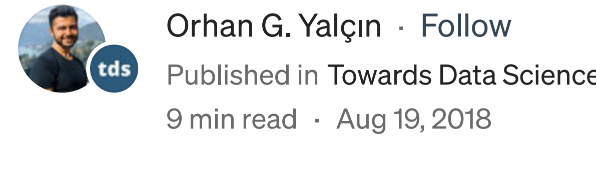
DEEP LEARNING CASE STUDIES

Image Classification in 10 Minutes with MNIST Dataset

Using Convolutional Neural Networks to Classify Handwritten Digits with TensorFlow and Keras | Supervised Deep Learning



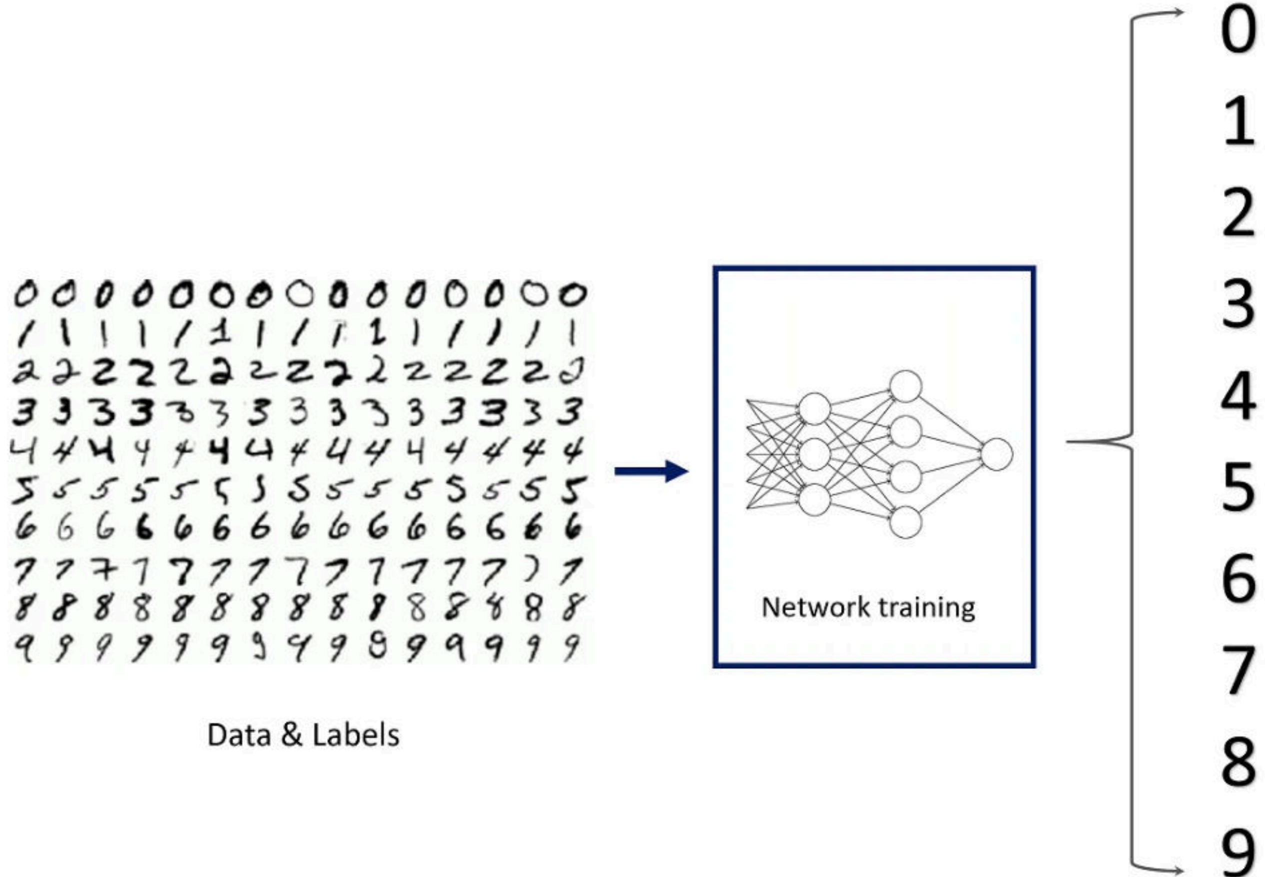
Orhan G. Yalçın · Follow

Published in Towards Data Science

9 min read · Aug 19, 2018

Listen Share More

> *If you are reading this article, I am sure that we share similar interests and are/will be in similar industries. So let's connect via Linkedin! Please do not hesitate to send a contact request! Orhan G. Yalçın - Linkedin*



MNIST Dataset and Number Classification by Katakoda

Before diving into this article, I just want to let you know that if you are into deep learning, I believe you should also check my other articles such as:

1 — Image Noise Reduction in 10 Minutes with Deep Convolutional Autoencoders where we learned to build autoencoders for image denoising;

2 — *Predict Tomorrow's Bitcoin (BTC) Price with Recurrent Neural Networks where we use an RNN to predict BTC prices and since it uses an API, the results always remain up-to-date.*

When you start learning deep learning with different neural network architectures, you realize that one of the most powerful supervised deep learning techniques is the Convolutional Neural Networks (abbreviated as "CNN"). The final structure of a CNN is actually very similar to Regular Neural Networks (RegularNets) where there are neurons with weights and biases. In addition, just like in RegularNets, we use a loss function (e.g. crossentropy or softmax) and an optimizer (e.g. adam optimizer) in CNNs [CS231]. Additionally though, in CNNs, there are also Convolutional Layers, Pooling Layers, and Flatten Layers. CNNs are mainly used for image classification although you may find other application areas such as natural language processing.

Why Convolutional Neural Networks

The main structural feature of RegularNets is that all the neurons are connected to each other. For example, when we have images with 28 by 28 pixels in greyscale, we will end up having 784 (28 x 28 x 1) neurons in a layer that seems manageable. However, most images have way more pixels and they are not grey-scaled. Therefore, assuming that we have a set of color images in 4K Ultra HD, we will have 26,542,080 (4096 x 2160 x 3) different neurons connected to each other in the first layer which is not really manageable. Therefore, we can say that RegularNets are not scalable for image classification. However, especially when it comes to images, there seems to be little correlation or relation between two individual pixels unless they are close to each other. This leads to the idea of Convolutional Layers and Pooling Layers.

