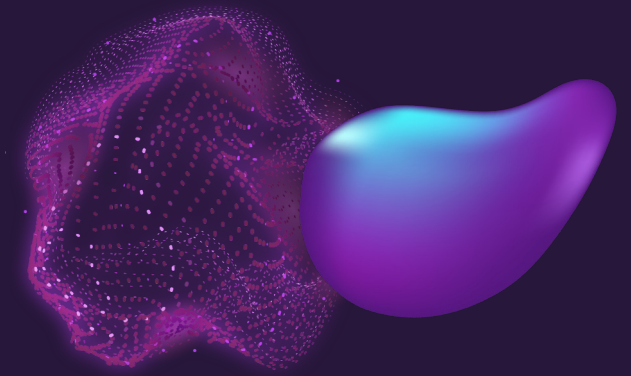


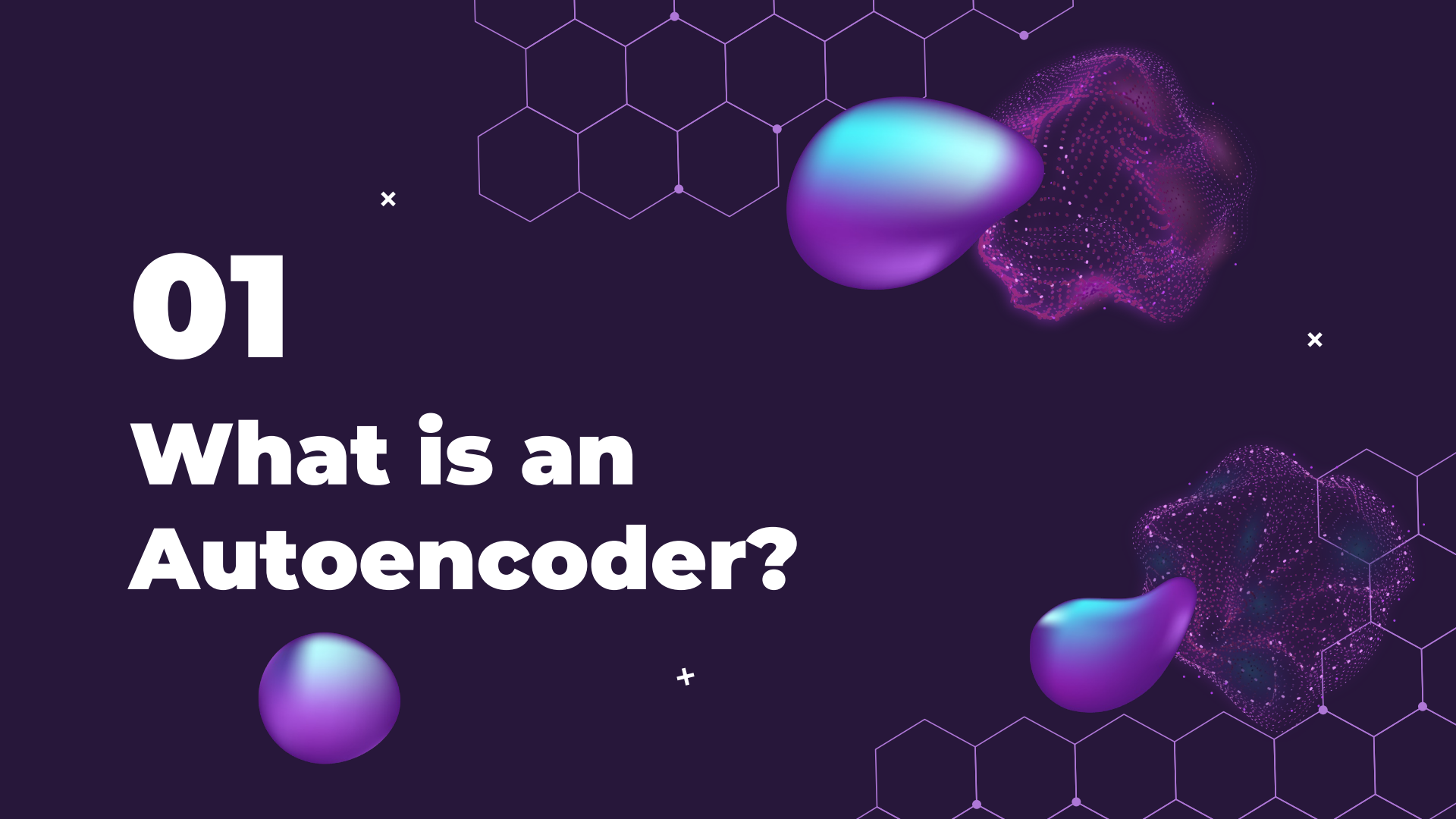
CONVOLUTIONAL AUTOENCODERS

Nitin Gupta

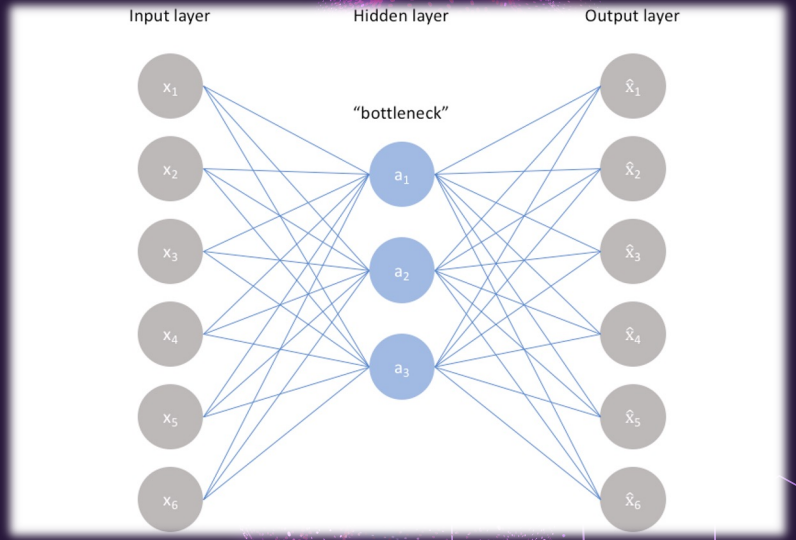
TABLE OF CONTENTS

01

What is an Autoencoder?



- An unsupervised learning technique.
- If the input features are dependent to each other (have some correlations), we can compress and reconstruct the features by imposing a bottleneck.
- We want our autoencoder to be sensitive enough to recreate the original observation but insensitive enough to the training data such that the model learns a generalizable encoding and decoding.



02

What are CNNs?

- Usual neural networks work well on inputs that are structured and have a low number a dimensions.
- However, images are unstructured and often contain a high number of dimensions.
- For example, a colored image of 256 by 256 pixels will have 256*256*3 = 196,608 dimensions as an input. A one-layer neural network will have 196,6082 parameters!
- So, neural networks need to exploit some special characteristics found in images.

