ITC INFOTECH

Business-friendly Solutions

Deep Learning based Computer Vision Techniques for Defect Detection & Monitoring

Ananda Prakash Jena
Associate Data Scientist
ITC Infotech, Bengaluru, India

Anindya Neogi
Chief Data Scientist
ITC Infotech, Bengaluru, India

Keywords

Deep Learning, Convolution Neural Network, Defect Identification, Defect Localization, Tensorflow, Keras, Theano, Open CV

Abstract

Everything from water to crude oil is being transported through millions of miles of pipelines all over the world through transport and distribution networks. This network is prone to many risks. The major threat that occurs in pipelines is **leakage.** The effects of leakage go beyond repair expense and cost of lost oil or gas. It also significantly affects the human lives and environment. To impede these huge costs, a reliable leak detection technique is crucial. Many researches have been done during the last decades to find the location and size of the leakage with high accuracy. In this work we devised a Deep Learning-based solution using ***Convolution Neural Networks(CNN)*** and ***image processing*** techniques to detect and localize the leakages in oil & water pipelines.

1. Introduction:

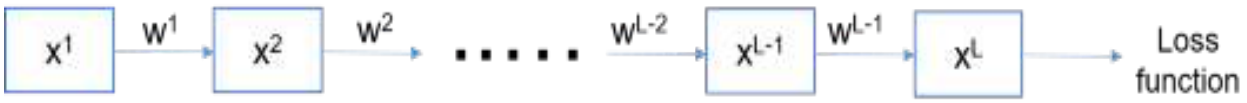
The problem of detecting leakage is tackled by using *Convolution Neural Networks(CNN),* a Deep Learning-based technique. Convolutional neural network (ConvNets or CNNs) is one of the primary techniques to perform *Image recognition, Images classification, Objects detection, and Face Recognition.*

To understand all the nitty-gritties of a CNN from a mathematical viewpoint we start with an explanation of tensors, vectorization, chain rule and then talk about the architecture of a CNN- all its layers *(convolution, activation, pooling, loss)* along with the training methodology of *Stochastic Gradient Descent (SGD)* by applying backpropogation.

Tensors can be thought of as, nothing but higher-order matrices. A tensor is to a matrix what a cube is to a square. A color image is represented as an order 3 tensor (H X W X 3) in height-width-channels format. Tensors can be easily vectorized and are highly used in different layers of a CNN.Chain rule is utilized in the learning process of a CNN. The general notation is:

$$dz/dx = (dz/dy).(dy/dx)$$

Here, z is a function of y, which in turn is a function of x. We apply the chain rule to get the derivative of z with respect to x using y in between. We'll see how this rule is applied in back-propagation. A CNN takes a tensor as an input, which is then processed sequentially, through a number of layers using the weight/parameter tensors.



Basic structure of a CNN—x^1, x^2 etc. are the tensor inputs and different layer responses; w^1, w^2 etc. are the learnable parameters/weights

Consider an object recognition problem with binary target classes. The target output, y, will be a 1-D array of size two, having the element as zero, except the correct class, having a value of one. The loss layer is used to measure the error between the predicted class probabilities and y. The layer just before the loss layer is a binary *cross-entropy function,* which converts the network predictions to a probability mass function of the binary target classes. Prediction using a CNN only requires a forward pass, where the input, x^1 is processed through the various layers *(using the learned weights w^1, w^2...)* to arrive at an estimated posterior probability distribution of x^1 for the categories. The class/category with the maximum probability is predicted.

$$Z = (½) \, || \, y - x^L \, ||^2$$

z represents the error term/squared loss function

Error back-propagation: The backpropagation algorithm is an approach to find out how much a particular weight at any layer is responsible for the total loss and hence, the amount by which it should be modified. The error is propagated from the loss layer to the previous layers through two sets of partial derivatives/gradients — *(i) gradient of the loss function with respect to weights at each layer, (ii) gradient of the loss function with respect to layer outputs at each layer. We use the chain rule to compute these gradients.*

