Open in app ↗

Search Medium

This member-only story is on us. Upgrade to access all of Medium.

✨ Member-only story

Using PCA to Reduce Number of Parameters in a Neural Network by 30x Times

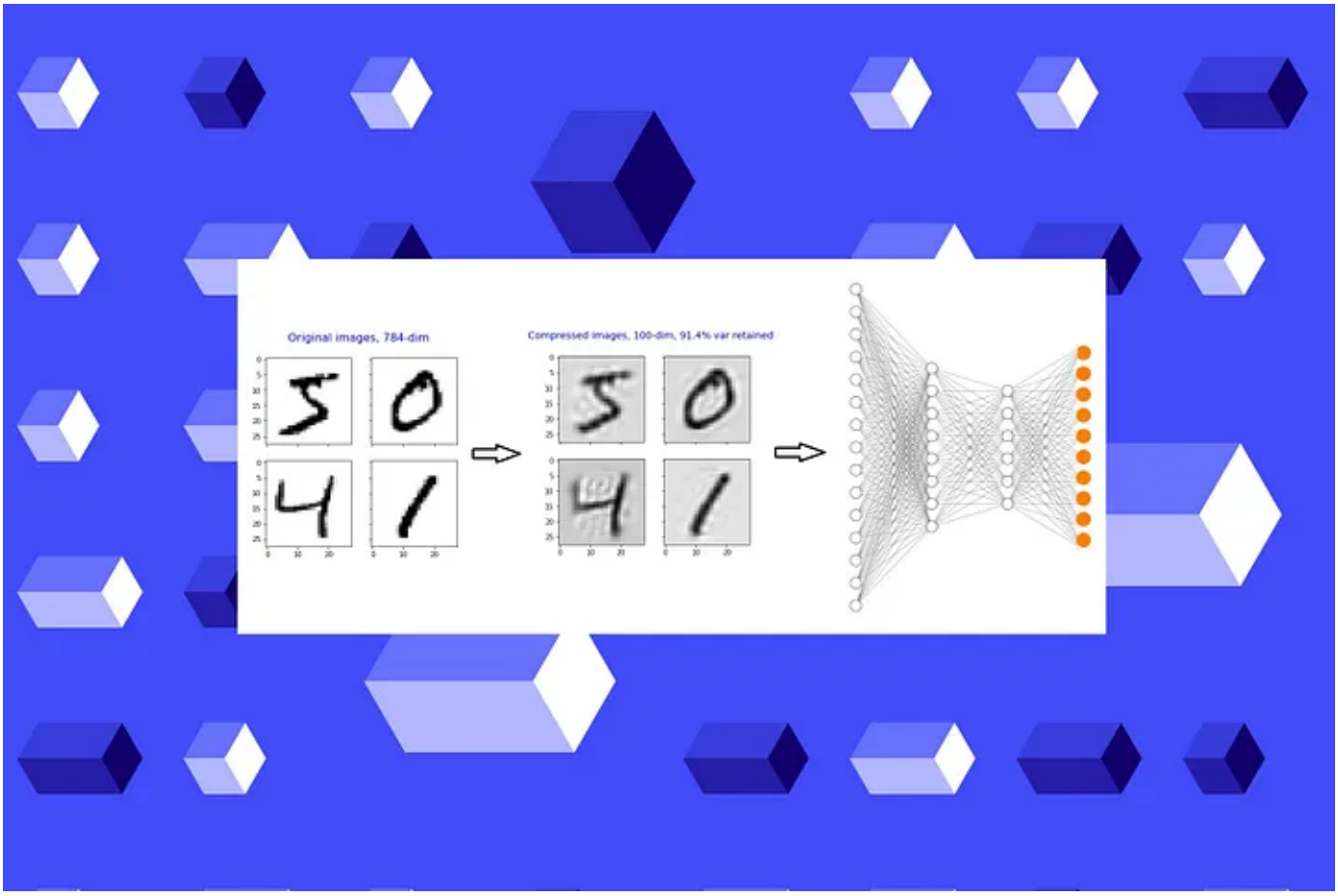
While still getting even better performance! — Neural Networks and Deep Learning Course: Part 17