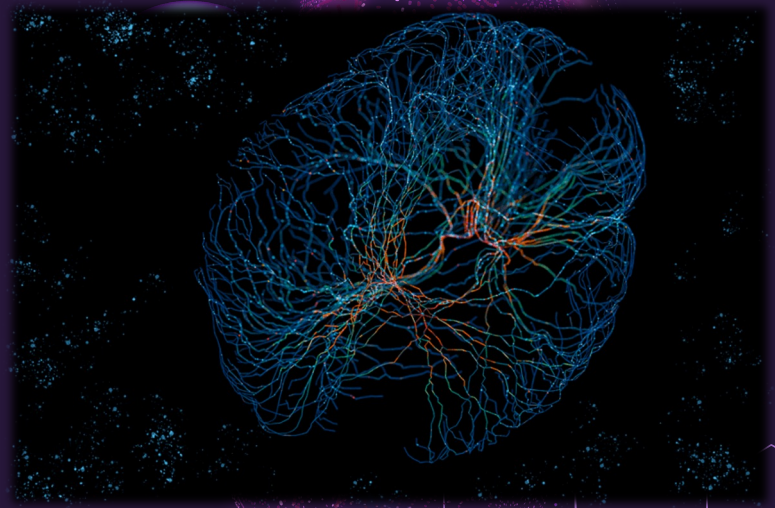


IMAGE FEATURES

Locality

Neighboring pixels look similar or are correlated in some way

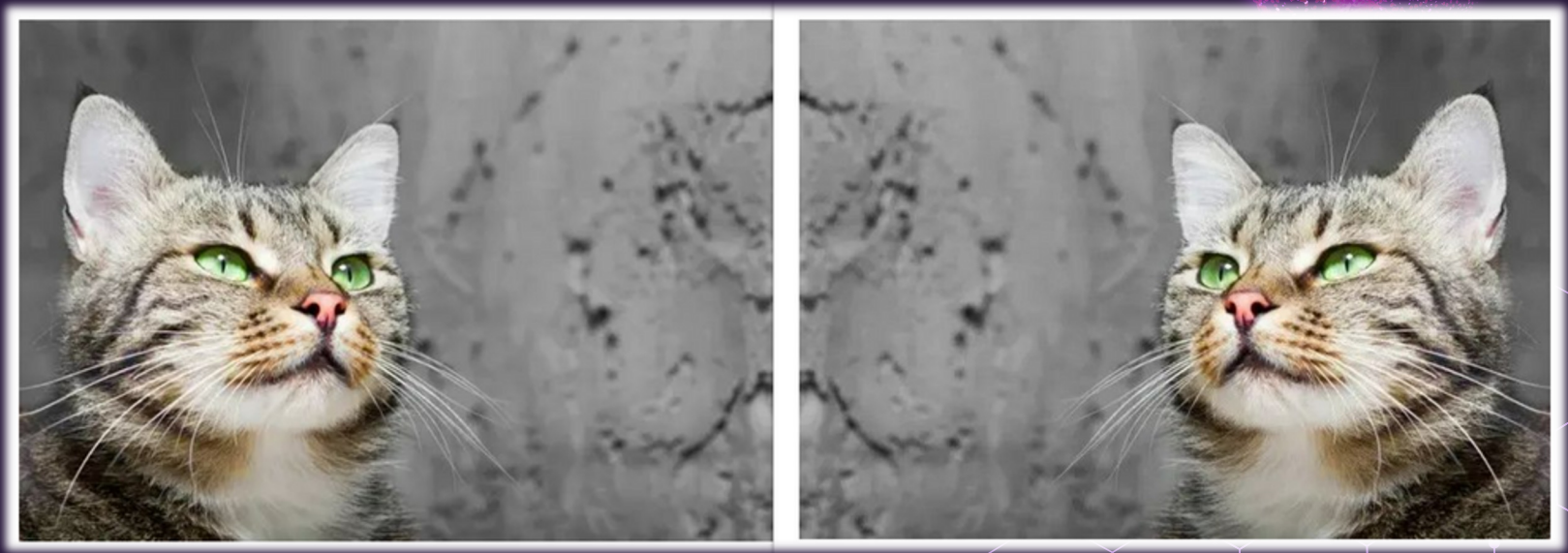
Translation Invariance

An objects' appearance is usually independent of its location.

To incorporate the idea of locality in a neural network, we adjust the operations that take place between layers.

- Instead of calculating the product between neurons, we will rather *utilize rectangular areas*, also known as receptive fields or kernels, that pass over the input while performing the operation of convolution.
- Convolution between the input and a kernel leads to a smaller image in terms of dimensions.

To incorporate the idea of translation invariance, we force the weights of a kernel to be shared across the input.



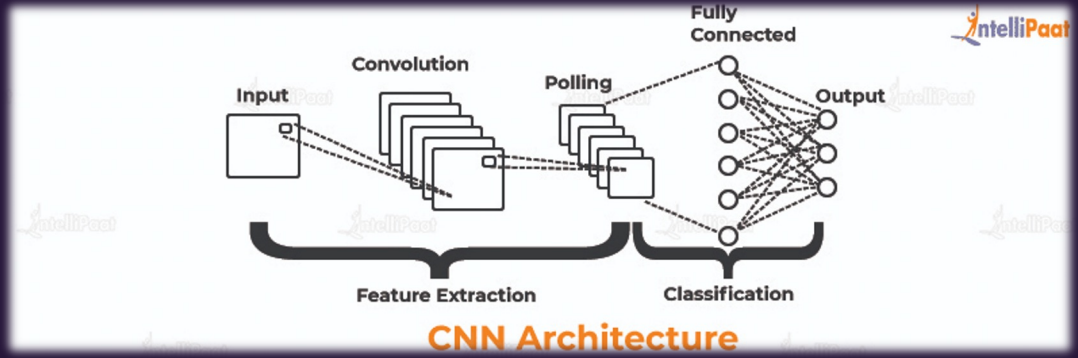
Kernels can be applied to an input in 4 different ways. These are 4 parameters that affect the convolution process.

Padding

Stride

Pooling

Dilation

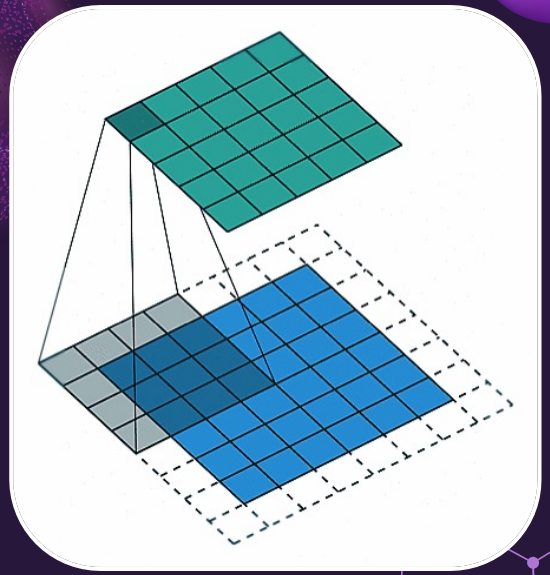


Padding

Padding adds extra pixels around the boundary (usually containing zeros).

By using padding, we can better control the output resolution, because the kernels will pay more attention to the edges of the image (since they will be at the center).

In a sense, padding helps preserve the input resolution.



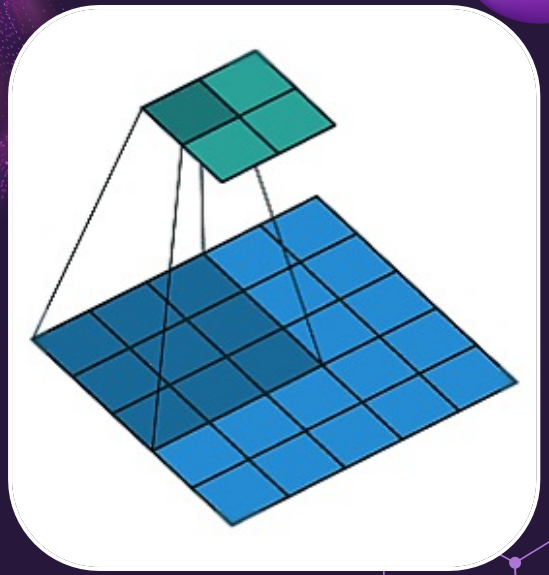
Padding

Strides

The stride handles the step size of the receptive field when applying convolution.

A stride of 1 means the receptive field will be moving 1 step at a time, while a stride of 2 indicates a 2-step movement over the input.

Strides assist to reduce the spatial resolution for better processing.