Layers in a CNN

We are capable of using many different layers in a convolutional neural network. However, convolution, pooling, and fully connected layers are the most important ones. Therefore, I will quickly introduce these layers before implementing them.

Convolutional Layers

The convolutional layer is the very first layer where we extract features from the images in our datasets. Due to the fact that pixels are only related to the adjacent and close pixels, convolution allows us to preserve the relationship between different parts of an image. Convolution is basically filtering the image with a smaller pixel filter to decrease the size of the image without losing the relationship between pixels. When we apply convolution to 5x5 image by using a 3x3 filter with 1x1 stride (1-pixel shift at each step). We will end up having a 3x3 output (64% decrease in complexity).

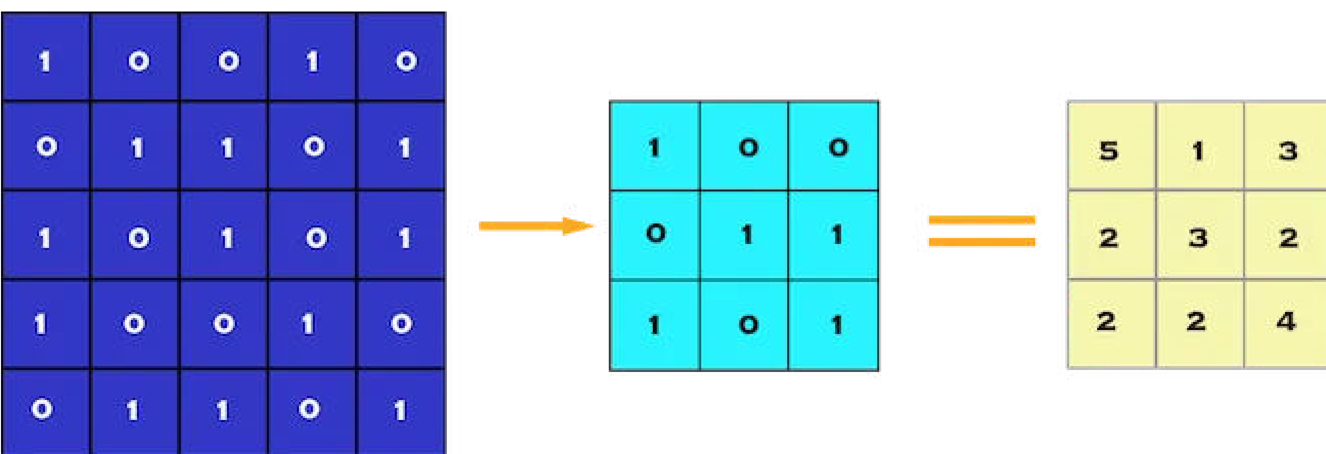
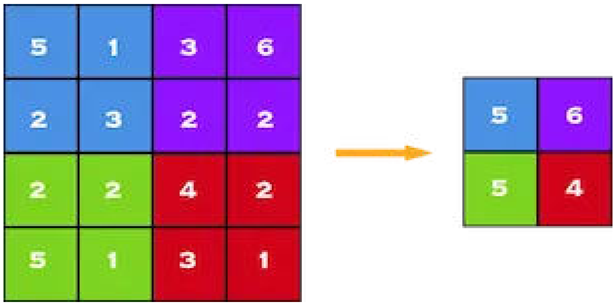


Figure 1: Convolution of 5 × 5 pixel image with 3 × 3 pixel filter (stride = 1 × 1 pixel)

Pooling Layer

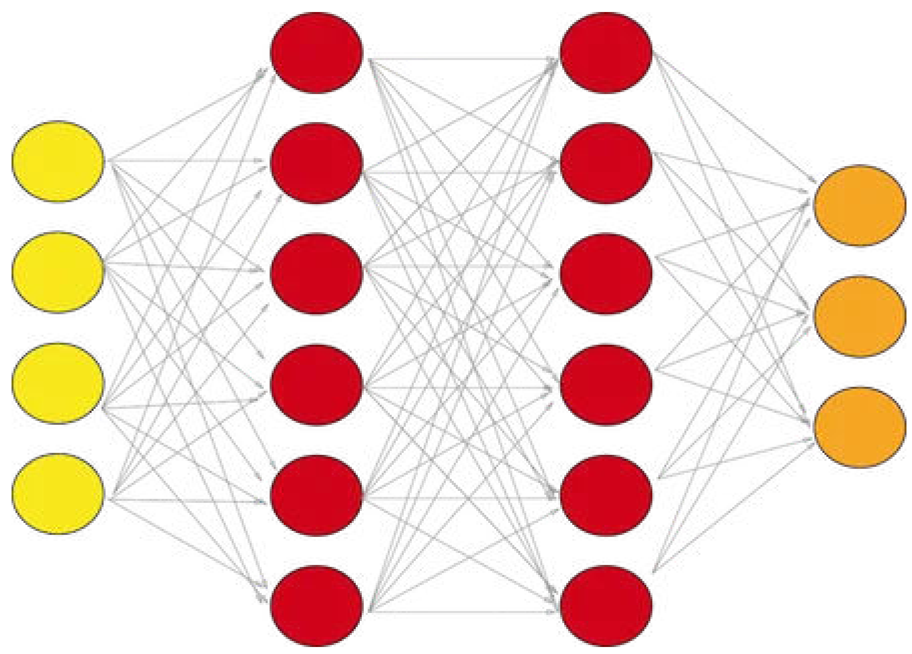
When constructing CNNs, it is common to insert pooling layers after each convolution layer to reduce the spatial size of the representation to reduce the parameter counts which reduces the computational complexity. In addition, pooling layers also helps with the overfitting problem. Basically we select a pooling size to reduce the amount of the parameters by selecting the maximum, average, or sum values inside these pixels. Max Pooling, one of the most common pooling techniques, may be demonstrated as follows:



Max Pooling by 2 × 2

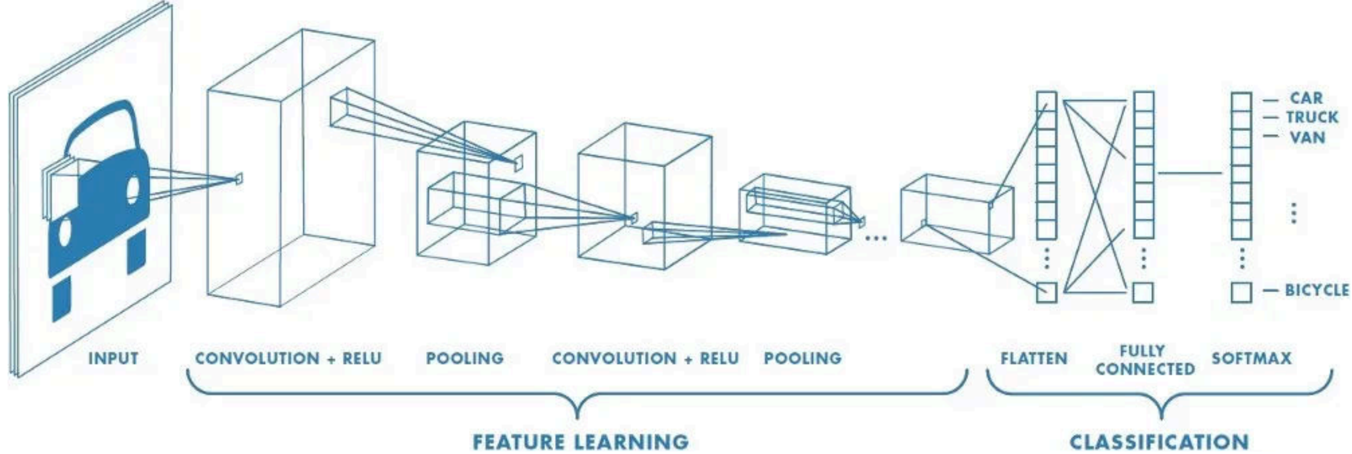
A Set of Fully Connected Layers

A fully connected network is our RegularNet where each parameter is linked to one another to determine the true relation and effect of each parameter on the labels. Since our time-space complexity is vastly reduced thanks to convolution and pooling layers, we can construct a fully connected network in the end to classify our images. A set of fully-connected layers looks like this:



A fully connected layer with two hidden layers

Now that you have some idea about the individual layers that we will use, I think it is time to share an overview look of a complete convolutional neural network.



A Convolutional Neural Network Example by Mathworks

And now that you have an idea about how to build a convolutional neural network that you can build for image classification, we can get the most cliche dataset for classification: the MNIST dataset, which stands for Modified National Institute of Standards and Technology database. It is a large database of handwritten digits that is commonly used for training various image processing systems.

Downloading the MNIST Dataset

The MNIST dataset is one of the most common datasets used for image classification and accessible from many different sources. In fact, even Tensorflow and Keras allow us to import and download the MNIST dataset directly from their API. Therefore, I will start with the following two lines to import TensorFlow and MNIST dataset under the Keras API.

```
1   import tensorflow as tf
2   (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

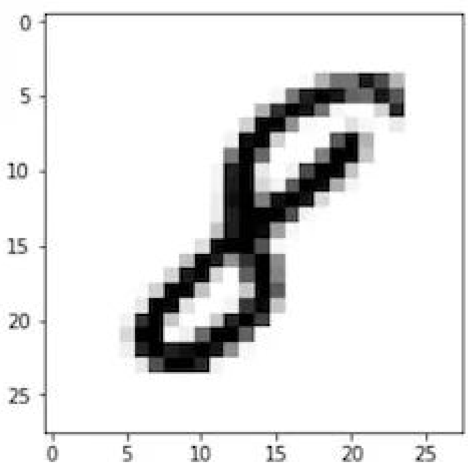
mnisttf.py hosted with ❤ by GitHub view raw

The MNIST database contains 60,000 training images and 10,000 testing images taken from American Census Bureau employees and American high school students [Wikipedia]. Therefore, in the second line, I have separated these two groups as train and test and also separated the labels and the images. x_train and x_test parts contain greyscale RGB codes (from 0 to 255) while y_train and y_test parts contain labels from 0 to 9 which represents which number they actually are. To visualize these numbers, we can get help from matplotlib.

```python
import matplotlib.pyplot as plt
%matplotlib inline # Only use this if using iPython
image_index = 7777 # You may select anything up to 60,000
print(y_train[image_index]) # The label is 8
plt.imshow(x_train[image_index], cmap='Greys')
```

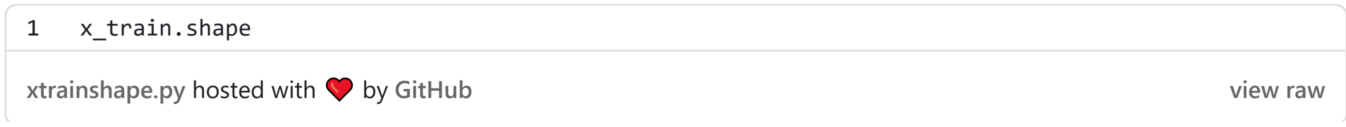
imshow.py hosted with ❤ by GitHub view raw

When we run the code above, we will get the greyscale visualization of the RGB codes as shown below.



A visualization of the sample image at index 7777

We also need to know the shape of the dataset to channel it to the convolutional neural network. Therefore, I will use the “shape” attribute of NumPy array with the following code:



```python
x_train.shape
```

xtrainshape.py hosted with ❤ by GitHub view raw

You will get (60000, 28, 28). As you might have guessed 60000 represents the number of images in the train dataset and (28, 28) represents the size of the image: 28 x 28 pixel.

Reshaping and Normalizing the Images

To be able to use the dataset in Keras API, we need 4-dims NumPy arrays. However, as we see above, our array is 3-dims. In addition, we must normalize our data as it is always required in neural network models. We can achieve this by dividing the RGB codes to 255 (which is the maximum RGB code minus the minimum RGB code). This can be done with the following code:

```python
# Reshaping the array to 4-dims so that it can work with the Keras API
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)
# Making sure that the values are float so that we can get decimal points after division
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
# Normalizing the RGB codes by dividing it to the max RGB value.
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print('Number of images in x_train', x_train.shape[0])
print('Number of images in x_test', x_test.shape[0])
```

preparing_mnist.py hosted with ❤ by GitHub view raw

Building the Convolutional Neural Network

We will build our model by using high-level Keras API which uses either TensorFlow or Theano on the backend. I would like to mention that there are several high-level TensorFlow APIs such as Layers, Keras, and Estimators which helps us create neural networks with high-level knowledge. However, this may lead to confusion since they all vary in their implementation structure. Therefore, if you see completely different codes for the same neural network although they all use TensorFlow, this is why. I will use the most straightforward API which is Keras. Therefore, I will import the Sequential Model from Keras and add Conv2D, MaxPooling, Flatten, Dropout, and Dense layers. I have already talked about Conv2D, Maxpooling, and Dense layers. In addition, Dropout layers fight with the overfitting by disregarding some of the neurons while training while Flatten layers flatten 2D arrays to 1D arrays before building the fully connected layers.

