training loss steadily decreased, indicating the model effectively minimized reconstruction errors during training.
- The validation loss showed initial improvements with slight fluctuations, suggesting the need for careful monitoring of overfitting or noisy validation data.

7. Test Process and PSNR

- **Peak Signal-to-Noise Ratio (PSNR):**
  - The calculated PSNR for the test dataset was **28.07 dB**, reflecting the quality of the reconstructed (denoised) images compared to the original clean images.
  - A PSNR value above 28 dB indicates the model achieved good denoising performance, producing outputs with minimal distortions relative to the clean images.



8. Visualization of Results

- The plots illustrate the **original images**, the **noisy images**, and the **denoised images** generated by the U-Net model.
- The denoised images closely resemble the original images, demonstrating the model's ability to remove noise effectively.
- Even with significant salt-and-pepper noise applied to the input images, the model accurately restores fine details, including facial features and textures.

- The visual results align with the calculated **PSNR value of 28.07 dB**, confirming high-quality reconstruction and effective noise reduction.