Rukshan Pramoditha · Follow

Published in Towards Data Science

7 min read · Jun 12, 2022

Listen Share ••• More

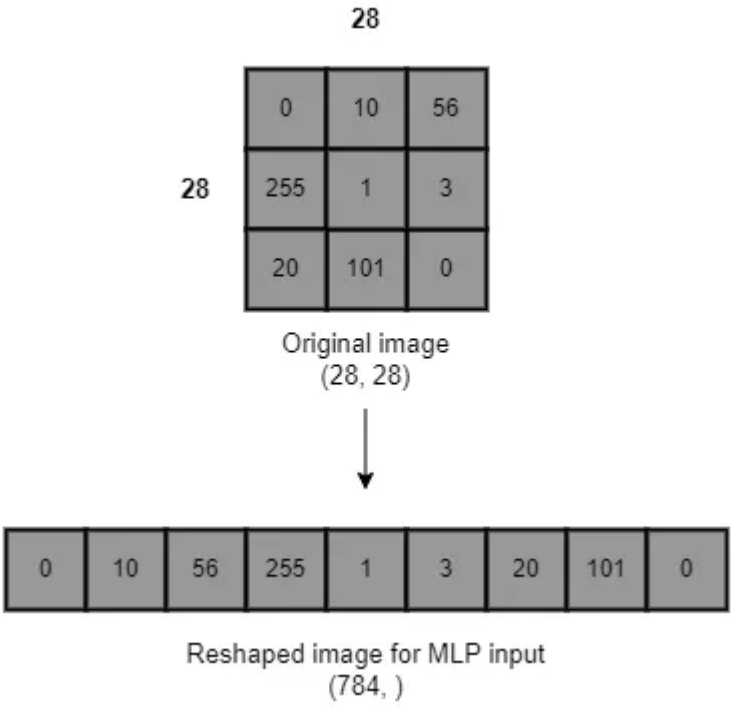


Original photo by MagicPattern on Unsplash, edited by author

In the previous article, we created a Multilayer Perceptron (MLP) classifier model to identify handwritten digits.

We used two hidden layers with 256 neurons for the network architecture. Even with such a small network, we got 269,322 total parameters (weights and bias terms). The main reason for getting such a big number of parameters for the network is the large size of the input layer.

Because the input layer of an MLP takes 1D tensors, we need to reshape 2-dimensional MNIST image data into 1-dimensional data. This process is technically called ***flattening*** the image.



(Image by author, made with draw.io)

Each pixel in the image represents an input. If there are 784 pixels in the image, we need 784 neurons in the input layer of the MLP. When the size of the input layer increases, we get a large number of total parameters in the network. That's why MLPs are not parameter efficient. The size of the input layer significantly increases when we use high-pixel value image data, for example, 500 x 500 px images.

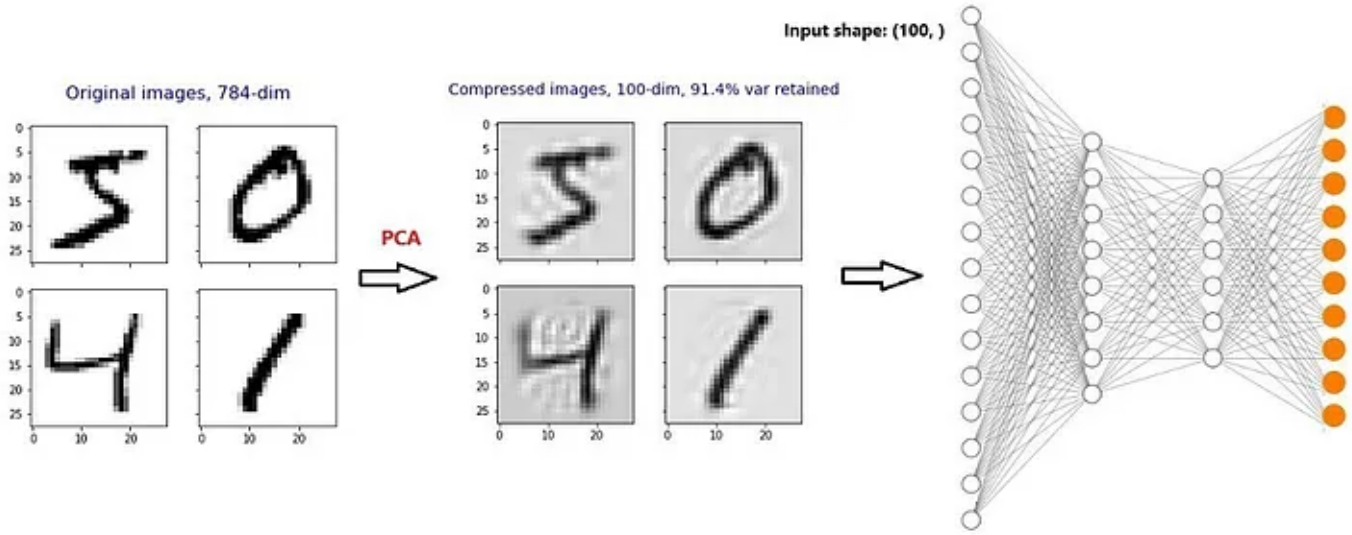
The problem with getting a large number of total parameters in the network is that we need a lot of computational resources to train the neural network and it is also time-consuming.

MLPs are not suitable for image data. There is an even better neural network architecture called **Convolutional Neural Network (CNN or ConvNets)** designed to work with image data. Its input layer can take high-dimensional data so we do not need to flatten the images.

We can still use MLPs with image data efficiently if there is a way to reduce the number of pixels in the image without losing much of the image quality. If we can do it, the size of the input layer will be significantly small and the MLP model will be parameter efficient.

We can apply **Principal Component Analysis (PCA)** to the flattened images to significantly reduce the number of pixels while still getting almost the same image quality but with fewer pixels now!

The general workflow is:



(Image by author, made with draw.io)

First, we apply PCA to the MNIST digits data and reduce dimensionality (number of pixels) by 7.48x times while keeping 91.4% image quality! Then, we build the same MLP model that we built in Part 16. Now, the input shape is (100,), not (784,). This will reduce the number of total parameters in the network significantly. Moreover, we have the opportunity to decrease the number of neurons in the hidden layers because the input size is small now. That will also help to reduce the number of total parameters in the network significantly.

