Project:
Concrete Surface Crack Detection using CNN

Hrishikesh Pawar • 2018meb1241

Dataset - **Courtesy : Mendley Data** (http://dx.doi.org/10.17632/5y9wdsg2zt.2)

Data

- The dataset contains concrete images having cracks. The data is collected from various METU Campus Buildings.
- The dataset is divided into two classes as negative and positive crack images for image classification.

Images

- These High-resolution images have variance in terms of surface finish and illumination conditions.
- Each class [Cracked and Not cracked] has 20000 images with a total of 40000 images with 227 x 227 pixels with RGB channel
- No data augmentation in terms of random rotation or flipping is applied.

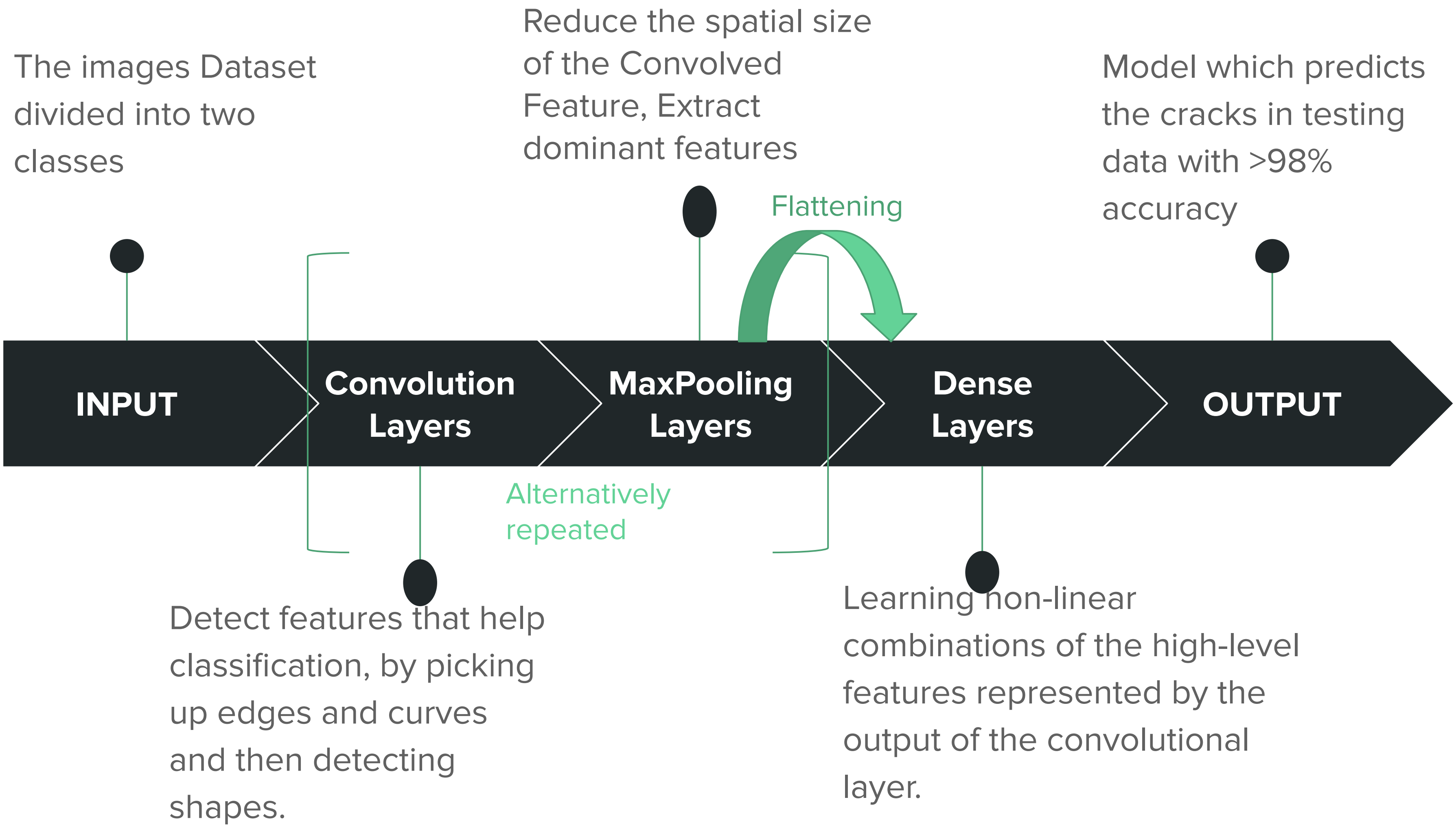
Model Overview

Data Preprocessing

Convolutional Layers with Max Pooling

Dense Fully Connected Layers

Loss Function and Optimizer


The images Dataset divided into two classes
Reduce the spatial size of the Convolved Feature, Extract dominant features
Model which predicts the cracks in testing data with >98% accuracy
Flattening
INPUT
Convolution Layers
MaxPooling Layers
Dense Layers
OUTPUT
Alternatively repeated
Detect features that help classification, by picking up edges and curves and then detecting shapes.
Learning non-linear combinations of the high-level features represented by the output of the convolutional layer.