```python
# Importing the required Keras modules containing model and layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
# Creating a Sequential Model and adding the layers
model = Sequential()
model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten()) # Flattening the 2D arrays for fully connected layers
model.add(Dense(128, activation=tf.nn.relu))
model.add(Dropout(0.2))
model.add(Dense(10,activation=tf.nn.softmax))
```

cnn_model_with_keras.py hosted with ❤ by GitHub view raw

We may experiment with any number for the first Dense layer; however, the final Dense layer must have 10 neurons since we have 10 number classes (0, 1, 2, …, 9). You may always experiment with kernel size, pool size, activation functions, dropout rate, and a number of neurons in the first Dense layer to get a better result.

Compiling and Fitting the Model

With the above code, we created a non-optimized empty CNN. Now it is time to set an optimizer with a given loss function that uses a metric. Then, we can fit the model by using our train data. We will use the following code for these tasks:

```
1   model.compile(optimizer='adam',
2                 loss='sparse_categorical_crossentropy',
3                 metrics=['accuracy'])
4   model.fit(x=x_train,y=y_train, epochs=10)
```

model_compile.py hosted with ❤ by GitHub view raw

You can experiment with the optimizer, loss function, metrics, and epochs. However, I can say that adam optimizer is usually out-performs the other optimizers. I am not sure if you can actually change the loss function for multi-class classification. Feel free to experiment and comment below. The epoch number might seem a bit small. However, you will reach to 98–99% test accuracy. Since the MNIST dataset does not require heavy computing power, you may easily experiment with the epoch number as well.

Evaluating the Model

Finally, you may evaluate the trained model with x_test and y_test using one line of code:

```
1   model.evaluate(x_test, y_test)
```

model_evaluate.py hosted with ❤ by GitHub view raw

The results are pretty good for 10 epochs and for such a simple model.

```
10000/10000 [==============================] - 1s 134us/step
Out[188]: [0.06737536886025228, 0.985]
```

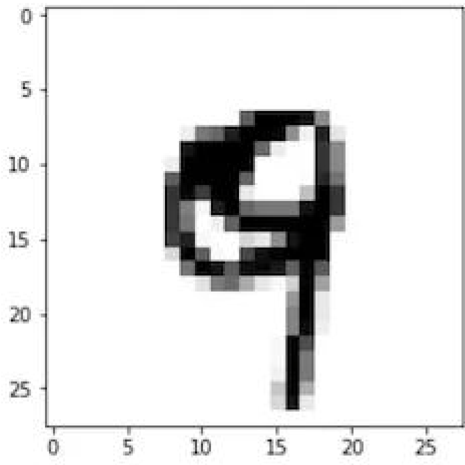
Evaluation shows 98.5% accuracy on test set!

We achieved 98.5% accuracy with such a basic model. To be frank, in many image classification cases (e.g. for autonomous cars), we cannot even tolerate 0.1% error since, as an analogy, it will cause 1 accident in 1000 cases. However, for our first model, I would say the result is still pretty good. We can also make individual predictions with the following code:

```
1    image_index = 4444
2    plt.imshow(x_test[image_index].reshape(28, 28),cmap='Greys')
3    pred = model.predict(x_test[image_index].reshape(1, 28, 28, 1))
4    print(pred.argmax())
```

single_prediction.py hosted with ❤ by GitHub   view raw

Our model will classify the image as a '9' and here is the visual of the image:



Our model correctly classifies this image as a 9 (Nine)

Although it is not really a good handwriting of the number 9, our model was able to classify it as 9.

Congratulations!

You have successfully built a convolutional neural network to classify handwritten digits with Tensorflow's Keras API. You have achieved accuracy of over 98% and now you can even save this model & create a digit-classifier app! If you are curious about saving your model, I would like to direct you to the Keras Documentation. After all, to be able to efficiently use an API, one must learn how to read and use the documentation.

Subscribe to the Mailing List for the Full Code

If you would like to have access to full code on Google Colab and have access to my latest content, subscribe to the mailing list: ✉️

Subscribe Now

Enjoyed the Article

If you like this article, consider checking out my other similar articles:

Image Noise Reduction in 10 Minutes with Convolutional Autoencoders

Using Deep Convolutional Autoencoders to Clean (or Denoise) Noisy Images with the help of Fashion MNIST | Unsupervised...

towardsdatascience.com

Image Generation in 10 Minutes with Generative Adversarial Networks

Using Unsupervised Deep Learning to Generate Handwritten Digits with Deep Convolutional GANs using TensorFlow and the...

towardsdatascience.com

Using Recurrent Neural Networks to Predict Bitcoin (BTC) Prices

Wouldn't it be awesome if you were, somehow, able to predict tomorrow's Bitcoin (BTC) price? Cryptocurrency market has...

towardsdatascience.com

Fast Neural Style Transfer in 5 Minutes with TensorFlow Hub & Magenta

Transferring the van Gogh's Unique Style to Photos with Magenta's Arbitrary Image Stylization Network and Deep Learning

towardsdatascience.com

Machine Learning · Convolution Neural Net · Computer Vision · Image Recognition · Deep Learning

Written by Orhan G. Yalçın

1.8K Followers · Writer for Towards Data Science

I write about AI and data apps here building them at Vizio.ai with my team. Feel free to get in touch!

More from Orhan G. Yalçın and Towards Data Science

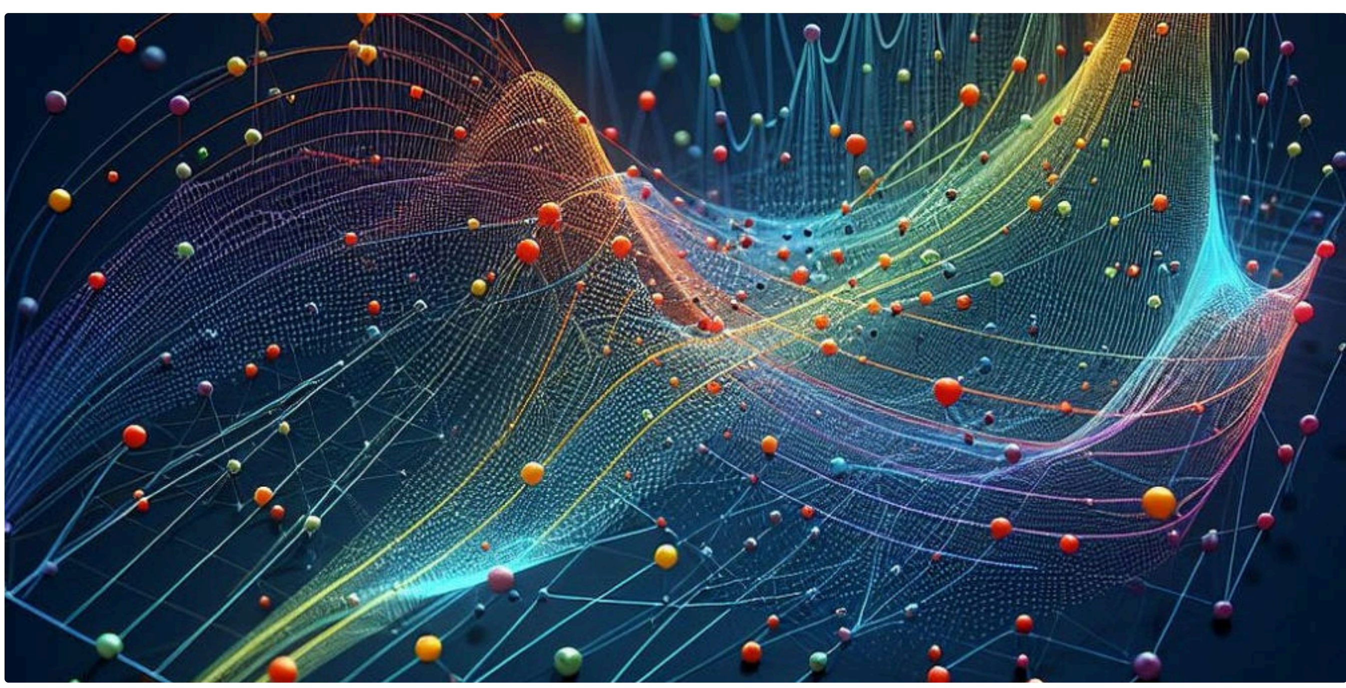
Orhan G. Yalçın in Towards Data Science

Eager Execution vs. Graph Execution: Which is Better?

Comparing Eager Execution and Graph Execution using Code Examples, Understanding When to Use Each and why TensorFlow switched to Eager...

· 9 min read · Oct 23, 2020