Apply PCA to the MNIST data

If you acquire the MNSIT data through the Keras API, you will get 60,000 grayscale images of handwritten digits for the training set and 10,000 grayscale images of handwritten digits for the testing set. Each grayscale image is 2-dimensional with 28 px width and 28 px height.

```
Train images shape: (60000, 28, 28)

Test images shape:  (10000, 28, 28)
```

(Image by author)

We need to flatten these images before applying PCA. After flattening, we get:

```
Train images shape: (60000, 784)
Test images shape:  (10000, 784)
```

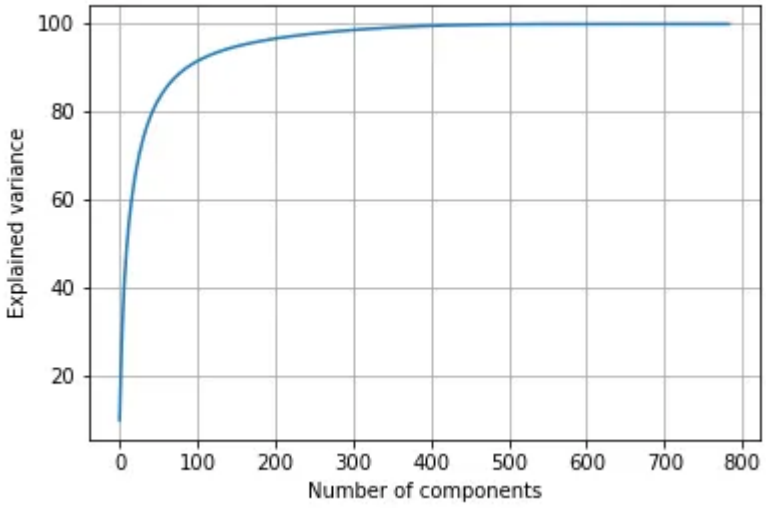
(Image by author)

We can think of this new array of training set images as a simple dataset with 60,000 rows (number of images) and 784 columns (number of pixels in an image). Now, we can apply PCA as usual. Note that we apply PCA to both train and test sets separately.

When considering a single flattened image, the number of pixels represents its dimensionality. When we reduce the dimensionality, we reduce the number of pixels in the image.

Apply PCA to the MNSIT data to select the best number of principal components (Code by author)

After running the above code, we get the following output.



Select the best number of principal components (Image by author)

We can see that the first 100 components capture about 90% of the variability in the original image data. That is enough to retain the quality of the original images. So, we select the first 100 components and apply PCA again with that number of selected components.

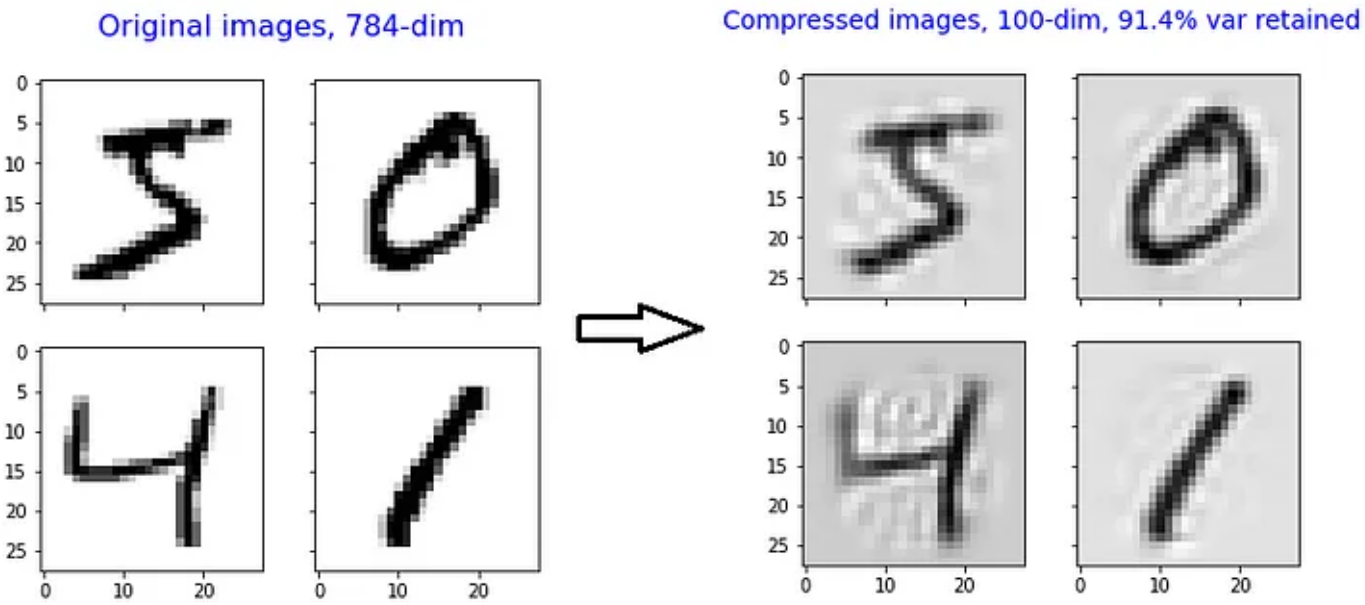
Apply PCA again with the selected components (Code by author)

```
Train images shape: (60000, 100)
Test images shape:  (10000, 100)

Var retained (%): 91.431076
```

(Image by author)

We have reduced the dimensionality of the original image data by 7.84x times (784/100) while keeping 91.43% variability in the original data.