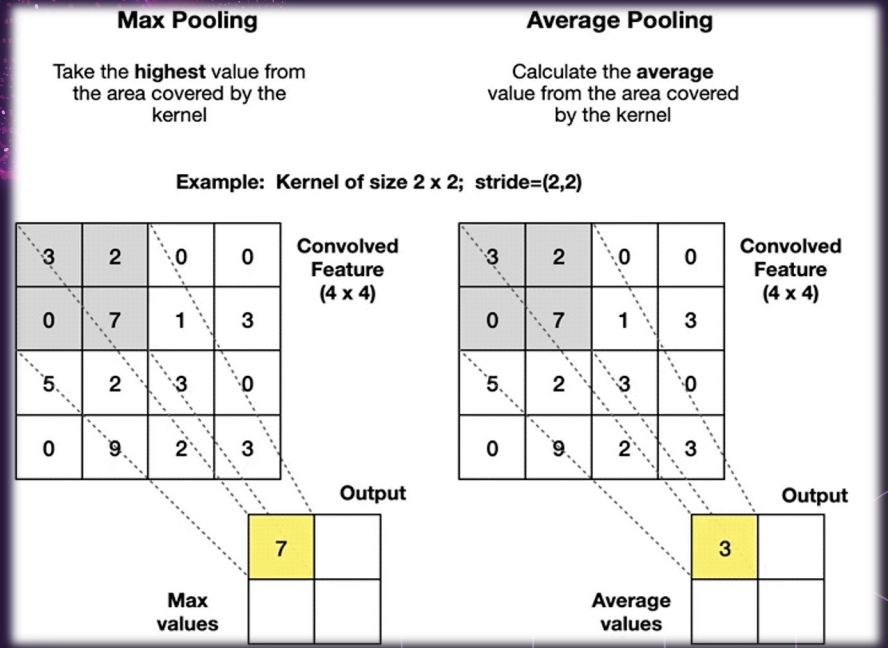
Stride 2 Convolution

Pooling

Convolutional layers produce location-dependent feature maps, making the CNN sensitive to shifts in object positions.

Pooling layers provide "Translational Invariance," making the CNN invariant to small translations in the input.

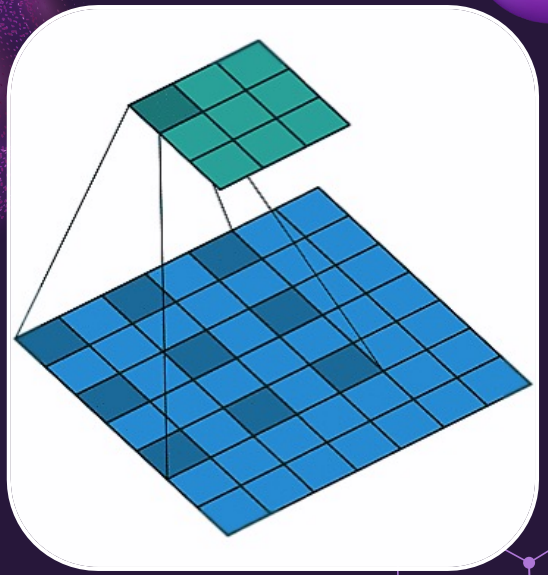
Max pooling highlights prominent features, while average pooling smoothens the image while retaining essential features.



Dilation

The dilation handles the expansion of the receptive field.

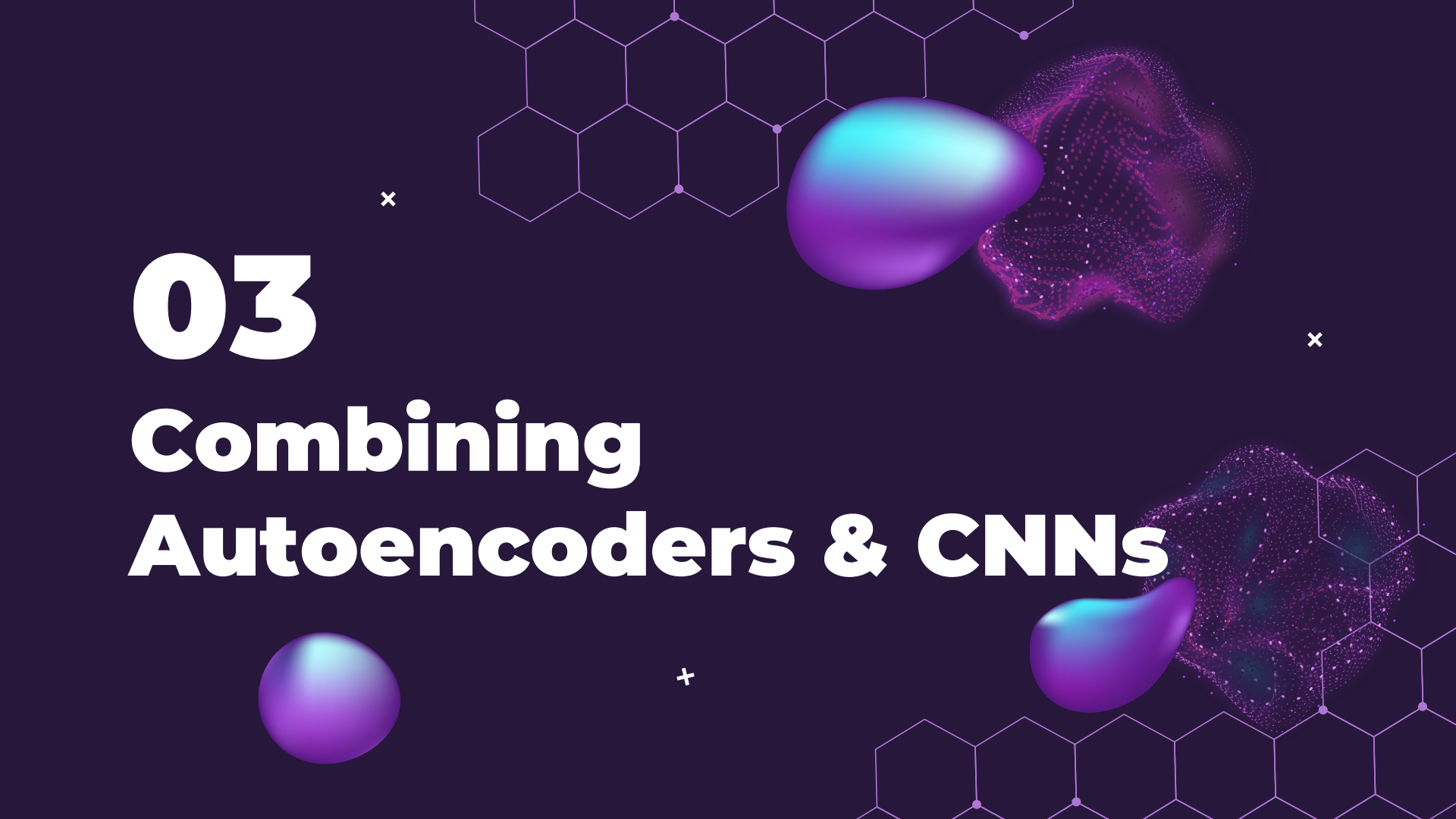
A dilation of 1 means that a 3x3 receptive field will remain 3x3, while a dilation of 2 means that the same field will be transformed into a 5x5 field, since it was expanded by adding “holes” between the weights.



