TRAFFIC SIGN CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORK

Nemanja Veličković[1], Zeljko Stojković[2] , Goran Dimić[2],
Jelena Vasiljević[2] and Dhinaharan Nagamalai[3]

[1]University Union, School of Computing,
Knez Mihailova 6/VI, 11000 Belgrade, Serbia
[2]Institute Mihajlo Pupin, University of Belgrade,
Volgina 15, 11 000 Belgrade, Serbia
[3]Wireill, Australia

ABSTRACT

Artificial Neural Networks enables solving many problems in which classical computing is not up to task. Neural Networks and Deep Learning currently provide the best solutions to problems in image recognition, speech recognition and natural language processing. In this paper a Neural Network, more specific - Convolutional Neural Network solution for the purpose of recognizing and classifying road traffic signs is proposed. Such solution could be used in autonomous vehicle production, and also similar solutions could easily be implemented in any other application that requires image object recognition.

KEYWORDS

Artificial neural network, convolutional neural network, classification, traffic sign

1. INTRODUCTION

Computer Vision Science (CVS) as a part of Artificial Intelligence (AI) deals with fetching, processing, analysing and understanding images. Main goal of the CVS is to simulate human vision through utilizing AI algorithms, which is still far from possible.

Classical computing for long tried solving these problems by modelling problem specific algorithms. Toady Deep Learning (DL) techniques provide us solutions in which a computer can "learn" the solution algorithm without us having to explicitly model the problem. The main trade off being we need a set of labelled data on which the computer will pick up the statistical differences which will enable it to discriminate the objects into classes.

One of the major problems that AI tries to solve is categorizing the input data based on the previous inputs. An example of that is e-mail spam filtering or classifying e-mails into *spam* and *not spam* categories. There are several ways of training such models, one of them being *supervised learning* where the idea is to use pairs of input and output data to predict the output data as close as possible, and the other *unsupervised learning* where only input data is used, and the model is trying to spot statistical differences in the input data.

This document describes the use of Neural Networks (NN), more specific - Convolutional Neural Networks (CNN) in classifying traffic road signs. CNN was chosen as a method of solving this problem because is their great success in solving similar problems. In order to get some background information in NN and CNN field chapters 2 and 3 will go through some basic theory of NN and CNN, how they work, how they are trained and their basic components. After that in chapter 4 we will show a way of using CNN for solving the problem of classifying the traffic road signs.

2. ARTIFICIAL NEURAL NETWORK

Artificial Neural Network is a set of connected neurons in such a way that it loosely mimics human brain. The outputs of one layer are connected as the inputs of the next layer of neurons. An example of such network can be seen in Figure 1. A typical ANN is composed of 3 basic layers: *input layer*, *hidden layer* and *output layer*.