$$\frac{\partial z}{\partial(\mathrm{vec}(\boldsymbol{w}^i)^T)} = \frac{\partial z}{\partial(\mathrm{vec}(\boldsymbol{x}^{i+1})^T)} \frac{\partial\,\mathrm{vec}(\boldsymbol{x}^{i+1})}{\partial(\mathrm{vec}(\boldsymbol{w}^i)^T)},$$
$$\frac{\partial z}{\partial(\mathrm{vec}(\boldsymbol{x}^i)^T)} = \frac{\partial z}{\partial(\mathrm{vec}(\boldsymbol{x}^{i+1})^T)} \frac{\partial\,\mathrm{vec}(\boldsymbol{x}^{i+1})}{\partial(\mathrm{vec}(\boldsymbol{x}^i)^T)}.$$

The error is propagated layer by layer using the second gradient equation. The first is used to update the weights in layer I.

Convolution layer: Convolution operation involves overlapping of a kernel of fixed size over the input tensor and then sliding across pixel-by-pixel to cover the entire image/tensor. For the overlapped area, we compute the product between the elements of the kernel and the image at the same location and then sum it up.

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

Input x Filter

4		

Feature Map

The spatial extent of the output is smaller than that of the input if the convolution kernel is larger than 1X1. To ensure that the input and output tensors have the same size, we can apply *zero padding* (padding the input image all around by zeros). For example, in the above fig, a 5X5 tensor is reduced to a 3X3 tensor by a 3X3 kernel/filter. Instead, if we add one row of zeros to the top and bottom, along with one column of zeros to the sides of the image, the output will be a 5X5 feature map, thus maintaining the input image size. Another important concept in convolution is *stride.* Stride is basically the number of steps by which the convolution kernel slides each time. Generally, a stride of 1 is used as in the above figure. However, if stride, s > 1, convolution is performed once every s pixels, both horizontally and vertically. In general, output size for a convolution layer is given by:

$$O = (I - (F - S) + 2P) / S$$

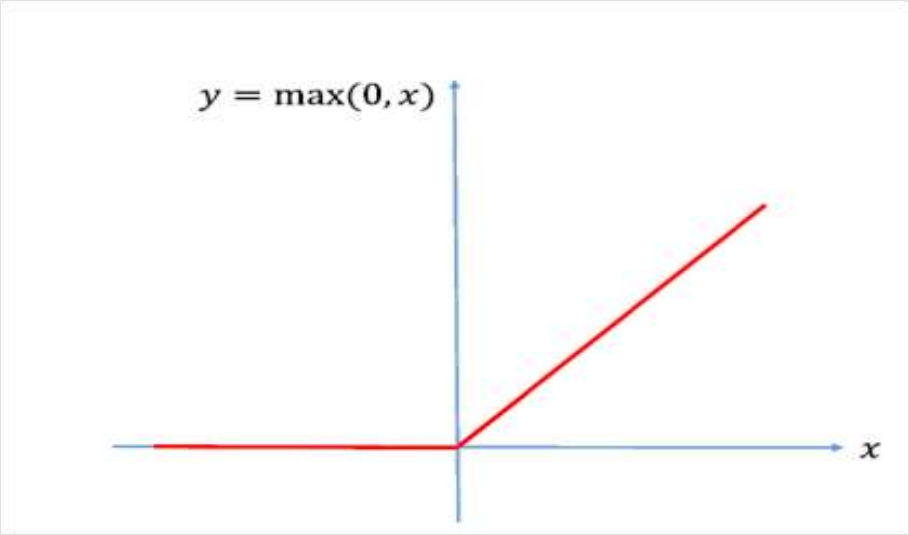
where, O is the output size, I is the input size, F is the filter/kernel size, P is the number of rows/columns padded and S is the stride. Convolution uses various filters, trained using backpropagation, to recognize simple patterns (edges, corners etc.) in images. In deeper layers, multiple feature detectors combine to detect complex patterns or objects. It helps the CNN in extracting features with local information.

Back-propagating errors in convolution layer: Backpropagation in convolution layers follows a similar approach as in fully connected/dense layers. They use the chain rule to calculate the partial derivative of the loss with respect to the output layer multiplied by the partial derivative of output layer with respect to the convolution filter/kernel. Gradient of loss with respect to the convolution filter is also calculated, to be used in updating the weight.