Dilation of 2

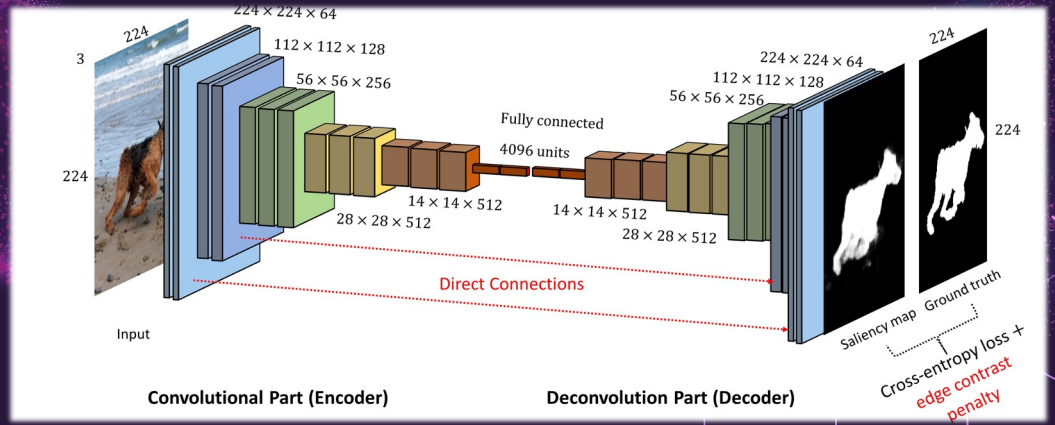
03

Combining Autoencoders & CNNs

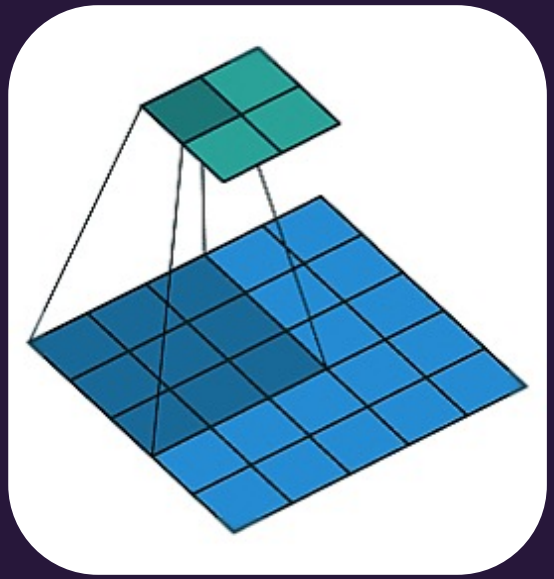


- To combine convolutional layers in an autoencoders, we can replace the encoding portion with a normal CNN.
- As for the decoding portion of the autoencoder, we need transposed convolutions which are the exact opposite of convolutions.

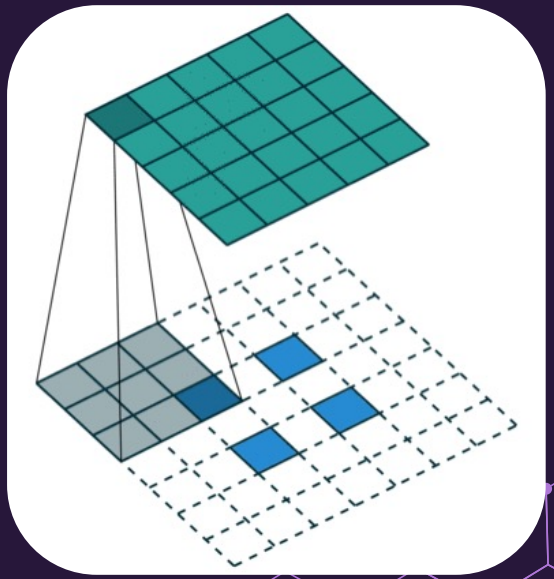
A convolutional layer decreases the resolution, while a transposed one increases it.