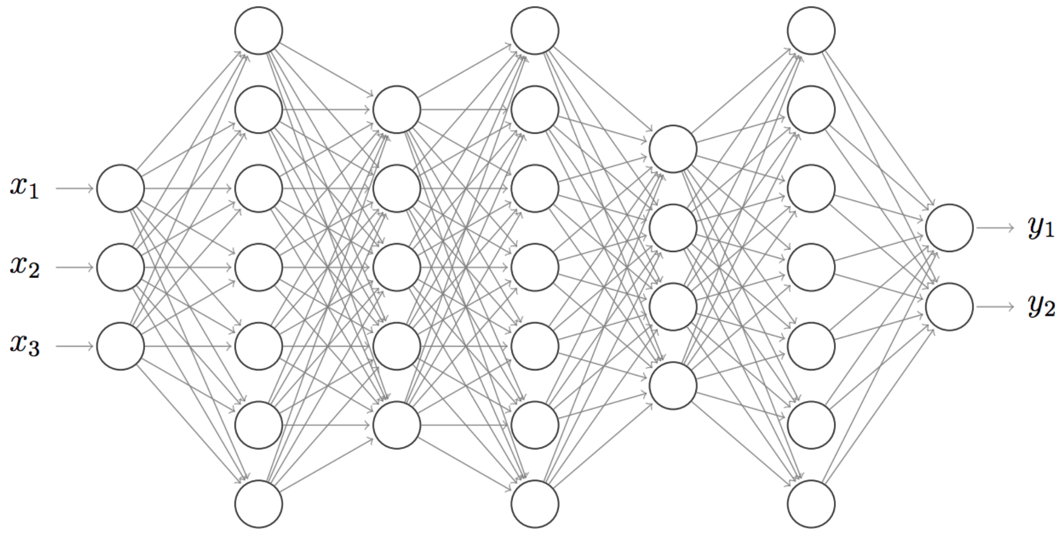

Figure 1. Set of connected neurons forming an ANN

Such network would get its inputs as a set of values [x_1, x_2, x_3] so we can consider the neurons labelled with x as the input layer of the network. Neurons of the input layer are connected to the neurons of the first layer of the hidden layer. The hidden layer neurons are connected to other neurons and their outputs are not visible as a network output (hence the term hidden). The hidden layer neurons are presented as unlabelled neurons in Figure 1. The outputs of the last layer of the hidden layer neurons are connected to the neurons of the output layer. The set [y_1, y_2] makes the output values of the network.

2.1. Neuron

An Artificial Neuron is a mathematical function conceived as a model of biological neurons. Artificial neurons are elementary units of an ANN. The neuron receives one or more inputs and sums them to produce an output (or activation). An example of such a neuron can be found in Figure 2.

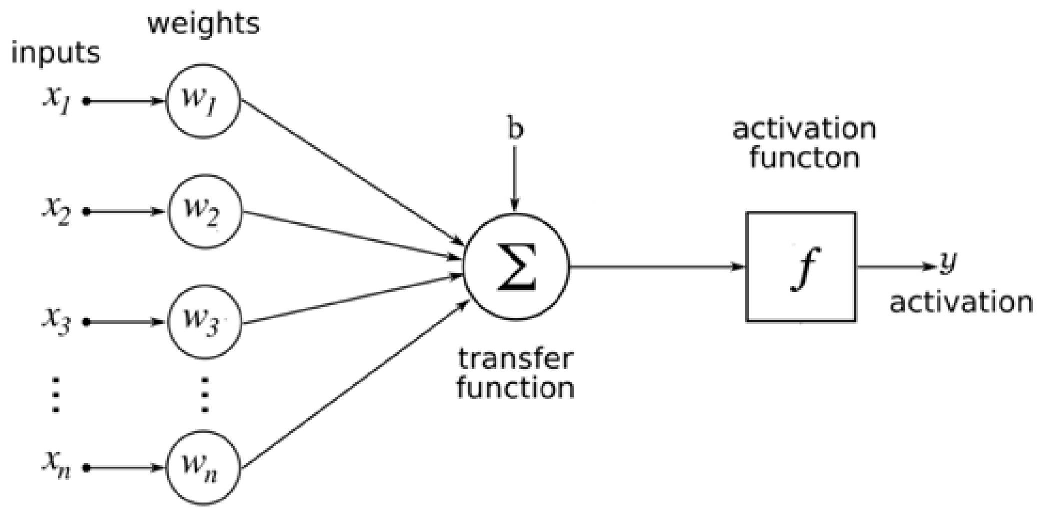

Figure 2. Neuron with inputs [x_1,..,x_n] and an output y

Mathematically the neuron from the Figure 2 is presented by the following expression:

$$y=f\left(\sum_{i=1}^{n} \square x_i \cdot w_i + b\right)$$

The expression holds two unknown values and an unknown function. Values [w_1,..,w_n] represent the weights of the corresponding inputs, value b is the bias and the function *f* is the activation function. The input values are first weighed, then they are put through the transfer function (in our case the transfer function is SUM - ©) ant lastly the output of the transfer function is passed through the activation function *f* which will be activated dependent of the value of the transfer function and the activation threshold to produce the output of the neuron. Depending of the application the transfer function must not only be the SUM function and functions like MAX, MIN, AVG, etc.