Consider x to be the input layer, y the output layer, F the convolution filter and z to be the loss function. Then with * as the convolution operator and *x tilde* as the row/column flipped version of the input

$$\partial z/\partial F = \partial z/\partial y * \tilde{x}$$

ReLU layer : ReLU activation function is denoted as : *relu(x) = max(0,x)*.

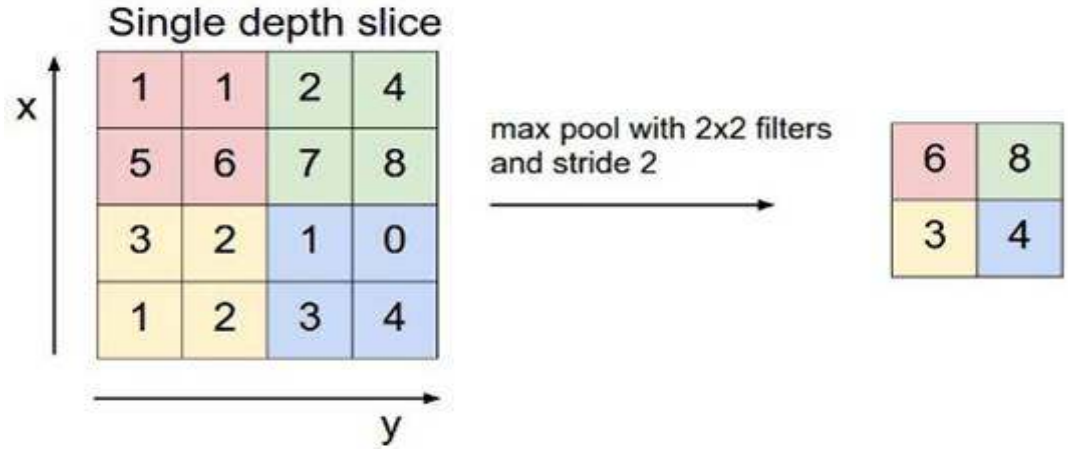


The ReLU function

Importance of ReLU activation: It adds non-linearity to the CNN model. The relationship between semantic features of an image and its pixel values are obviously highly non-linear. ReLU helps in modelling that non-linearity to some extent, by truncating the negative feature map values patterns at some particular regions. Combining many such object parts' detection helps in classifying the correct target class ReLU's gradient being 1 for the activated features helps it in learning. Compare that with a *sigmoid* or *tanh* activation functions, where the problem of *vanishing gradients* makes learning difficult. Moreover, gradient calculation in ReLU is faster than sigmoid or tanh, speeding up the entire learning process considerably.

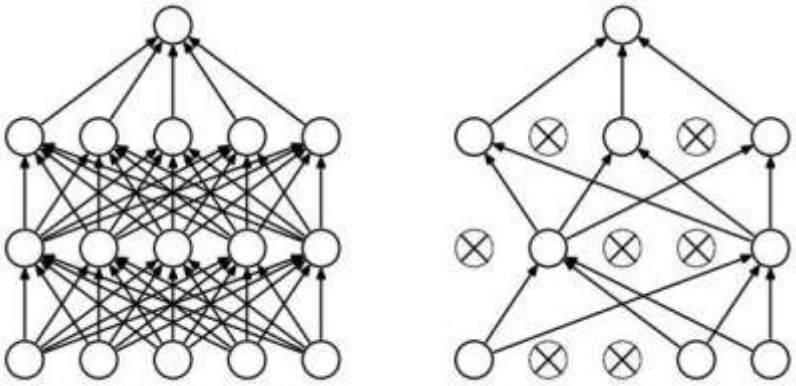
Pooling layer: Pooling is a local operator which is applied individually on each image channel. The spatial extent (H X W) and the stride of the pool are a part of the design of CNN structure. The most commonly used pooling setup is a 2X2 (HXW) region with a stride of 2. The pooling layer encodes the neighborhood information of a region into a single pixel. This helps in achieving a bit of invariance (positional + rotational). Pooling reduces the size of the *receptive field* significantly, thus reducing the training time, avoiding overfitting etc.