An example of strides and their transposed version



Stride 2 Convolution

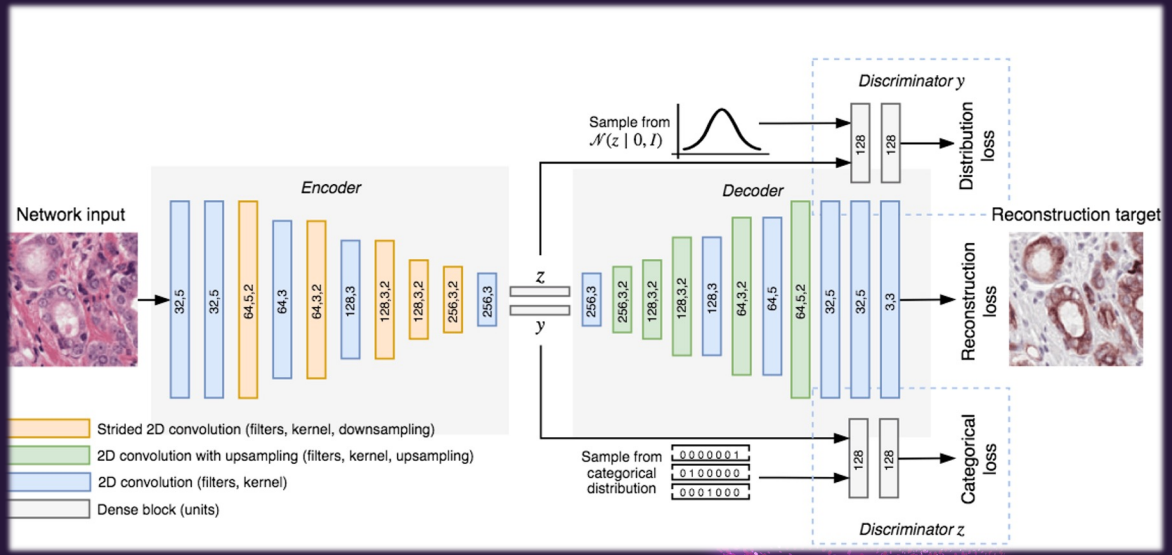


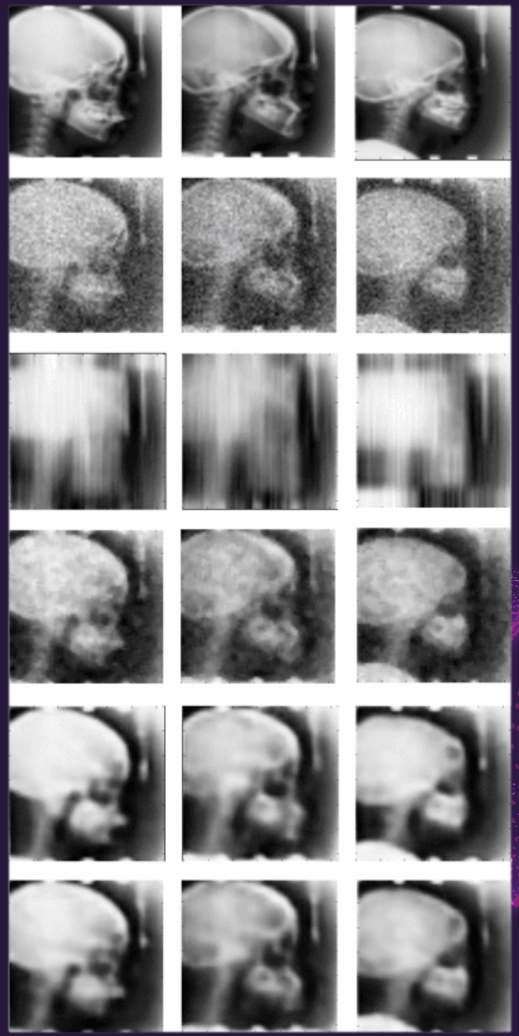
Stride 2 Convolution Transposed

04

Real-World Use Cases

Cancer Detection





Medical Image Denoising

05

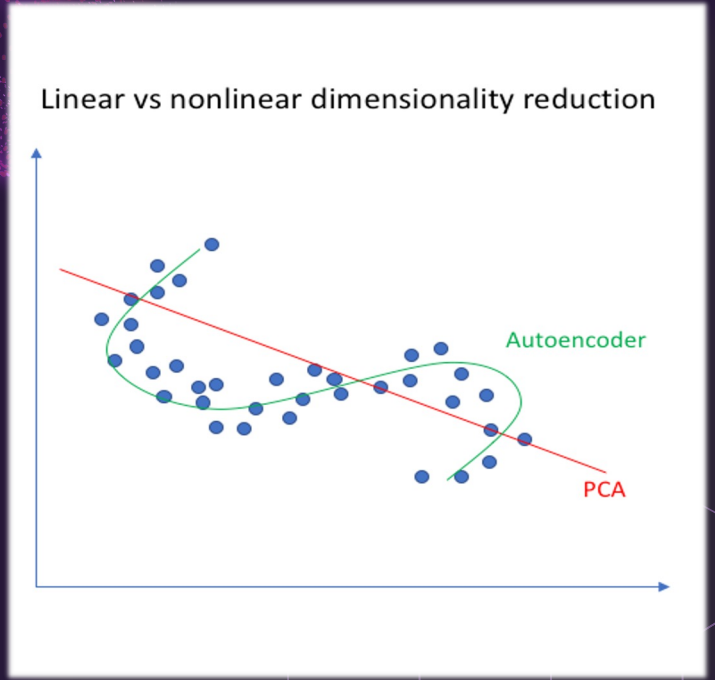
Applications

vs. PCA

Recall that Principal Component Analysis (PCA) uses linear algebra to transform and reduce dimensions.

In contrast, the autoencoder techniques can perform non-linear transformations with their non-linear activation function and multiple layers.

Thus, autoencoders show their merits when the data problems are complex and non-linear.



vs. PCA

A paper by Hinton and Salakhutdinov (2006) showed that training an autoencoder yielding a more minor errors compared to the first 30 principal components of a PCA and a better separation of the clusters.

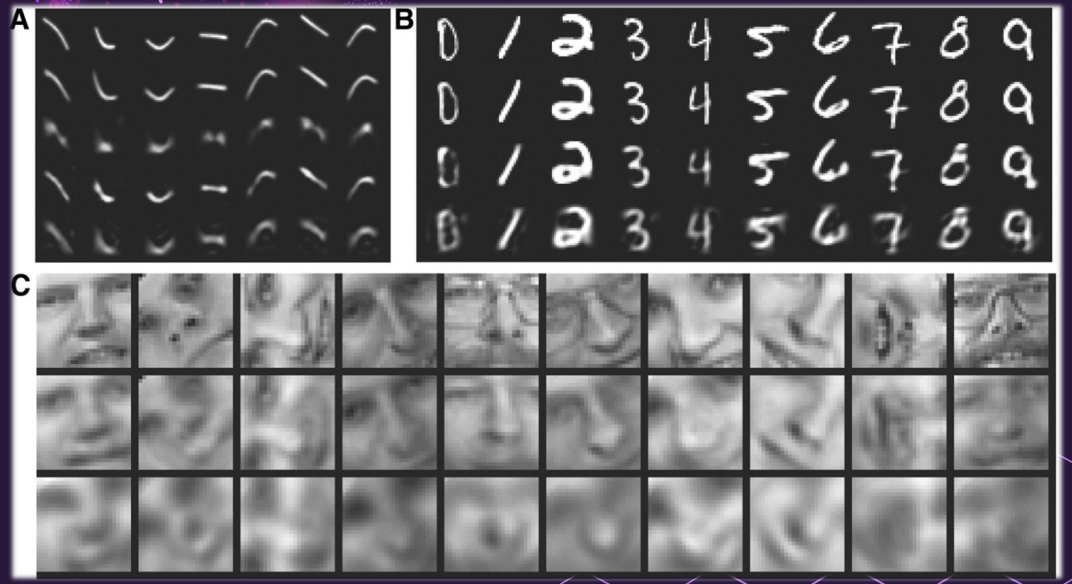
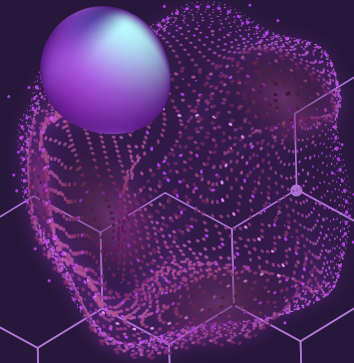


Image Coloring



But Why?

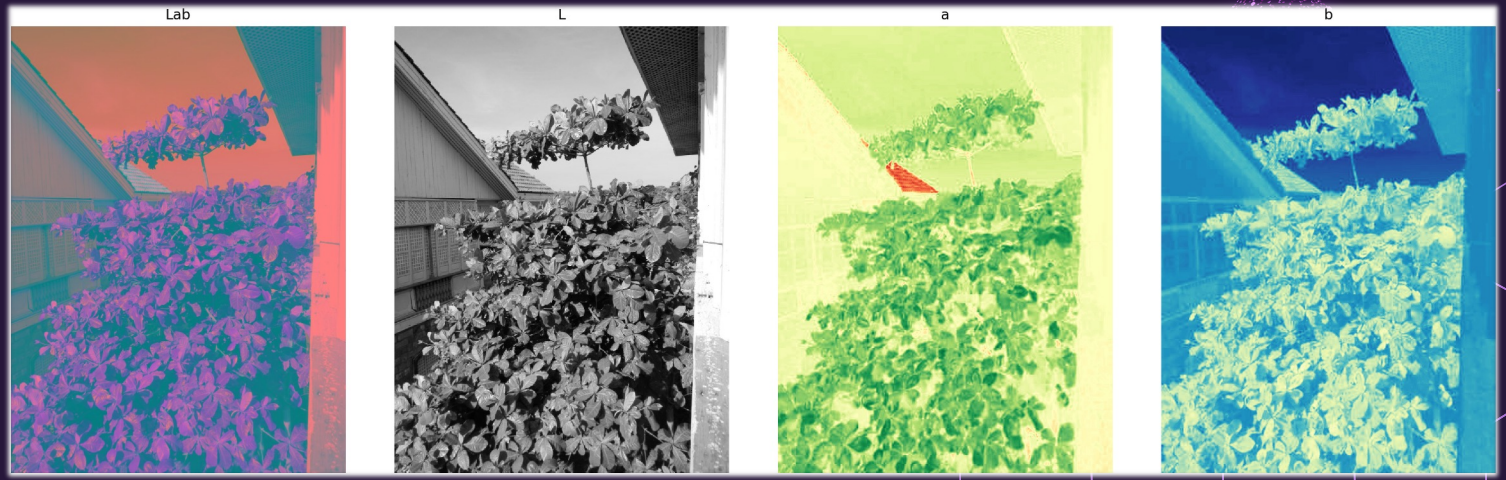
- Image colorization traditional often involved a large amount of human effort.
- This problem is challenging because it is multimodal -- a single grayscale image may correspond to many plausible colored images. As a result, traditional models often relied on significant user input alongside a grayscale image.
- Furthermore, image colorization is not only about adding realism but also about creating aesthetically pleasing visualizations. Artists and designers may use autoencoders to experiment with different color schemes and styles.

- In image colorization, the goal is to build a model capable of applying realistic color to black and gray images.

- However, since the input prompt is a black and gray image, we do not want to strictly recreate it, but enhance it in a sense.

- Generally, images have the shape of (channels, width, height). The input images (black and white) contain only 1 channel, but their colorized versions (in RGB) contain 3 channels: red, green, and blue.

- So, the autoencoder is deemed to generate 2 channels in total.
- To simplify this, instead of using the RGB format, we can utilize the *LAB format*, where *L* indicates the lightness, *a* is the balance between green and magenta, and *b* is the balance between blue and yellow. Although *LAB* uses the same number of elements to illustrate a pixel, the *L* part is essentially our input image.
- In other words, by using the LAB format the autoencoder will generate only 2 channels, a and b, considering that the third one, the L, will come from the input.