2.2. Activation function

Neuron without an activation function (or with an activation function *f* (*x*) = *x*) represents an ordinary linear combination of input parameters. Activation function must be nonlinear in order for the NN to learn nonlinear functions. Simply speaking outputs of the hidden layer neurons with linear activation function would all be linear combination of the input parameters. With nonlinear activation function neural network has far more possibilities over a simple perceptron network (a network only built with simple nodes only containing an input and an output).

There are many types of activation functions, two most commonly being used are:

- TANH – mapping $f:(-,)(-1,1):$

$$f(x)=\frac{e^{2x}-1}{e^{2x}+1}$$

and

- SIGMOID – mapping $f:(-,)(0,1)$, which could be used when we are expecting only positive outputs:

$$f(x)=\frac{1}{1+e^{-x}}$$

2.3. Training an Artificial Neural Network

The majority of practical machine learning uses supervised learning. Supervised learning is where you have input values and corresponding output values (input data has been labelled) and you use this data to learn the mapping function of the input to the output. The goal is to approximate the mapping function so well that when you introduce new input data you can predict the output values for that data. It is called supervised learning because the process of training the ANN can be taught as a teacher supervising the learning process. We know the correct answers, the ANN will repeatedly predict the answer, the error will be back propagated, and the ANN will know a bit more about the data. The learning stops when the ANN achieves an acceptable prediction success rate.

Unsupervised learning is where you only have input data and no corresponding output values. The goal for unsupervised learning is to model the underlying structure in the data in order to learn more about the data. It is called unsupervised learning because there is no corresponding output data and the error cannot be calculated and back propagated through the ANN. Unsupervised learning is commonly used to solve clustering problems.

2.3.1. Back propagation of errors

Back propagation is a form of supervised learning. It composes of two fazes *forward pass* and *backward pass*. A collection of weights makes the ANN model because it is an attempt to model data's relationship to training data labels. Model normally starts out bad and ends up less bad, changing over time as the ANN updates its parameters. This is because the ANN is born in ignorance, it does not know which weights and biases will translate the input data into desired result. So, it starts guessing. The first set of input is given to the ANN. It makes a prediction the best it can. The weights map the input to a set of guesses.

$$guess = input * weights$$

The ANN will then compare the predicted result with the actual truth value. The difference between the truth value and the predicted value is the ANN prediction error.

$$error = truth - guess$$

The network measures that error and walks the error back over its model, adjusting weights to the extent that they contributed to the error.

$$adjustment = error * weight$$

The ANN will then add the adjustment value to the corresponding weight, and the next time the ANN makes a prediction it will be that much closer to the truth result. This way we can fine tune the ANN to give better and better scores after each iteration. This algorithm is repeated until we are satisfied with the way the ANN predicts the result.

3. CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks are deep Artificial Neural Networks used primarily to classify images (name what they see), cluster them by similarity (photo search), and perform recognition with scenes. CNN are the algorithms that can identify faces, individuals, traffic signs, tumours and many other aspects of visual data. The efficiency of CNNs in image recognition is one of the main reasons why the world has woken up to the efficiency of deep learning algorithms.

CNN works with multidimensional data $x \in R^n \times R^n$ as opposed to regular ANN which works with scalar data. CNN ingest data as tensors, and tensors are matrices of numbers with additional dimensions. Neurons of the CNN convolution layers are called *feature maps* and their weights are called *kernels*.

