Question 10 :

To determine the dimensions of the output tensor after applying a convolution with a 5x5 kernel, padding 1, and stride 2 to an image with 3 channels and resolution 1024x768, you need to calculate the dimensions of the output tensor using the following formula:

Copy code

output\_height = (input\_height + 2 \* padding - kernel\_size) / stride + 1 output\_width = (input\_width + 2 \* padding - kernel\_size) / stride + 1 output\_channels = output\_channels

In this case, the input tensor has dimensions (3, 1024, 768), the kernel size is 5, the padding is 1, the stride is 2, and the number of output channels is 16. Plugging these values into the formula above, we get:

Copy code

output\_height = (1024 + 2 \* 1 - 5) / 2 + 1 = 513 output\_width = (768 + 2 \* 1 - 5) / 2 + 1 = 385 output\_channels = 16

Therefore, the dimensions of the output tensor after applying the convolution will be (16, 513, 385). The number of output channels will be 16, as specified in the input, and the height and width of the output tensor will be determined by the kernel size, padding, and stride of the convolution.

If we apply the same convolution to an image with 3 channels and resolution 1920x1080, the dimensions of the output tensor will be (16, 961, 641). This is because the input tensor has different dimensions than in the previous case, and therefore the output tensor will also have different dimensions.

If we apply the same convolution to an image with 8 channels and resolution 1920x1080, the output tensor will have dimensions (16, 961, 641), but the convolution will not be applied correctly, as the number of input channels (8) does not match the number of input channels expected by the convolution (3). In order to apply the convolution correctly, the number of input and output channels must match the dimensions expected by the convolution. In this case, the convolution expects an input tensor with 3 channels, so you would need to either change the number of channels in the input tensor to 3 or change the convolution to expect an input tensor with 8 channels.

Question 11 :

To implement a global max pool in PyTorch, you can use the **torch.max()** function and specify the **dim** argument to be **-1** (the default value is **None**, which pools over all dimensions). Here is an example:

Copy code

# Perform global max pooling on x x\_max\_pool = torch.max(x, dim=-1)

To implement a global mean pool in PyTorch, you can use the **torch.mean()** function and specify the **dim** argument to be **-1**. Here is an example:

Copy code

# Perform global mean pooling on x x\_mean\_pool = torch.mean(x, dim=-1)

Note that in both cases, the **dim** argument specifies which dimension to pool over. In the case of a 4D tensor with dimensions (batch, channels, height, width), setting **dim=-1** will perform pooling over the last dimension, which corresponds to the width of the image. This will produce a tensor with dimensions (batch, channels) that is the result of applying max or mean pooling across all the images in the batch.

Text

Description automatically generated

Question 13

One downside of resizing all the data to a uniform resolution of 64x64 is that it may cause a loss of information or distortion in the images. This is because resizing an image can cause loss of detail or blurriness, depending on the resizing method used. Additionally, resizing all the data to a uniform resolution may not be necessary if the original data already has a consistent resolution, and it may result in unnecessary computational overhead.

Another potential downside of resizing all the data to a uniform resolution of 64x64 is that it may not be optimal for the network's performance. This is because the network may be designed to operate on a specific input resolution, and resizing the data to a different resolution may affect the network's performance.

Furthermore, resizing all the data to a uniform resolution of 64x64 may not be necessary if the network is designed to handle variable input resolutions. In this case, resizing the data to a uniform resolution may not provide any benefits and may even hinder the network's performance.

In summary, the downsides of resizing all the data to a uniform resolution of 64x64 include potential loss of information or distortion in the images, unnecessary computational overhead, and potentially suboptimal performance of the network.