There are two kinds of pooling—*max pooling and average pooling.* Max pooling captures the highest activation value in a sub-region. Average pooling takes the mean of all activation values in the concerned sub-region. This results in feature detection with a rough idea of its location. Though we lose some information about the feature's exact position, we gain a lot through the highly reduced size of the feature maps.



Max-pooling divides the image into small sub-regions and then extracts only the useful information from them to transmit further down the network

Classification: After the convolution and pooling layers, the classification part consists of a few fully connected layers. However, these fully connected layers can only accept 1 - dimensional data. To convert our 3D data to 1D, we used the flatten function, which essentially arranges our 3D volume into a 1D vector. The last layers of a Convolutional NN are fully connected layers. Neurons in a fully connected layer have full connections to all the activations in the previous layer. This part is in principle the same as a regular Neural Network. Dropout is a technique used to improve over-fit on neural networks. Like other regularization techniques the use of dropout also make the training loss error a little worse. But that's the idea, our intention is to trade training performance for more generalization.



(a) Standard Neural Net (b) After applying dropout.

Modelling Technique: The modelling was done in a two stage process, where the first stage was identification of leakages using CNN, and second stage was leakage localization using the sliding window technique in OpenCV. For CNN we used *Keras, a Python library for deep learning* that wraps the efficient numerical libraries *TensorFlow and Theano.*

Image Classification: Convolutional Neural Network model which extracts the feature map outof an image by understanding the spatial relationship between pixel values in an RGB Image. The CNN's feature maps are passed into a fully connected Neural Network to classify if there is a leak or not in a given image.

Above are the sample raw data for our analysis. We applied some image pre-processing steps to improve the quality of data as given below:

Preprocessing steps: Cropping out the effective leakage region from the image, so as to reduce the size and area of interest. *Image Data Augmentation* where images would be horizontally and vertically flipped, rotated *(zoomed in and zoomed out)* to detect leaks at various directions, using Keras. After the preprocessing step, the data was passed into a sequential model. Below is the code block snippet describing the specifications of the model.

Architecture of CNN

```
model = Sequential()
model.add(Conv2D(128, kernel_size=(3, 3),activation='relu', input_shape=(32, 32, 3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25)) model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',  metrics=['accuracy'])
history = model.fit(x_train, y_train, batch_size=128,epochs=60,
verbose=1,validation_data=(x_train, y_train))
```

Object Localization

To localize the leak in an image with a probabilistic score, we used *Sliding Window* approach with stride (1,1) over the image. Each of the image subset is passed into the model for prediction. Based on the output from the model, which is a probability score based on whether a leak is detected, and creating a bounding box signifying leakage.

Model Accuracy and Validation:

Loss function: Binary Cross-entropy/Log-loss

To compute loss, we need to penalize wrong predictions. If the probability associated with the true class is 1.0, we need its loss to be zero. Conversely, if that probability is low, say, 0.01, we need its loss to be large. It turns out, taking the negative log of the probability suits us well enough for this purpose since the log of values between 0.0 and 1.0 is negative, we take the negative log to obtain a positive value for the loss.

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

Binary Cross-Entropy/Log Loss

where y is the label, 1 for **leak** class and 0 for **non-leak** class and p(y) is the predicted probability of the point belonging to l**eak** class for all **N** samples. Reading this formula, it tells you that, for each **class** (y=1), it adds ***log(p(y))*** to the loss, that is, the log probability of it being a **leak.** Conversely, it adds ***log(1-p(y)),*** that is, the log probability of it being non-leak, for each (y=0).