739 4



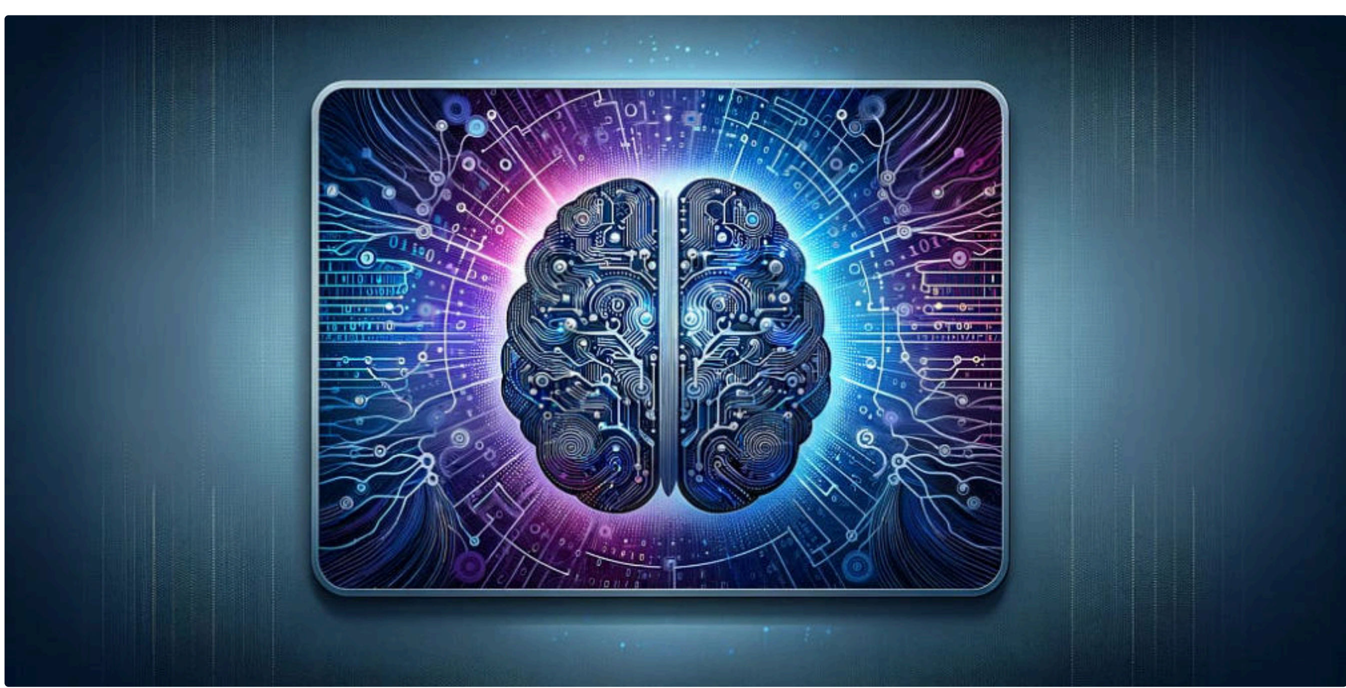
Tim Sumner in Towards Data Science

A New Coefficient of Correlation

What if you were told there exists a new way to measure the relationship between two variables just like correlation except possibly...

10 min read · Mar 31, 2024

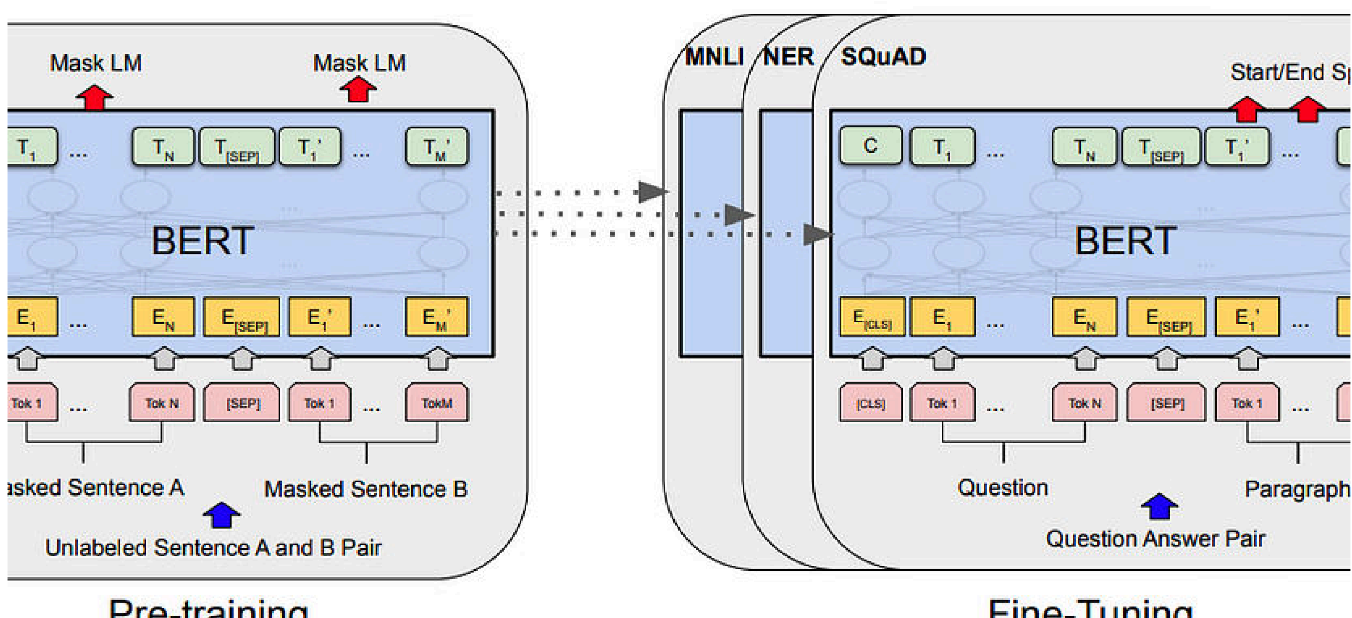
2.9K 34



Cristian Leo in Towards Data Science

The Math Behind Neural Networks

Dive into Neural Networks, the backbone of modern AI, understand its mathematics, implement it from scratch, and explore its applications



Orhan G. Yalçın in Towards Data Science

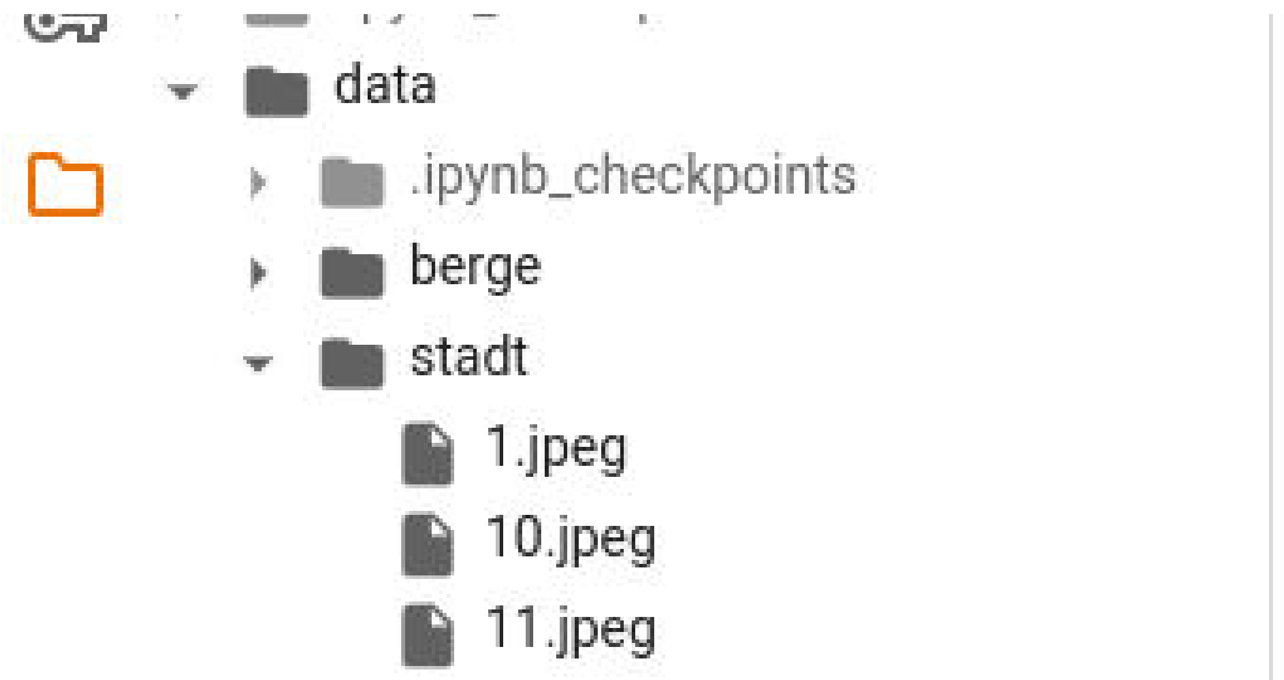
Sentiment Analysis in 10 Minutes with BERT and Hugging Face

Learn the basics of the pre-trained NLP model, BERT, and build a sentiment classifier using the IMDB movie reviews dataset, TensorFlow...

See all from Orhan G. Yalçın

See all from Towards Data Science

Recommended from Medium



Sarka Pribylova

Image classification using CNN

Convolutional Neural Network (CNN) is a well established data architecture. It is a supervised machine learning methodology used mainly in...

5 min read · Feb 24, 2024

3

Data
Train
Valid
Test
class_a
class_b
class_a
class_b
Test_folder
class_a_1.jpg
class_b_1.jpg
class_a_500.jpg
class_b_500.jpg
test_1.jpg

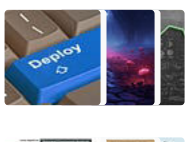
Open in app ↗

Search