Data and Preprocessing

- Before proceeding with the actual implementation, we need a large dataset containing colorized images. The chosen dataset is the *Google Landmark* Dataset containing approximately 5 million colorized images about human-made and natural landmarks.

- The dimensions of the images vary, so only images of size 600x800x3 were chosen.

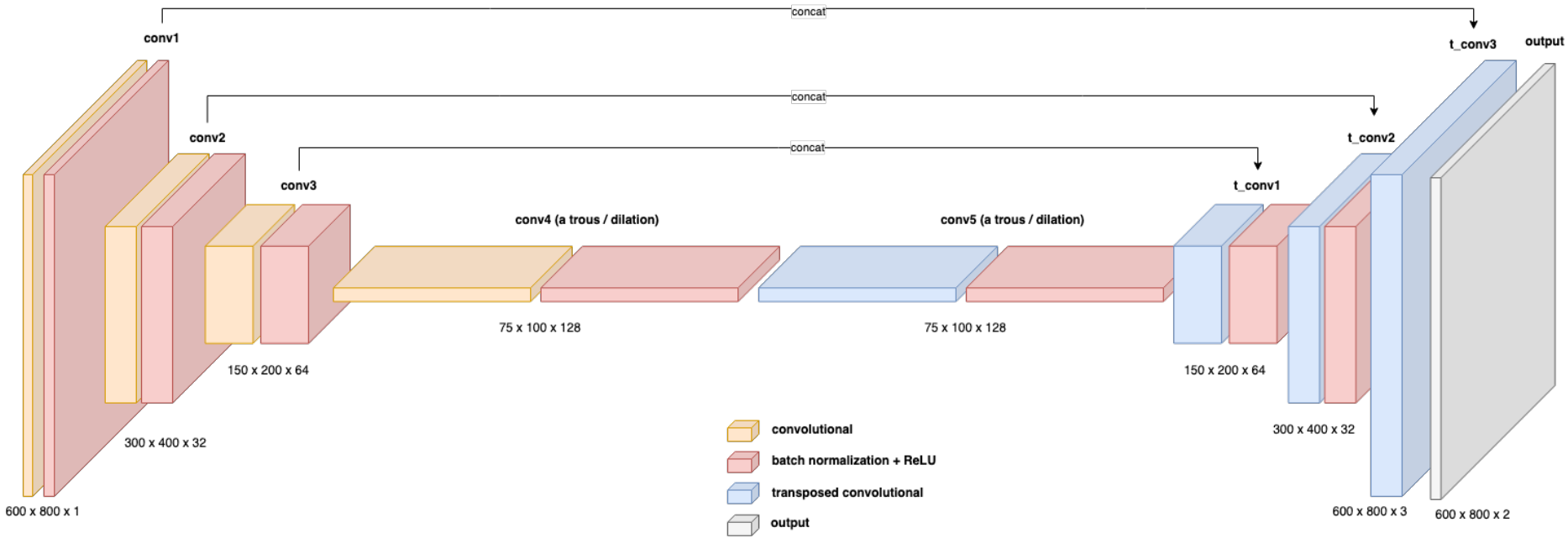
- The only preprocessing that took place was the conversion of RGB to LAB format.

Architecture and Configuration

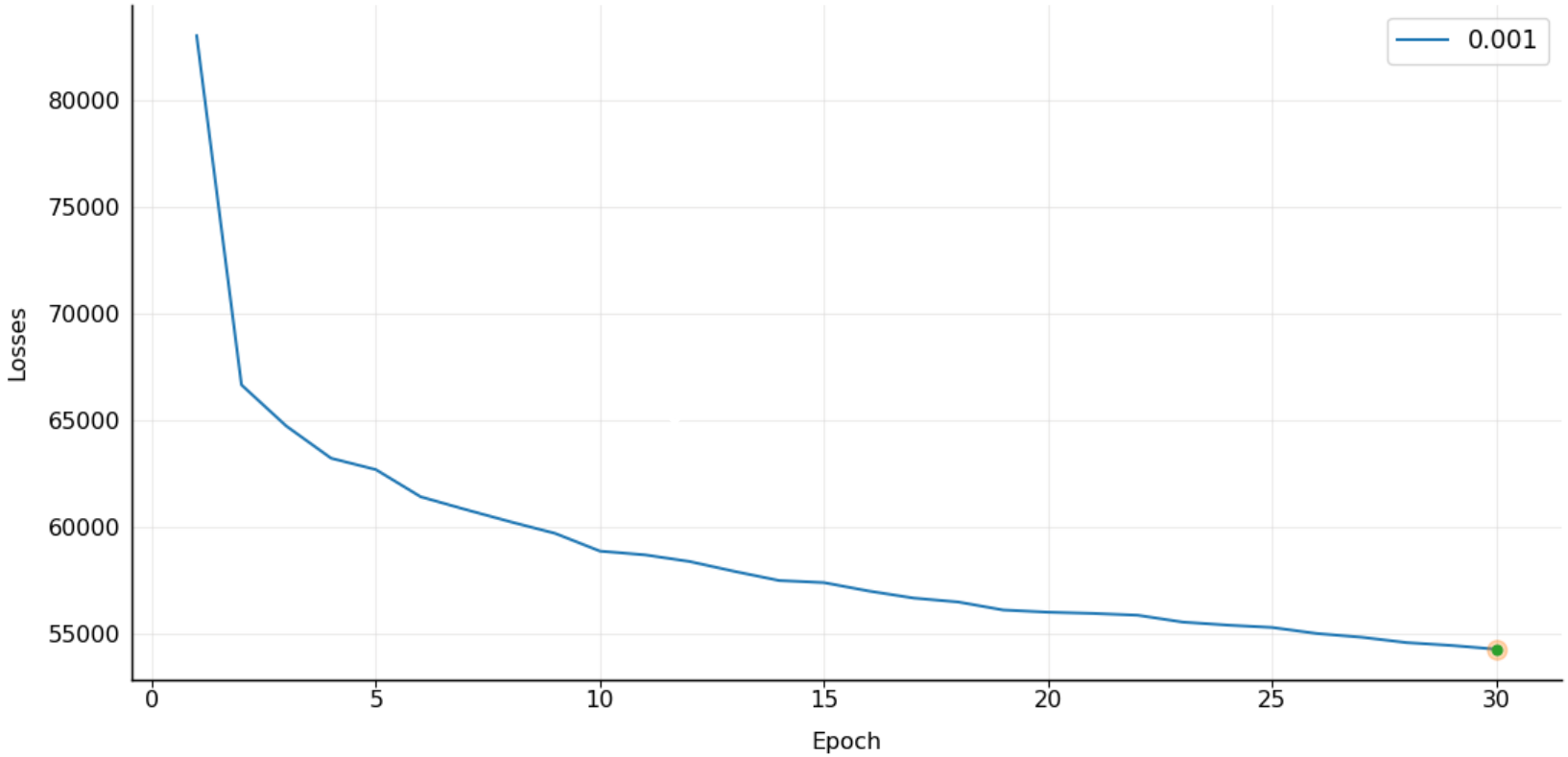
- The encoder will consist of convolutional layers and the decoder will contain transposed convolutional layers.
- The input will be images of 600x800 with 1 channel, the L value (lightness). In the output, we will get a 600x800 image with 2 channels, the a and b values. The final colorized image will be constructed by combining the predicted a and b with the input L.