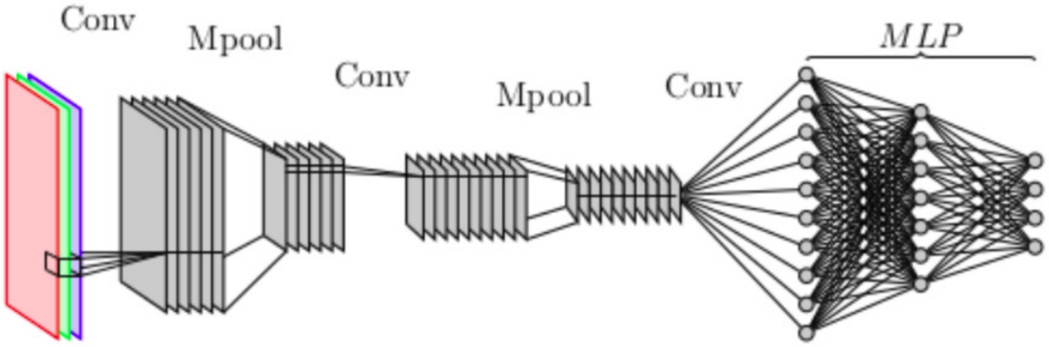

Figure 3. An architectural example of a Convolutional Neural Network

An example of a basic CNN structure can be seen in Figure 3. The input of such a network can either be a 3-channel coloured image or a one channel monochromatic image. The inputs width and height are the same of the image width and height and the colour channels dictate the depth of the input data. If the image is 300x300 pixels in width and height and images colours are written in RGB format and we input each colour as a single channel, then the dimensions of input data would be 300x300x3. Since monochromatic images only have one colour the dimensions of such input data would be 300x300x1. After the input layer come alternately connected *convolution layers* and *pooling layers.* After the convolutional and pooling layers come the fully-connected layers much like a regular NN and it is responsible to compute the score. In this way CNN transforms the original image layer by layer from the original pixel values to the final score. There are two types of layers found in CNN:

- Convolutional layer
- Pooling layer

3.1. Convolutional layer

Convolutional layer is the core building block of a CNN that does most of the computational work. The parameters of a convolutional layer consist of a set of learnable filters (*kernels*). Every filter is small spatially but extends the full depth of the input volume. An example of such filter would be a filter with dimensions 5x5x3 (5 pixels for width and height and depth 3 because the image has 3 channels RGB). During the forward pass we convolve (slide) each filter across the width and height of the input data and compute dot products between the entries of the filter and the input at any position. The result of this convolution will be a 2-dimensional activation map that gives the responses of that filter at every spatial position. The CNN will learn the filters that activate when they see some type of visual feature. Now that we have a set of all the filters from all of the layers we have the CNN model on which we can test our results. An example of a convolutional layer can be seen in Figure 4.

3.1.2. Local connectivity

When dealing with high-dimensional inputs such as images it is impractical to connect a neuron with every neuron from the next layer. Instead we connect each neuron to only a local group of the input volume. The spatial extent of this is a parameter called the *receptive field* of the neuron. The extent of connectivity along the depth is always equal to the depth of the input data.