Data Preprocessing

Rescaling images to 128 X 128 pixels

- Why 128 ?
- Power of 2 [We will perform Pooling with strides=2 thus reducing the output image size to half the input dimensions at each pooling layer]
- 80-20 distribution

Data Augmentation

- Horizontal Flipping
- Vertical Flipping
- Random Rotation - nearest fill mode
- Why? => Prediction accuracy of the Supervised Deep Learning models is largely reliant on the amount and the diversity of data available during training

Convolutional Layers

Inside the model

- We use the 3X3 and 1x1 convolutional filters.
- Regularization : L2 => “squared magnitude” of coefficient as penalty term to the loss function.

Purpose

- Does feature extraction, using convolutional filters, rectilinear activation, and further subsampling.

Pooling Layer

Why Pooling?

- Pooling layer is responsible for reducing the spatial size of the Convolved Feature
- Decrease the computational power required to process the data through dimensionality reduction

2X2 with strides=(2,2)

Why Max Pooling ?

- Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction
- This is the reason Max Pooling is considered better over other options

Dense Layers

Activation - Relu, Sigmoid
Regularization - L2

Why at the end?

- The dense layers at the end form a neural network that takes in the high-level features and classify the images on the basis of these features.
- These are the layers actually responsible for classification.

Deep is better than Wide

- Deeper networks capture the natural “hierarchy” that is present everywhere in nature. It captures low level features in first layer, a little better but still low level features in the next layer and at higher layers object parts and simple structures are captured.

Optimization

Loss Function

Binary Cross Entropy

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

Optimizer

Adam

The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

Model Summary - keras backend

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 128, 128, 256)	1024
max_pooling2d_3 (MaxPooling2	(None, 64, 64, 256)	0
conv2d_4 (Conv2D)	(None, 64, 64, 256)	65792
max_pooling2d_4 (MaxPooling2	(None, 32, 32, 256)	0
conv2d_5 (Conv2D)	(None, 32, 32, 256)	65792
max_pooling2d_5 (MaxPooling2	(None, 16, 16, 256)	0
flatten_1 (Flatten)	(None, 65536)	0
dense_4 (Dense)	(None, 128)	8388736
dense_5 (Dense)	(None, 64)	8256
dense_6 (Dense)	(None, 32)	2080
dense_7 (Dense)	(None, 1)	33

Total params: 8,531,713
Trainable params: 8,531,713
Non-trainable params: 0

Results

We were able to train the model

97% accuracy on training data

> 98% accuracy on the validation dataset

Results

```
Epoch 1/10
1000/1000 [==============================] - 3070s 3s/step - loss: 0.5396 - accuracy: 0.9166 - val_loss: 0.2541 - val_accuracy:
0.9801
Epoch 2/10
1000/1000 [==============================] - 3050s 3s/step - loss: 0.2536 - accuracy: 0.9737 - val_loss: 0.2043 - val_accuracy:
0.9795
Epoch 3/10
1000/1000 [==============================] - 3056s 3s/step - loss: 0.2110 - accuracy: 0.9748 - val_loss: 0.1808 - val_accuracy:
0.9691
Epoch 4/10
1000/1000 [==============================] - 3035s 3s/step - loss: 0.1784 - accuracy: 0.9768 - val_loss: 0.2002 - val_accuracy:
0.9803
Epoch 5/10
1000/1000 [==============================] - 3040s 3s/step - loss: 0.1642 - accuracy: 0.9773 - val_loss: 0.1448 - val_accuracy:
0.9781
Epoch 6/10
1000/1000 [==============================] - 3034s 3s/step - loss: 0.1417 - accuracy: 0.9778 - val_loss: 0.1138 - val_accuracy:
0.9827
Epoch 7/10
1000/1000 [==============================] - 3028s 3s/step - loss: 0.1407 - accuracy: 0.9781 - val_loss: 0.1130 - val_accuracy:
0.9831
Epoch 8/10
1000/1000 [==============================] - 3028s 3s/step - loss: 0.1342 - accuracy: 0.9779 - val_loss: 0.1432 - val_accuracy:
0.9824
Epoch 9/10
1000/1000 [==============================] - 3030s 3s/step - loss: 0.1277 - accuracy: 0.9798 - val_loss: 0.1390 - val_accuracy:
0.9793
Epoch 10/10
1000/1000 [==============================] - 3131s 3s/step - loss: 0.1333 - accuracy: 0.9799 - val_loss: 0.1162 - val_accuracy:
0.9851
```

Extra : Insight into dense neural networks

NN Architecture	*Loss (Train)*	*Accuracy(Train)*	*Loss(Val)*	*Accuracy(Val)*
3 CNN – 1 NN	0.3942	0.9042	0.2530	0.9306
3 CNN – 2 NN	0.3536	0.9260	0.3023	0.9815
3 CNN – 3 NN	0.4419	0.9386	0.3099	0.9783
3 CNN – 4 NN	0.3810	0.9293	0.1681	0.9799
3 CNN – 5 NN	0.5396	0.9166	0.2541	0.9801
3 CNN – 5 NN (10 epochs)	0.1333	0.9799	0.1162	0.9851

Credits

Dataset

Özgenel, Çağlar Fırat (2019), "Concrete Crack Images for Classification", Mendeley Data, V2, doi: 10.17632/5y9wdsg2zt.2

Tensorflow

Keras, ImagePreprocessing

OS Module

Path - Directories

Jupyter Python Notebook

IDE

Python

THANK

YOU