A sample of images: Before and after applying PCA (Image by author)

The compressed images are still identifiable!

Note: I'm not going to explain more about the PCA process here. Kindly refer to the following article series (written by me) to learn everything about PCA and dimensionality reduction.

Click on this image to access my PCA and Dimensionality Reduction Special Collection (Screenshot by author)

Build the MLP classifier with compressed (reduced) image data

Now, we build the same MLP classifier model as we built in Part 16, but with compressed image data (fewer dimensions and fewer pixels)!

(Code by author)

```
Test loss: 0.10373429208993912
Test accuracy: 0.9745000004768372
```

(Image by author)

This time we get an accuracy score of 0.9745. Previously (i.e. before applying PCA), we got an accuracy score of 0.9804. We only lose a very small amount of accuracy score. That is 0.0059.

If we check the number of total parameters now, we get:

```
MLP.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                6464

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 10)                330

=================================================================
Total params: 8,874
Trainable params: 8,874
Non-trainable params: 0
```

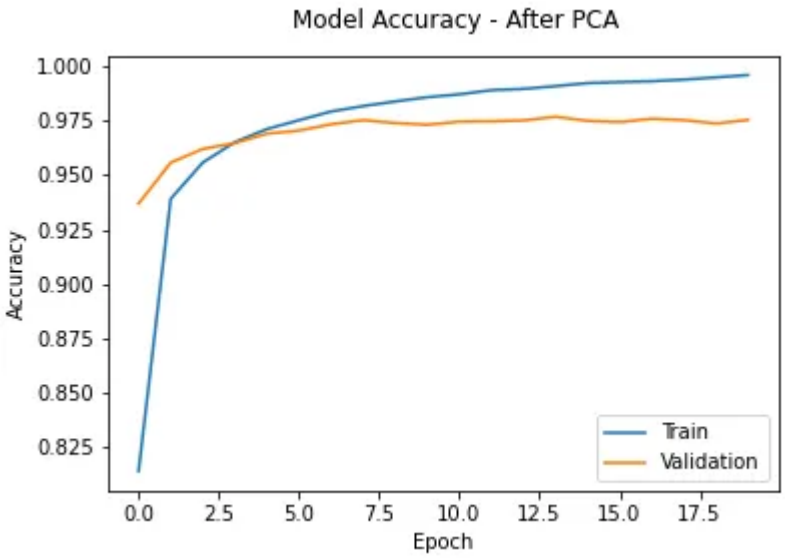
Number of trainable parameters after applying PCA (Image by author)

Previously (i.e. before applying PCA), we got 269,322 total parameters. This time we only got 8,874 total parameters. So, we've reduced the number of total parameters by 30x times (269,322 / 8,874) after applying PCA! We only lost 0.0059 model accuracy.

We can get even better results by applying PCA. This time I'm talking about the problem of overfitting.

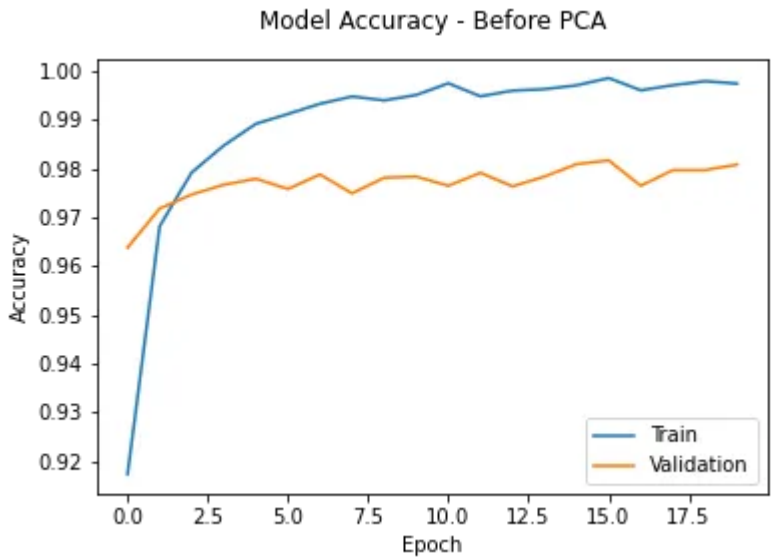
I measured the performance of the model during the training.

(Code by author)



(Image by author)

This can be compared with the corresponding previous case: before applying PCA.



(Image by author)

It is very clear that the ***model is less overfitting after applying PCA***. The model generalizes well on the new unseen data (test data). The reason is that:

> *PCA removes the noise in the data and keeps only the most important features in the dataset. That will mitigate the overfitting of the data and increase the model's performance on unseen data. In addition to that, a model can become more complex when there are many features in the data. Complex models tend to overfit the data. PCA takes care of model complexity by reducing the number of features (dimensionality) in the data — Source:* ***How to Mitigate Overfitting with Dimensionality Reduction*** *(my own article)*

Summary

- We applied PCA to the MNIST data and successfully reduced the dimensionality of the original image data by 7.84x times. So, we could significantly reduce the size of the MLP input layer. We kept 91.43% variability in the original data. So, images are still identifiable.