- Activation function: ReLU
- Optimizer: Adam
- The batch size was set to 20, which means training the network with batches containing 20 images combined.
- Epochs : 30
- Number of parameters of the architecture: 889082
- Training on around 10,000 (initially) and 28,000 images (final version).
- Using Python, Pytorch, and Scikit-Image

Architecture



Model Loss Per Epoch



Learning Rate = 0.001

Input

Actual

Predicted

Loss: 0.00310 Loss: 0.00128 Loss: 0.00153 Loss: 0.00432

Training Set

Input

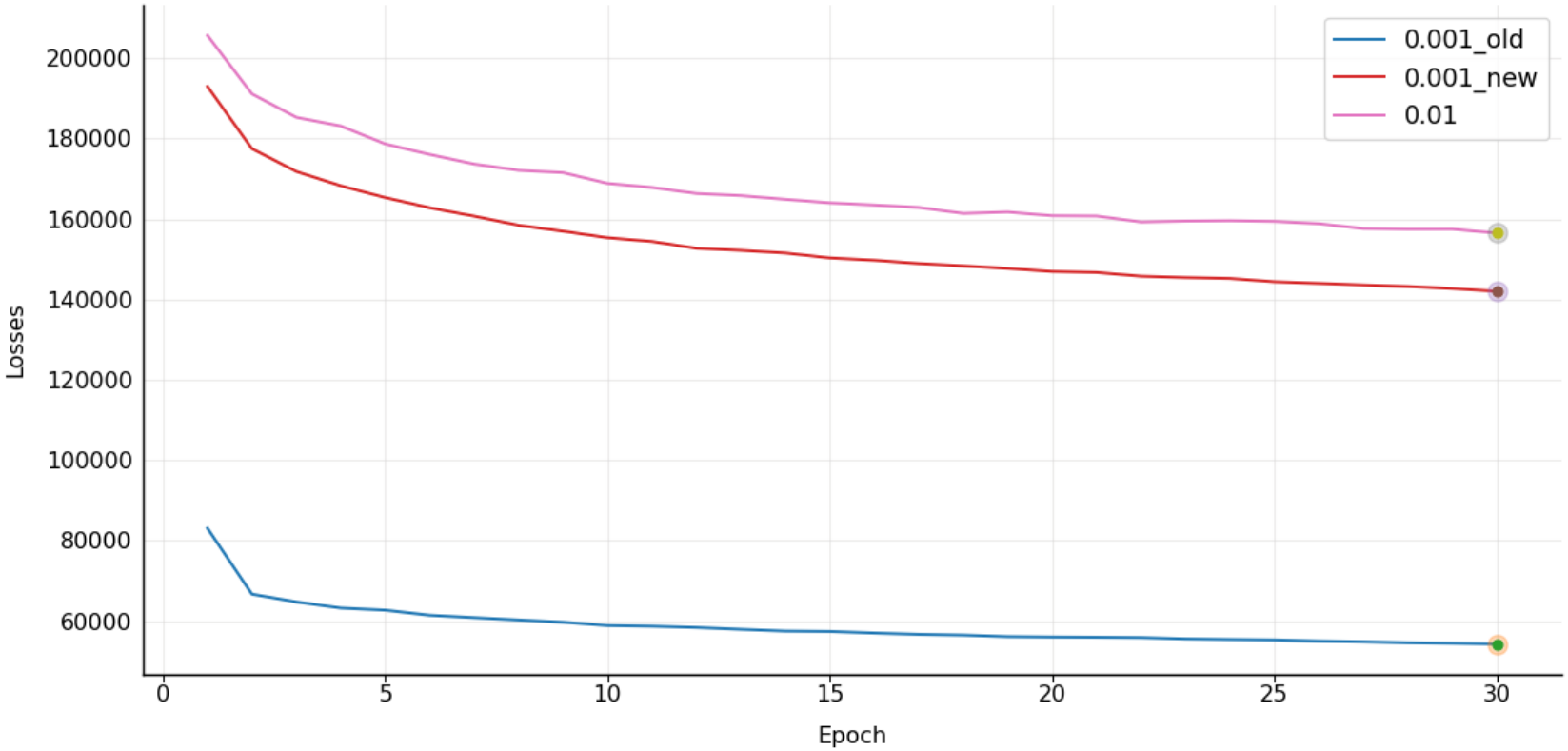
Actual

Predicted

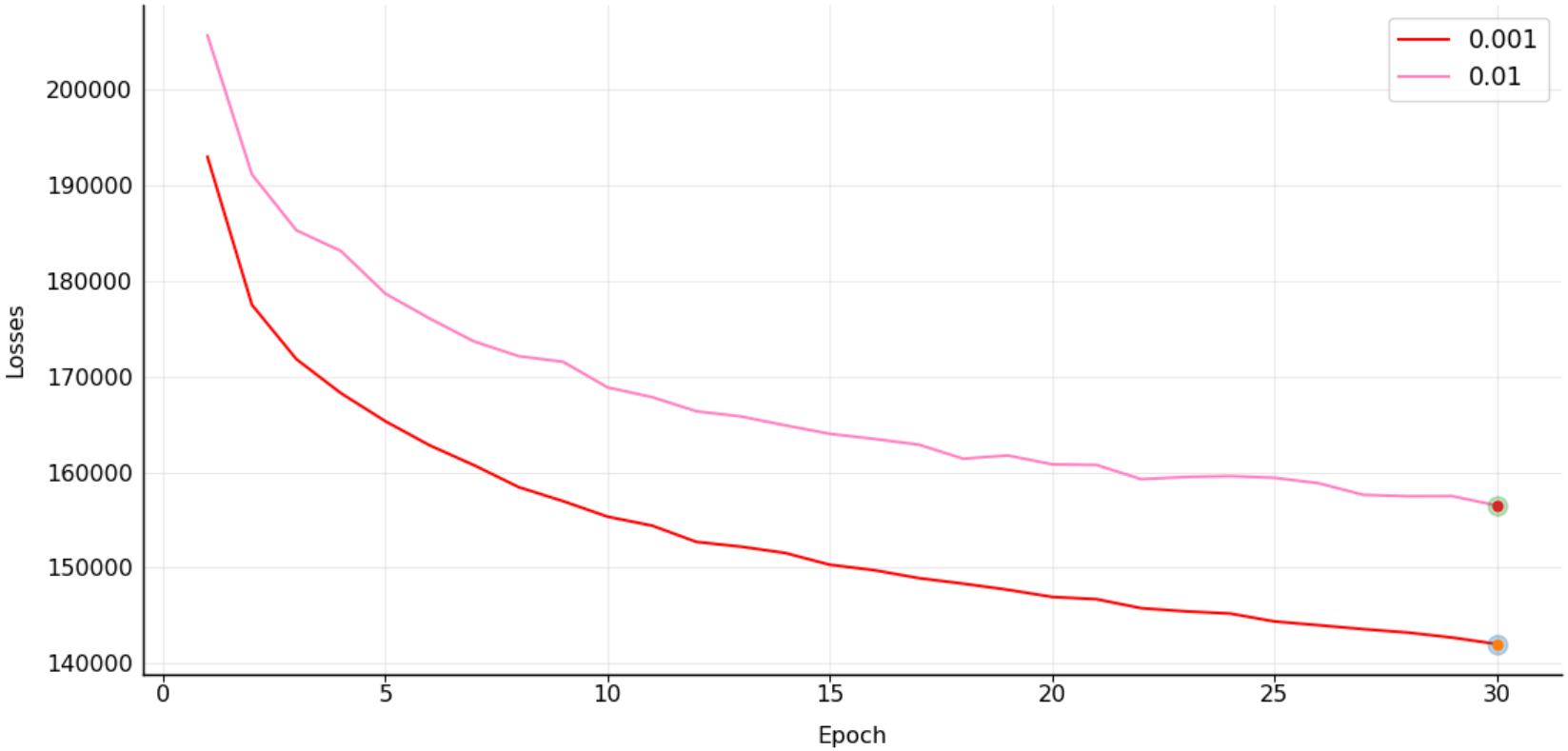
Loss: 0.00084 Loss: 0.00476 Loss: 0.01391 Loss: 0.00056