4 min read · Nov 2, 2023

93

Lists



Predictive Modeling w/ Python

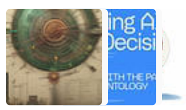
20 stories · 1117 saves

Practical Guides to Machine Learning

10 stories · 1339 saves

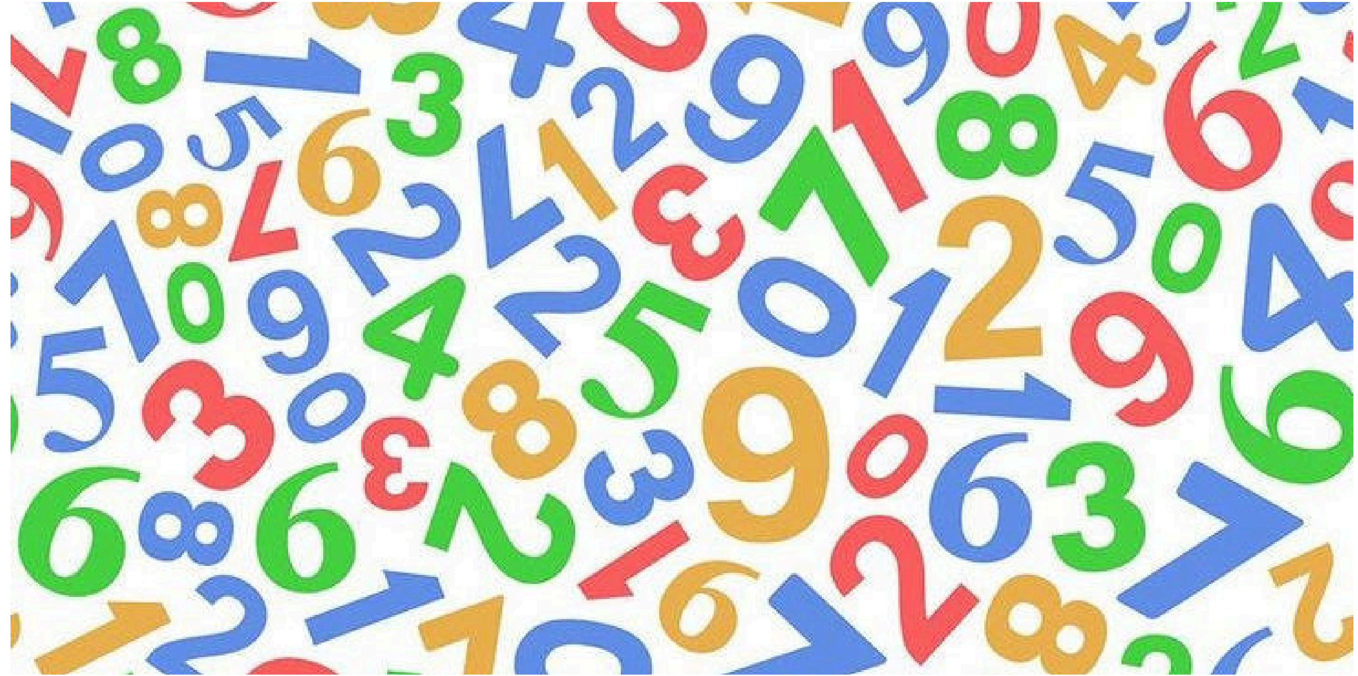
Natural Language Processing

1390 stories · 890 saves



data science and AI

40 stories · 135 saves



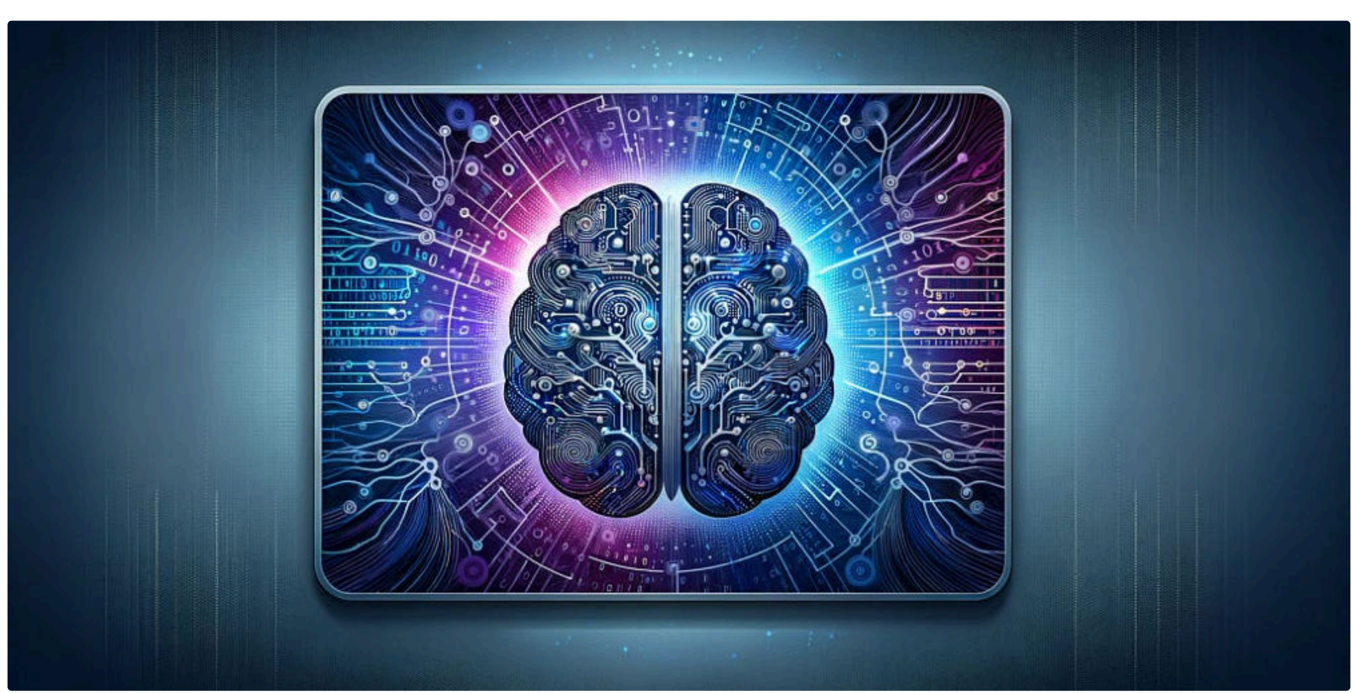
Azka Redhia

Handwritten Digit Recognition

Making A Simple Neural Network Model for Handwritten Digit Classification

7 min read · Nov 24, 2023

50 1



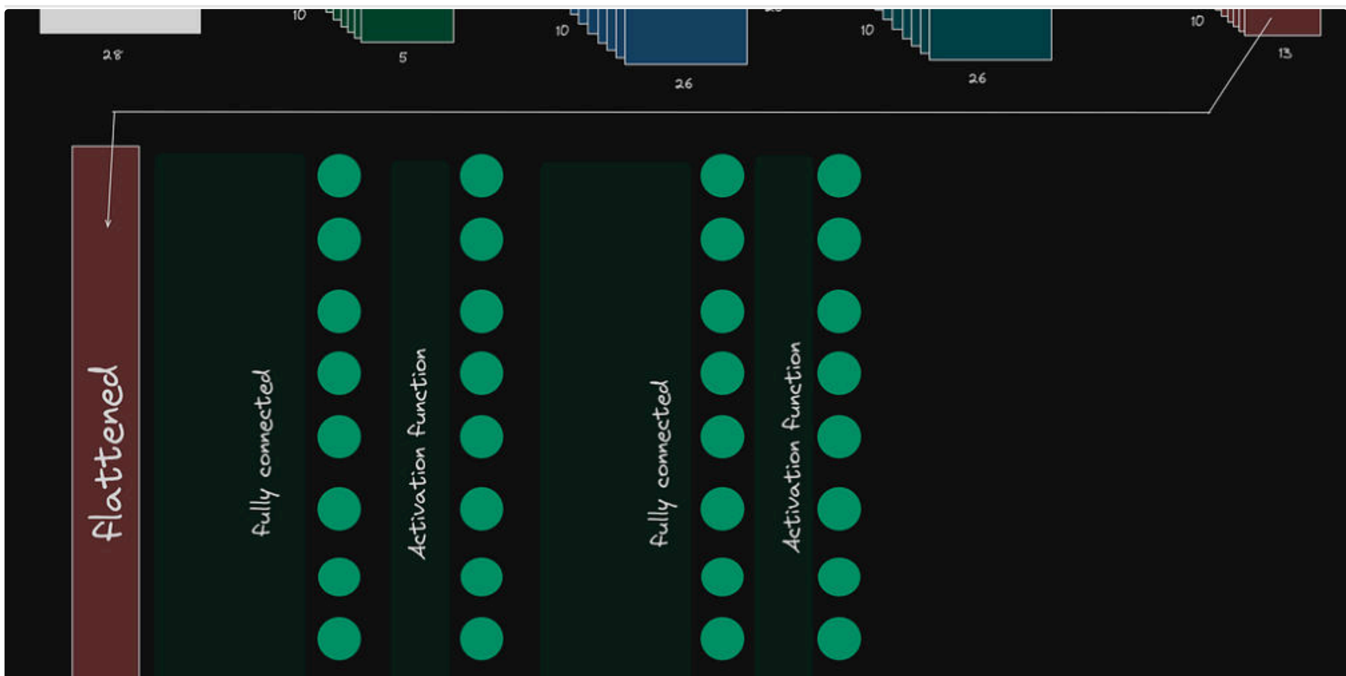
Cristian Leo in Towards Data Science

The Math Behind Neural Networks

Dive into Neural Networks, the backbone of modern AI, understand its mathematics, implement it from scratch, and explore its applications

· 28 min read · Mar 29, 2024

2.9K 20

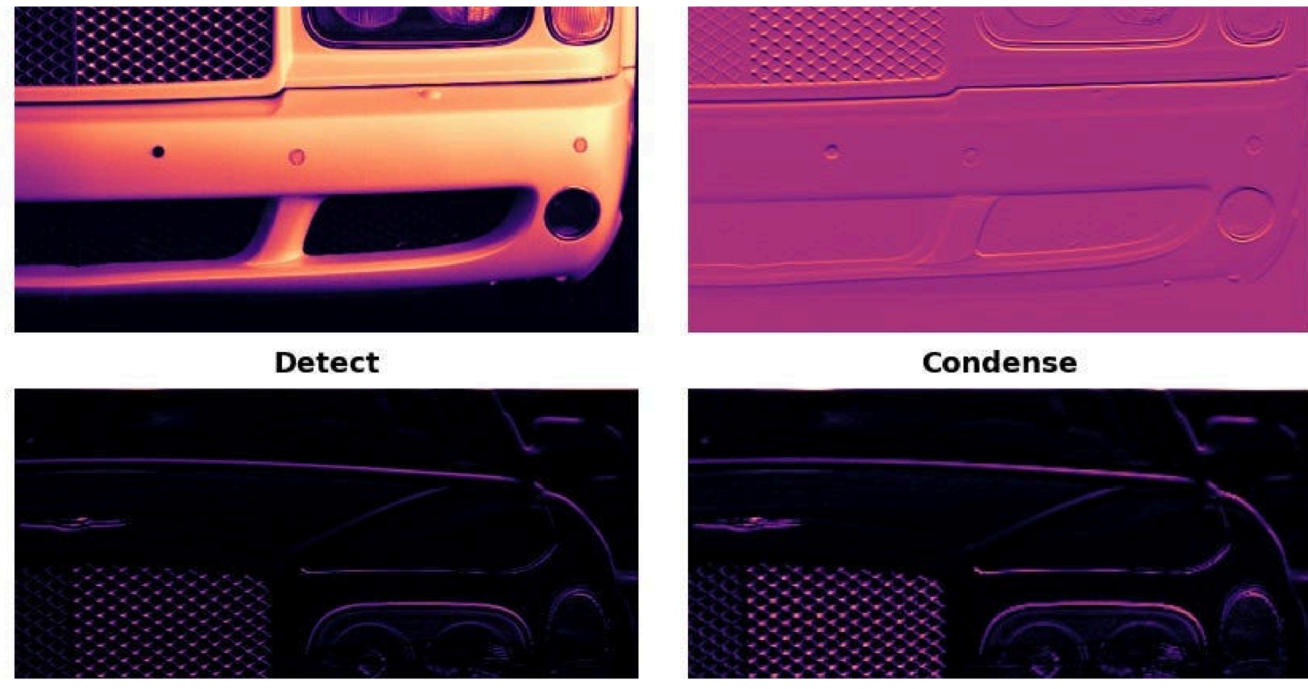


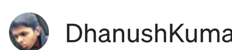
Nikhil Shivanath

CNNs from scratch (NumPy only)

This project is a personal undertaking and its sole purpose was to de-mystify the working of a convolutional neural network . I've loved...

9 min read · Apr 11, 2024





DhanushKumar

Convolution and ReLU

Convolution is a mathematical operation that combines two functions to produce a third. In the context of signal processing, it involves...

9 min read · Nov 23, 2023

54

See more recommendations