- We only lost a very small amount of accuracy after applying PCA.

- We could reduce the number of total parameters in our MLP model by 30x times! This is because we significantly reduced the size of the MLP input layer and the number of neurons in the hidden layers. Because the input size is small now, we have the opportunity to decrease the number of neurons in the hidden layers. You will not get much performance improvement by increasing the number of neurons

in the hidden layers. However, it is better to do some experiments by setting different values and then measuring the performance.

- After applying PCA, the model is less overfitting. I explained the reason for that. Also, note that the model is still overfitting because we haven't applied any regularization technique to the model yet.
- It is easy to apply PCA for MNIST image data because the images are grayscale which are represented as two-dimensional (2D) tensors. The PCA process is complicated with RGB images which are represented as three-dimensional (3D). Read this article to learn how to apply PCA to an RGB image. To learn the difference between RGB and grayscale images, read this article.

This is the end of today's post.

Please let me know if you've any questions or feedback.

I hope you enjoyed reading this article. If you'd like to support me as a writer, kindly consider ***signing up for a membership*** *to get unlimited access to Medium. It only costs $5 per month and I will receive a portion of your membership fee.*

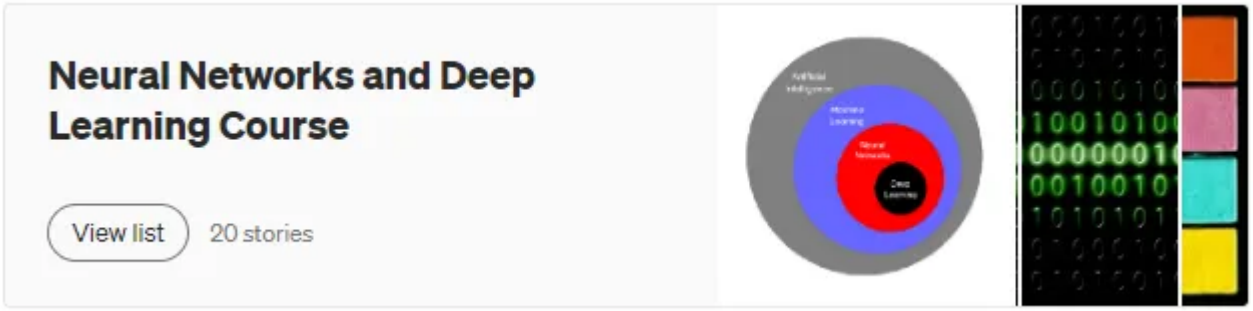
Join Medium with my referral link - Rukshan Pramoditha

As a Medium member, a portion of your membership fee goes to writers you read, and you get full access to every story...

rukshanpramoditha.medium.com

Thank you so much for your continuous support! See you in the next article. Happy learning to everyone!

Join my neural network and deep learning course



Click on this image to access my Neural Networks and Deep Learning Course (Screenshot by author)

Rukshan Pramoditha

2022–06–12

Artificial Intelligence | Deep Learning | Neural Networks | Multilayer Perceptron | Pca

Follow

Written by Rukshan Pramoditha

6.1K Followers · Writer for Towards Data Science

2,000,000+ Views | BSc in Stats | Top 50 Data Science, AI/ML Technical Writer on Medium | Data Science Masterclass: https://datasciencemasterclass.substack.com/

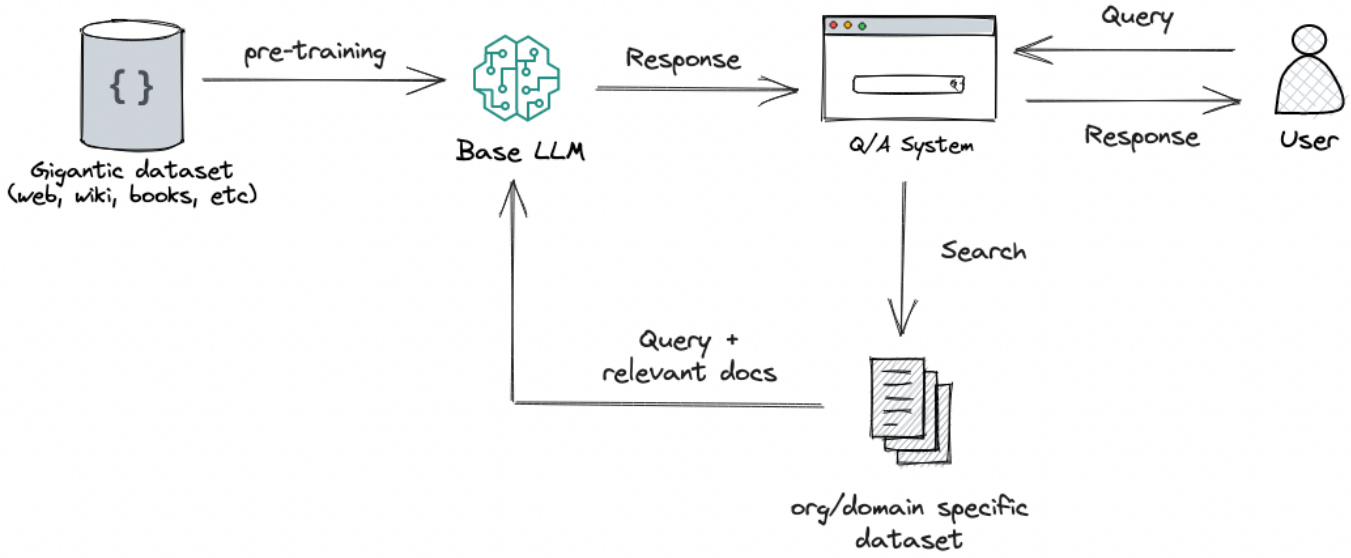
More from Rukshan Pramoditha and Towards Data Science

