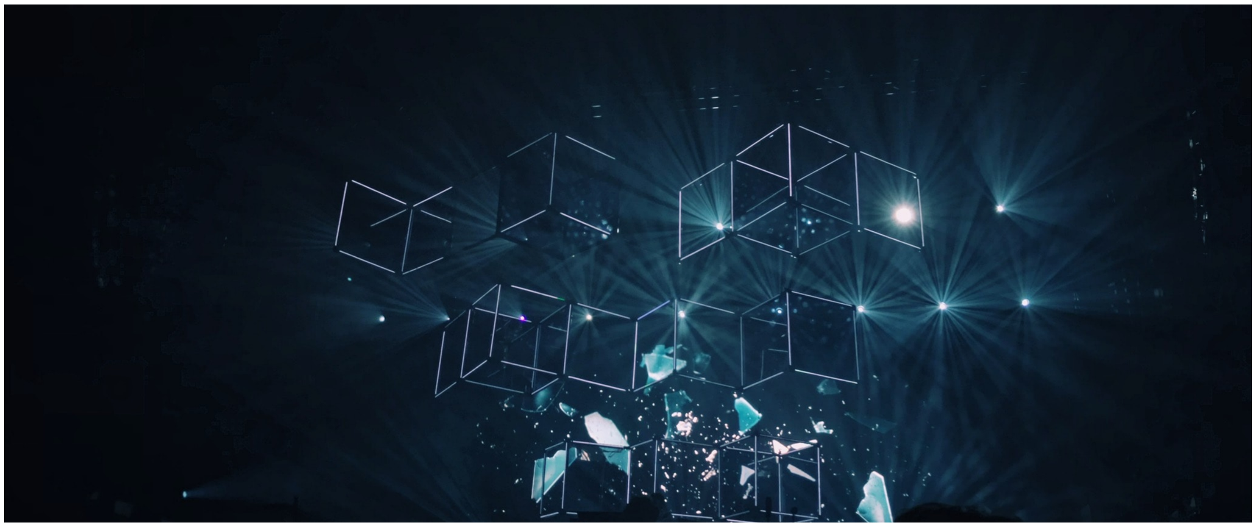

Enhancing Vision with Convolutional Neural Networks

1. **Convolution**: A filter that passes over the image in order to change the underlying image. For every pixel, its value and it's neighbor's value forms a filter grid. The neighbors are decided by the size of the filter ex: 3x3. Now multiply each value of the grid by its corresponding value in the filter, and this becomes the new pixel value of the current pixel.
2. Some convolutions will change the image such that certain features of the image are emphasized.
3. **Pooling**: Compressing an image. Convolution + Pooling = Powerful. Simple example of pooling would be, take a pixel and it's 4 neighbors (2 on the right and 1 at down) and then select the largest value amongst those pixel values and then carry this max value forward. This preserves the features highlighted by the convolution while simultaneously quartering the size of the image.
4. **Conv2D**: 2D Convolution layer such as spatial convolution over images. Inherits from Layer, Module.
    a. This layer creates a convolution kernel that is combined with the input layer to produce a tensor of outputs.

   b. If use_bias = True, adds a bias vector to the output and also applies activation function if activation is not None.
   c. When using Conv2D as the first layer in the model, provide the argument, input_shape (tuple of integers). Use None when input dimension is variable an not constant.
5. **MaxPool2D**: Max Pooling operation for 2D spatial data. Inherits from Layer, Module.
   a. This layer down samples the input along its spatial dimensions (height & width) by taking the maximum value over an input window (of size given by pool_size parameter).
   b. The window is shifted by parameter strides along each dimension.
   c. When the padding = "valid", the output shape will be:

$$Outputshape = math.floor((inputshape - poolsize)/strides) + 1$$

   d. And when the padding = "same", the output shape will be:

$$Outputshape = math.floor((inputshape - 1)/strides) + 1$$

6. kernel == filter
7. model.summary(): The method allows you to inspect the layers of model and see the journey of the image through the convolutions.
8. Initially we use a convolution layer and a max polling layer (can be multiple) so that after that when the input is passed to the first layer (Flatten layer), it is already reduced/compressed or simplified and have learned the essential features.
9. If the filter size is 3x3, the output of the convolution will be, (X-dim - 1) pixels reduced from X-axis and (Y-dim - 1) pixels reduced from y-axis. Example; If we use a 3x3 filter on a 28x28 pixel image, the output shape of the convoluted layer would be (3- 1)x (3-1) reduced from each dimension, resulting in 26x26 shape. Similarly, if we use, 5x5 filter, we will be getting output (5-1) smaller on X-axis and (5-1) smaller on Y-axis.
10. Next, the max pooling layer, 2x2, halves the X-axis and the Y-axis. If we have 26x26 from convoluted layer, we would now have 13x13 image shape from the max

pooling layer.

Using Real-Life Images

1. Image Generator: An API of tensorflow. If you point it at a directory, the sub-directories of that will automatically generate labels for you. ImageGenerator class inherits from tf.keras.preprocessing.image.
2. We have to instantiate the image data generator like this:
    a. train_datagen = ImageDataGenerator(rescale = 1./255)
    b. train_generator = train_datagen.flow_from_directory(dir_name)
    c. dir_name = Parent directory that conatins sub-directory that contains the images,
    d. Names of the sub-directories would be the anmes of the labels of your images
    e. When images are loaded, they are resized uniformly.
    f. But you can also resize them here, giving the target_size = (300, 300)
    g. batch_size = Number of images in a input batch.
    h. class_mode = binary or multi-classifier
3. validation_generator is same as data generator except it points it at the validation directory.
4. Optimizer →
    a. Sparse categorical crossentropy → for multi-class classifier
    b. binary_crossentropy → for binary classifier
5. Activation Function for Last layer→
    a. Softmax → Efficient for multi-class classification, If x > 0, return x; else return 0
    b. Sigmoid → Good for binary classifier. Ranges between 0 to 1. If f(x) ≥ 0.5 (threshold), return 1, else 0

$$f(x) = 1/1 + e^{-x}$$

6. For generator, we do not use model.fit(), we use: model.fit_generator(), which takes the following parameters:
    a. train_generator → Generator that you created from the training directory
    b. steps_per_epoch → Total no. of images / batch_size
    c. epochs
    d. validation_data → validation_generator
    e. validation_steps → no. of images in validation set / batch size
    f. verbose → 2 or int