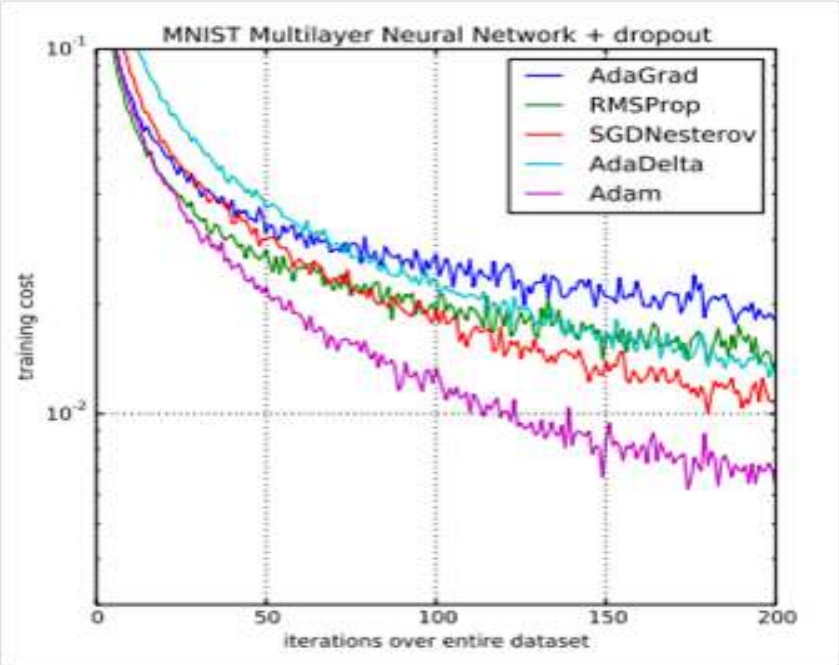
Model Accuracy Metric: A Model Accuracy metric is a function that is used to judge the performance of a model. In our CNN model we defined ***Accuracy*** *as:*

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

If ŷi is the predicted value of the i sample and y_i is the corresponding true value, then the fraction of correct predictions over $n_{samples}$ 1(x) is the indicator function.

2. Optimization:

We have used **Adam** as the optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based on training data. Adam is different from the classical stochastic gradient descent. Stochastic gradient descent maintains a single learning rate for all weight updates and the learning rate does not change during training. **Adam** maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems). It also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). Adam also makes use of the average of the second moments of the gradients. This means the algorithm does well on online and non-stationary problems (e.g. noise). Adam is a popular algorithm in the field of deep learning because it achieves good results fast.



Comparison of Adam to Other Optimization Algorithms Training

Validation: We had few images to begin with, hence we used all the training data in the validation step by introducing some pixel noise corresponding to images by applying suitable pre-processing techniques.To test the model, we kept a few random images that were not used in developing the model and used data-augmentation to create multiple images and finally used them for testing our model. We achieved a predictive probability of about 81%.

A snapshot of the output of the model localizing the region of leakage.

3. Parameter Selection:

The numeric parameters for our model was selected after monitoring the model performance using different values. We used GridSearchCVfrom sklearn,a machine learning library for the Python programming language to scan possible parameter space to detect significant parameters for the model.

4. Summary:

The model successfully detects leakages in pipelines even in the presence of limited data. Such a model is highly sensitive to hyper parameters.Hence their selection is crucial for better model performance. A Deep Learning model in general performs better when the amount of data available is large. Increasing the number of images would result in selecting the optimal hyper parameters for better model stability and reliability.

References:

[1] Francois Chollet, Deep Learning with Python

[2] Michael Nielsen, Neural Networks and Deep Learning

[3] Srivastava, Nitish; C. Geoffrey Hinton; Alex Krizhevsky; Ilya Sutskever; RuslanSalakhutdinov (2014), Journal of Machine Learning Research.

[4] Zhang, Wei (1991). "Error Back Propagation with Minimum-Entropy Weights: A Technique for Better Generalization of 2-D Shift-Invariant NNs". Proceedings of the International Joint Conference on Neural Networks.

[5] Habibi, Aghdam, Hamed (2017). Guide to convolutional neural networks : a practical application to traffic-sign detection and classification. Heravi, ElnazJahani. Cham, Switzerland. ISBN 9783319575490.

[6] He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian (2016). "Deep Residual Learning for Image Recognition". 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

[7] Romanuke, Vadim (2017). "Appropriate number and allocation of ReLUs in convolutional neural networks" . Research Bulletin of NTUU "Kyiv Polytechnic Institute".1: 69–78. doi:10.20535/1810-0546.2017.1.88156. Retrieved 17 February 2019

[8] Krizhevsky, A.; Sutskever, I.; Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks" (PDF). Advances in Neural Information Processing Systems. 1: 1097–1105.