Rukshan Pramoditha in Towards Data Science

20 Necessary Requirements of a Perfect Laptop for Data Science and Machine Learning Tasks

· 7 min read · Jun 5, 2021

801 19



Heiko Hotz in Towards Data Science

RAG vs Finetuning—Which Is the Best Tool to Boost Your LLM Application?

The definitive guide for choosing the right method for your use case

· 19 min read · Aug 24

2.1K 16

Giuseppe Scalamogna in Towards Data Science

New ChatGPT Prompt Engineering Technique: Program Simulation

A potentially novel technique for turning a ChatGPT prompt into a mini-app.

9 min read · Sep 3

1.4K 14

Rukshan Pramoditha in Towards Data Science

11 Dimensionality reduction techniques you should know in 2021

Reduce the size of your dataset while keeping as much of the variation as possible

· 14 min read · Apr 13, 2021

1.1K 10

See all from Rukshan Pramoditha

See all from Towards Data Science

Recommended from Medium

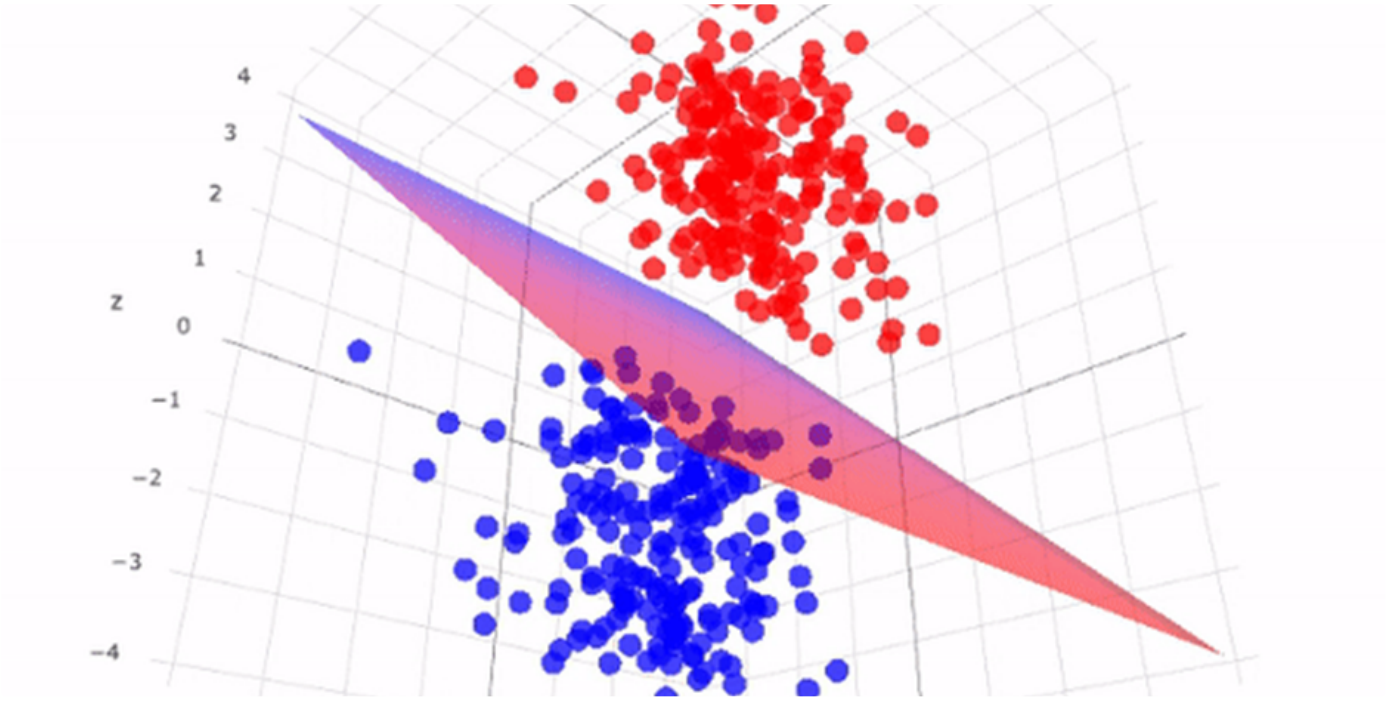
Rukshan Pramoditha in Data Science 365

Determining the Right Batch Size for a Neural Network to Get Better and Faster Results