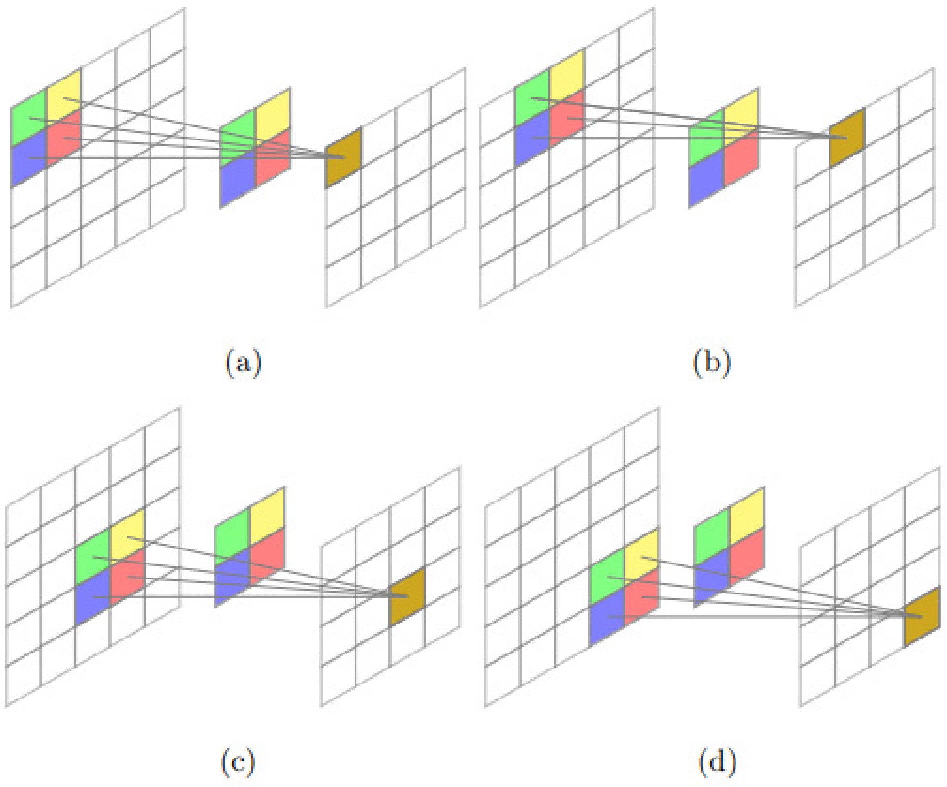

Figure 4. Graphical convolution example

3.2. Pooling layer

Pooling layers have a function of reducing the size of the of the filter outputs and are usually periodically placed between the convolutional layers of the Convolutional Neural Network. The task of reducing the number of outputs is needed in order to reduce the number of parameters and calculations in the network, and also to prevent *over fitting* the network. The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2x2, applied with a stride of 2 down samples every depth slice in the input by 2 along both width and height, discarding 75% of activators. The depth dimension remains unchanged.

There are several types of pooling functions to choose from, some of them being:

- MAX pooling
- MIN pooling
- AVG pooling
- L2-norm pooling

An example of pooling layer can be seen in Figure 5.

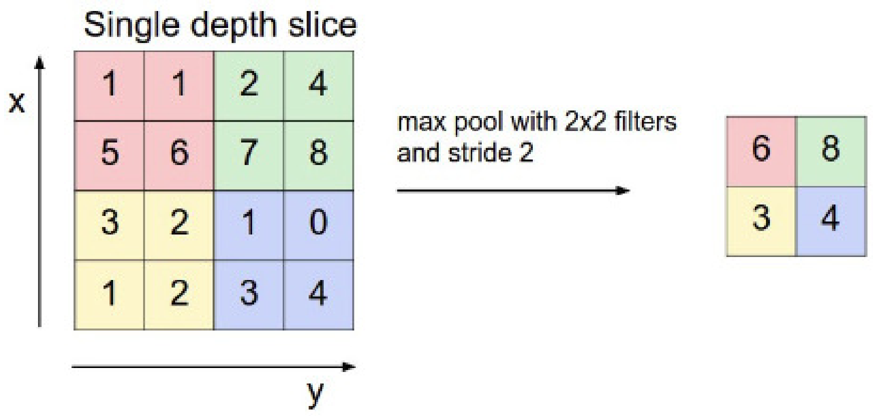

Figure 5. Graphical pooling layer example using max pool filter with

When using MAX pooling function, it is common to keep an index of the max activation, so the back propagation could be done more efficiently. Also, since during the forward propagation only the max values are taken into account the back propagation will update only those max values when error correcting. Many people dislike the pooling operation and think they can go without it. Common ways of getting rid of pooling layer is to have convolutional layers with larger stride so the convolution would in effect reduce the size of the output dramatically.

3.3. Convolutional Neural Network parameters

Convolutional Neural Network parameters (also called hyperparameters) are a set of variables used to control the network. It is a must that the hyperparameters are initializes before training the network. Hyperparameter optimization is a process of finding close-to-optimal CNN parameters. Common hyperparameters in CN networks are:

- **Learning rate** is one of the most important parameters of a neural network. ANN convergence is dependent on this parameter. Optimizing these parameters is possible by using random values and training the network and afterwards choosing the best parameter value from the lot

- **Epoch number** parameter can easily be optimized using early stop technique, simply put you would print the error after each epoch you can choose how long you will train you network

- **CNN architecture** presents a set of decisions which model our network. Choosing the right number and orientations in the network, choosing the input data dimensions, kernel dimensions, strides, etc… It is needed to construct the network in such a way that the network is large enough to solve our particular problem, but not too large so we get a sub-performing network