Test Set

Model Loss Per Epoch



Model Loss Per Epoch



Learning Rate = 0.01

Input

Actual

Predicted

Loss: 0.00399 Loss: 0.00367 Loss: 0.00204 Loss: 0.00111

Training Set

Input

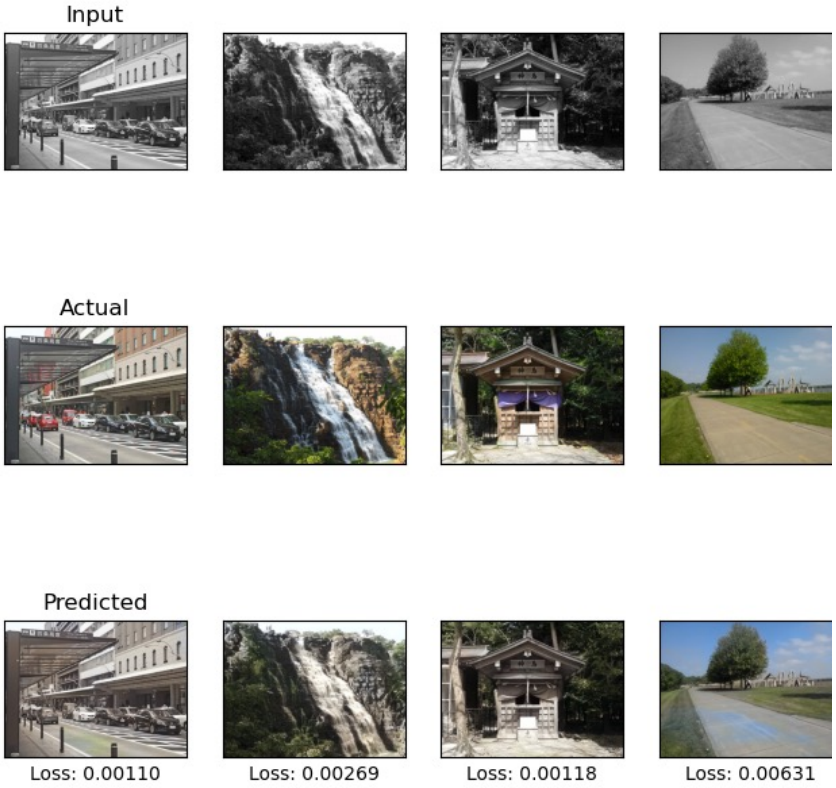
Actual

Predicted

Loss: 0.00716 Loss: 0.00151 Loss: 0.00270 Loss: 0.00185

Test Set

Learning Rate = 0.001 (new)

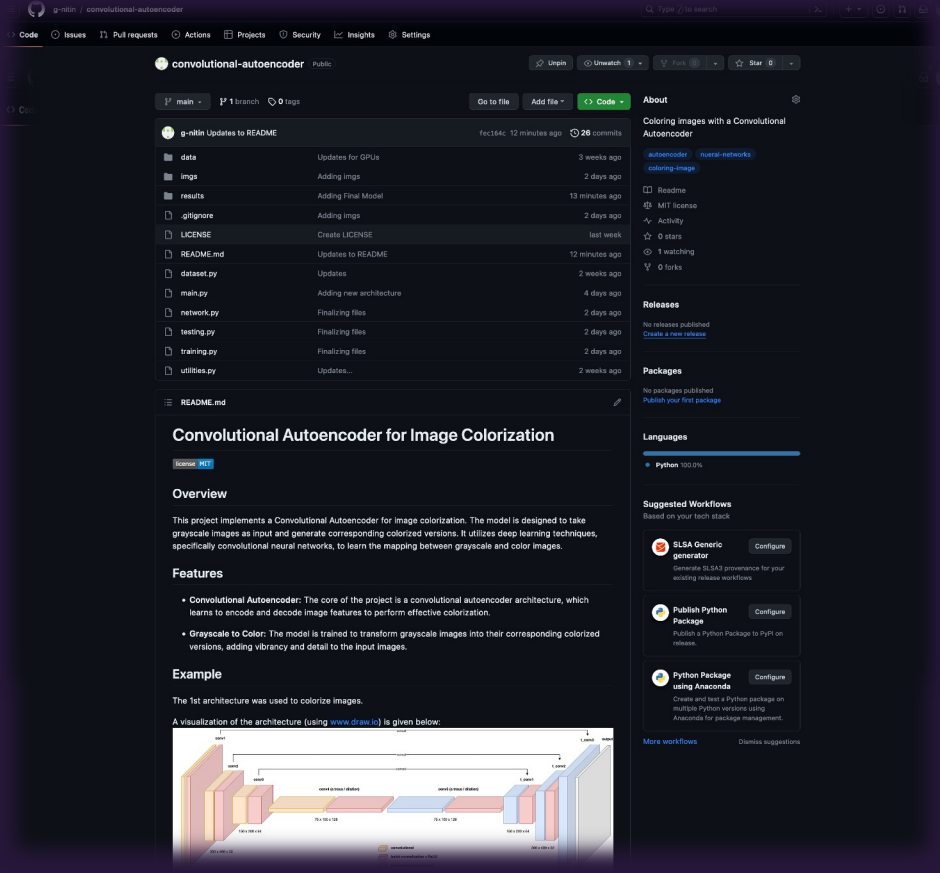


Further Improvements & Steps

- More training images.
- Testing with other architectures.
- Varying epochs.
- Varying batch sizes.
- Use GPUs!
- Apply on videos.

Example of Coloring a Video

Acknowledgements



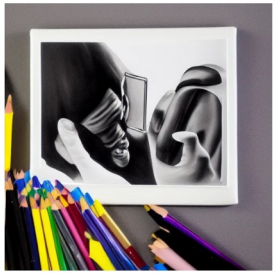


Image generated by Stable Diffusion

Building an Image Colorization Neural Network — Part 4: Implementation

George Kamtziridis · Follow

Published in MLearning.ai · 8 min read · Sep 19, 2022

Information Sources

- github.com/gkamtzir/cnn-image-colorization
- github.com/g-nitin/convolutional-autoencoder
- www.jeremyjordan.me/autoencoders/
- www.medium.com/mlearning-ai/building-an-image-colorization-neural-network-part-1-generative-models-and-autoencoders-d68f5769d484
- www.ieeexplore.ieee.org/document/7836672
- https://www.wouterbulten.nl/posts/unsupervised-cancer-detection-using-deep-learning-adversarial-autoencoders/
- www.towardsdatascience.com/anomaly-detection-with-autoencoder-b4cdce4866a6
- www.science.org/doi/10.1126/science.1127647
- https://xiangyutang2.github.io/auto-colorization-autoencoders/

Image Sources

- www.intellipaat.com/blog/tutorial/artificial-intelligence-tutorial/convolution-neural-network/
- www.blog.paperspace.com/pooling-and-translation-invariance-in-convolutional-neural-networks/
- www.towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1
- www.pub.towardsai.net/introduction-to-pooling-layers-in-cnn-dafe61eabe34
- www.github.com/vdumoulin/conv_arithmetic/tree/master/gif
- www.github.com/arthurmeyer/Saliency_Detection_Convolutional_Autoencoder
- www.github.com/PacktPublishing/Advanced-Deep-Learning-with-Keras/tree/master/chapter3-autoencoders
- https://commons.wikimedia.org/w/index.php?curid=34991651
- https://commons.wikimedia.org/wiki/File:PyTorch_logo_icon.svg
- https://images.app.goo.gl/jtFvCod4FXQGVXy36
- https://www.youtube.com/watch?v=LluZarKPY-o

THANKS!

ANY QUESTIONS?