Guidelines for choosing the right batch size to maintain optimal training speed and accuracy while saving computer resources

· 4 min read · Sep 26, 2022

52



Tasmay Pankaj Tibrewal in Low Code for Data Science

Support Vector Machines (SVM): An Intuitive Explanation

Everything you always wanted to know about this powerful supervised ML algorithm

17 min read · Jul 1

728 4

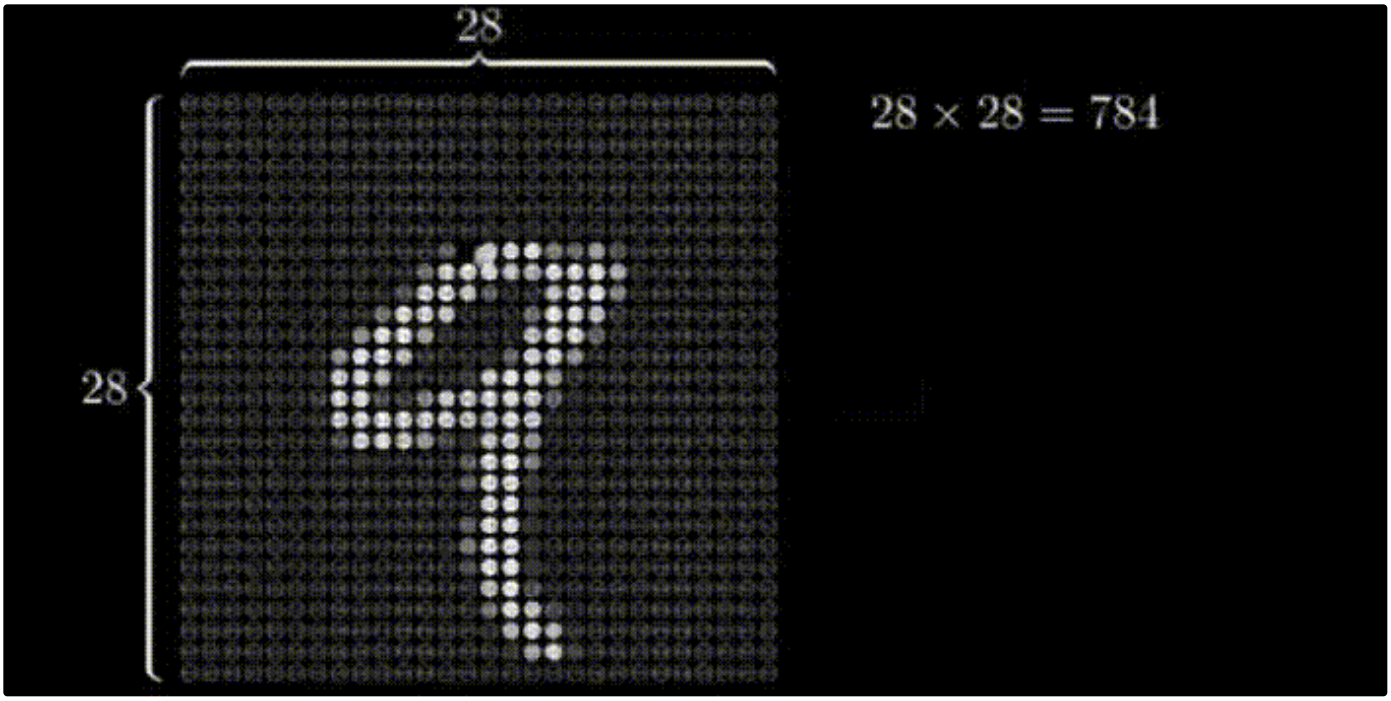
Lists

AI Regulation
6 stories · 136 saves

ChatGPT prompts
24 stories · 433 saves

ChatGPT
21 stories · 169 saves

Natural Language Processing
652 stories · 257 saves



Sadaf Saleem

Neural Networks in 10mins. Simply Explained!

What are Neural Networks?

