**Compression via the Encoder for 128\*128**

Suppose the input image has dimensions of 128x128 pixels and 3 color channels. We'll go through each layer in your encoder to see how the image is compressed:

1. **First Convolutional Layer:**
   * New dimension

A black text on a white background

Description automatically generated

* + **Input Size**: 128×128×3128×128×3
  + **Operation**: Conv2d with 32 filters, kernel size = 3, stride = 2, padding = 1
  + **Output Size**: Since stride is 2, output dimensions will be halved.
  + **Calculation**

A black numbers on a white background

Description automatically generated

1. **Subsequent Convolutional Layers:**
   * **Pattern**: Repeated application of Conv2d layers continues to halve the dimensions each time (due to stride = 2) while potentially altering the depth based on the number of filters.
   * Each layer follows the same formula, reducing spatial dimensions and increasing depth:
     1. After the second Conv2d: 64×64×32 goes to 32×32×64
     2. After the third Conv2d: 32×32×64 goes to 16×16×64
     3. After the fourth Conv2d: 16×16×64 goes to 8×8×64
2. **Flatten**:
   * The output of the final Conv2d layer, 8×8×64, is flattened.
   * This results in a single vector of size 8×8×64=40968.
3. **Linear Layers for Latent Representation:**
   * This flattened vector is then fed into linear layers that map it to the latent space vectors 𝑚𝑢*mu* and 𝑙𝑜𝑔\_𝑣𝑎𝑟*log*\_*var*, each of size **latent\_space** (200 in your case).

When dealing with a Variational Autoencoder (VAE) that compresses and reconstructs images, it's essential to understand how each layer in the encoder and decoder affects the dimensions of the data. Let's take a more detailed look at the process, specifically when you start with an image of size 128x128.

**Reconstruction via the Decoder**

Starting from the latent space representation, the decoder aims to reconstruct the image back to its original size:

1. **Linear and Reshape:**
   * The latent vector is first expanded through a linear layer back to the size of 4096, then reshaped to match the feature map size before flattening in the encoder, which is 8×8×64.
2. **Transposed Convolutional Layers:**
   * These layers progressively upscale the feature maps back to larger spatial dimensions:
   * First layer: 8×8×64 goes to approximately 16×16×64
   * Second layer: 16×16×64 goes to approximately 32×32×64
   * Third layer: 32×32×64 goes to approximately 64×64×32
   * Fourth layer: 64×64×32 aims to go to 128×128×3
3. **Trim:**
   * If the last transposed convolution layer slightly overshoots the desired size due to padding and kernel size specifics (which might lead to an output like 129×129×3, the Trim layer adjusts it back to 128×128×3.
4. **Sigmoid Activation:**
   * Ensures that pixel values are normalized between 0 and 1, suitable for images.

**Encoder and Decoder Architecture for 1000\*1000 images:**

**Encoder:**

1. **Convolutional Layer 1**:
   * Input: 1000x1000x3 (height x width x channels)
   * Convolution: 32 filters, kernel size 4x4, stride 2, padding 1
   * Output size:
     + ((1000 - 4 + 2\*1) / 2) + 1 = 500x500x32
2. **Convolutional Layer 2**:
   * Input: 500x500x32
   * Convolution: 64 filters, kernel size 4x4, stride 2, padding 1
   * Output size:
     + ((500 - 4 + 2\*1) / 2) + 1 = 250x250x64
3. **Convolutional Layer 3**:
   * Input: 250x250x64
   * Convolution: 128 filters, kernel size 4x4, stride 2, padding 1
   * Output size:
     + ((250 - 4 + 2\*1) / 2) + 1 = 125x125x128
4. **Convolutional Layer 4**:
   * Input: 125x125x128
   * Convolution: 128 filters, kernel size 4x4, stride 2, padding 1
   * Output size:
     + ((125 - 4 + 2\*1) / 2) + 1 = 62x62x128

**Decoder:**

1. **Linear Layer**:
   * Input: Latent space vector (e.g., of size 200)
   * Output: Fully connected layer output of size 128 \* 62 \* 62 = 487,424
   * This linear layer expands the latent space representation to a size suitable for convolutional operations.
2. **Reshape**:
   * Reshape the output of the linear layer to a 4D tensor of size 128x62x62.
   * This prepares the data for the transposed convolutional layers.
3. **Transposed Convolutional Layer 1**:
   * Input: 128x62x62
   * Transposed Convolution: 128 filters, kernel size 4x4, stride 2
   * Output size:
     + ((62 - 1) \* 2 + 4) = 125x125x128
   * This layer upsamples the feature map to a larger spatial size.
4. **Transposed Convolutional Layer 2**:
   * Input: 125x125x128
   * Transposed Convolution: 64 filters, kernel size 4x4, stride 2, padding 1
   * Output size:
     + ((125 - 1) \* 2 + 4 - 2\*1) = 250x250x64
   * This layer further upsamples the feature map.
5. **Transposed Convolutional Layer 3**:
   * Input: 250x250x64
   * Transposed Convolution: 32 filters, kernel size 4x4, stride 2, padding 1
   * Output size:
     + ((250 - 1) \* 2 + 4 - 2\*1) = 500x500x32
   * This layer continues upsampling the feature map.
6. **Transposed Convolutional Layer 4**:
   * Input: 500x500x32
   * Transposed Convolution: 3 filters (output channels), kernel size 4x4, stride 2, padding 1
   * Output size:
     + ((500 - 1) \* 2 + 4 - 2\*1) = 1000x1000x3
   * This layer generates the final output image.
7. **Trim**:
   * Trim the output to match the desired output size, if necessary.
   * In this case, it trims the output from 1001x1001 to 1000x1000.