9 min read · May 15

164 1

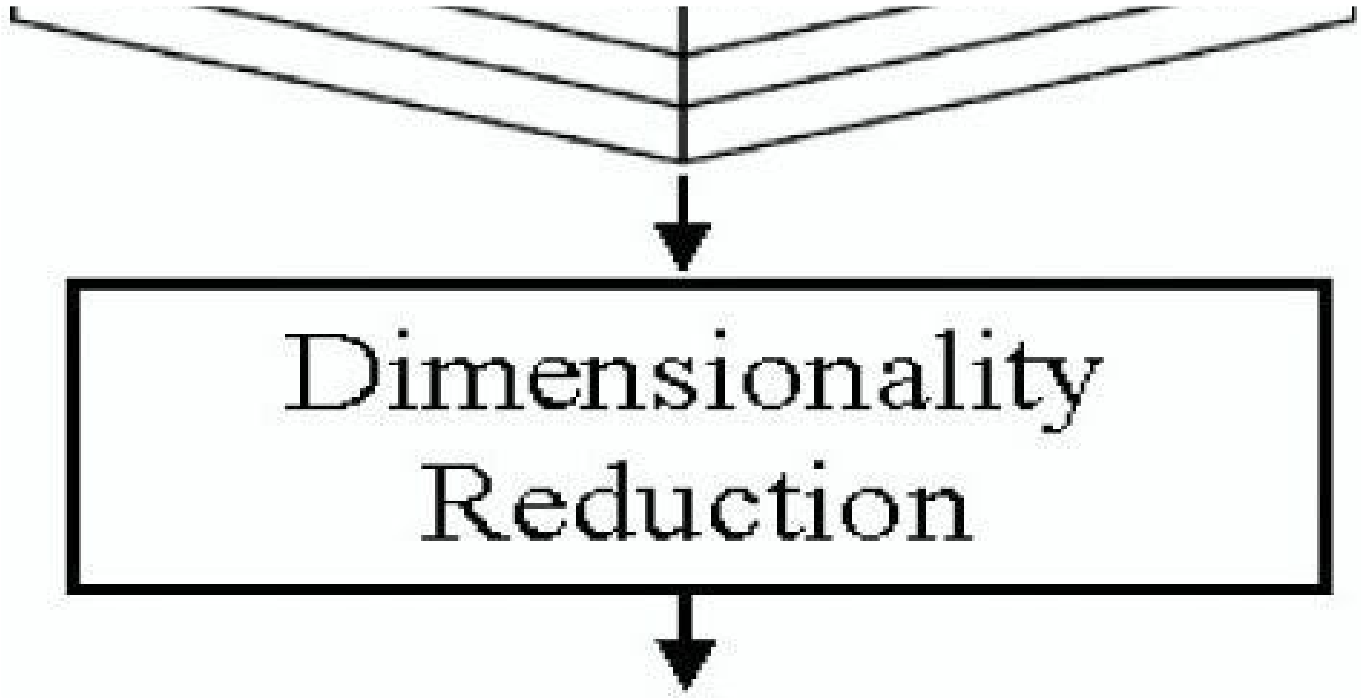
Huda Saleh

Categorical Variables Encoding

Categorical variables are a common type of data encountered in machine learning tasks. These variables represent categories or labels and...

6 min read · Sep 9

3



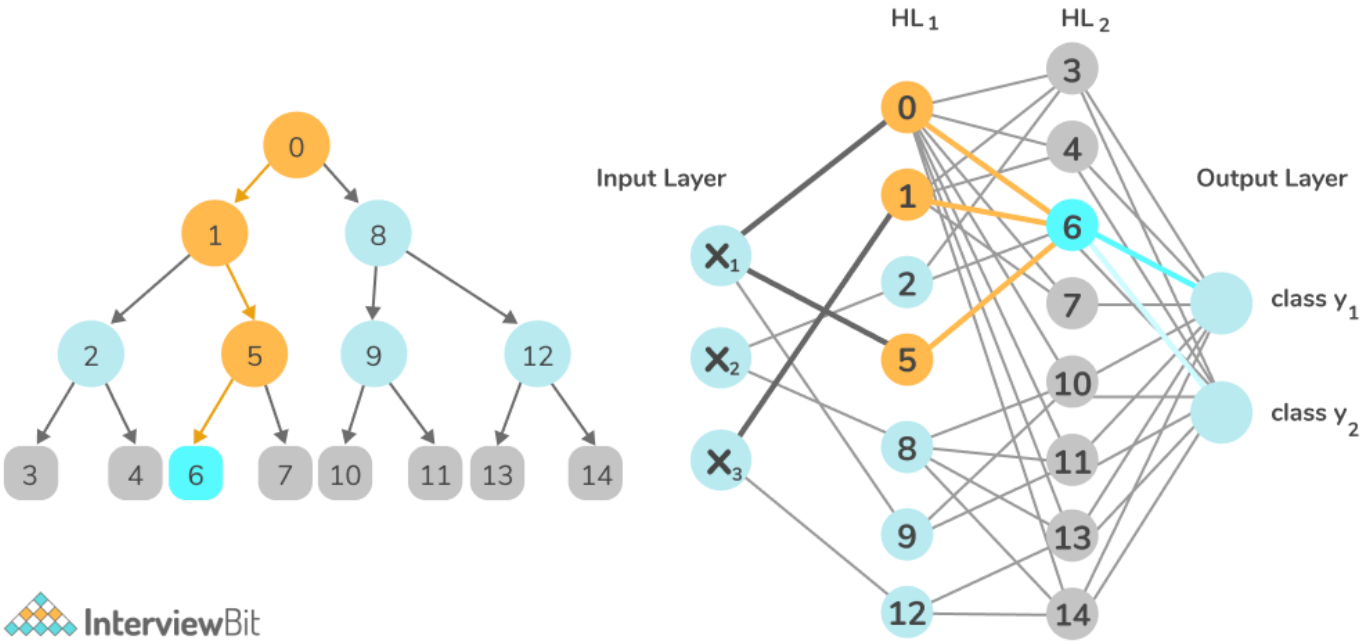
Manpreet Kaur

Dimensionality Reduction

Dimensionality reduction is a technique used to reduce the number of features in a dataset while retaining as much of the important...

3 min read · Sep 7

1



Mohsen Nabil

Decision Trees vs. Neural Networks

Both decision trees, including tree ensembles as well as neural networks are very powerful, very effective learning algorithms. When should...

3 min read · Jul 21

9

See more recommendations