- **Activation function** is in most cases the same value across the ANN. In CNN applications most commonly used activation function is ReLU (Rectifier Linear Units)

- **Weights initialization** should be done very carefully in order not to slow down the network in the beginning stages of training process. Commonly used weight initialization function is *uniform distribution*

- **Cost function** is an important ANN parameter on which the convergence of the ANN is dependent on. It is used to calculate the error between the output and the input data. Most commonly used cost function is *mean squared error.*

- **Strides** represents the value by which the filter is going to move along the width or height of the input data

- **Padding** represents adding an additional frame of empty pixels to the input data, most common of value 0. Padding is used to ensure the ANN does not neglect the outer pixels

4. TRAFFIC SIGNS CLASSIFICATION USING CNN

Recognising traffic sign is a huge step in producing reliable autonomous vehicles. This chapter will show you how we implemented ANN based solution to classifying traffic signs. Since we already covered the basic theory in previous chapter the assumption is that the reader knows the basics about ANN.

The dataset consists of 43 different traffic sign classes and each image being of size 30x30 pixels totalling about 1200 images in total. The dataset contained an equal amount of images of each class. All of the data from the dataset was normalized to [0,1] values. All of the images are RGB coloured meaning their full dimension is 30x30x3. The dataset was split into two sets:

- **Training set** consisting of about 80% of the dataset, and

- **Test set** consisting of the rest 20% of the dataset

4.1. The architecture

After trying several different CNN architecture examples we decided to go with the following architecture based on the LeNet.

- Input with dimensions 30x30x3

- First CONVOLUTION layer with 32 output filters, kernel size 3x3, stride 1x1, non-biased

- First MAXPOOL layer with kernel size 2x2

- Second CONVOLUTION layer with 64 output filters, kernel size 5x5, stride 1x1, non-biased

- Second MAXPOOL layer with kernel size 2x2
- First FULLY CONNECTED layer with 1600 inputs and 1200 outputs
- Second FULLY CONNECTED layer with 1200 inputs and 1024 outputs
- Third FULLY CONNECTED layer with 1024 inputs and 512 outputs
- Final OUTPUT layer with 512 inputs and 43 outputs (one output per class)

All layers are without padding and all layers having activation function ReLU. The network weights were initialized by using uniform distribution and the selected loss function was mean squared error. The implementation was performed in deeplearning4j framework library.

4.2. Results and discussion

We trained and evaluated the CNN several times and we averaged accuracy of about 80%. While there is still room for improvement we certainly showed that traffic sign classification using CNN and DL is possible. We noticed that the CNN had problems classifying similar shaped signs. For instance, the speed limit traffic signs. Both of them are circular signs and only differing in the value written in the centre of the sign. This can be caused by many factors like quality of the images, similar (if not the) same sign shapes. This could be fixed by adding new Convolutional Layers that could enable the network to learn even more.

5. CONCLUSION AND FUTURE WORK

The application of neural networks in traffic sign recognition has recently been a field of study. With this paper we showed that using Convolutional Neural Network in traffic sign classification is possible. This application, with some improvements, could be used in autonomous vehicle production and in traffic safety.

Future work would be to increase the prediction success rate of the Convolutional Neural Network described in the previous chapter. This could be achieved by using larger sets of data on which the CNN would learn. This means preparing more traffic sign images and feeding them to the CNN. Also, the performance of the CNN could be increased by using more layers, but this would mean the complexity of the network is dramatically increased and with that the time for the network to learn is also increased. Also, some type of object localization algorithms could be used with this CNN in order for the network to be capable of working with video stream.

ACKNOWLEDGEMENTS

This work was supported by the Ministry of Education and Science, Republic of Serbia, Science and Technological Development grant III 43002.

REFERENCES

[1] W.A. Awad and S.M. ELseuofi, “Machine Learning Methods for Spam e-mail Classification”, International Journal of Computer Science & Information Technology, 2011

[2] V.R.Kulkarni, Shaheen Mujawar1 and Sulabha Apte, “Hash Function Implementation Using Artificial Neural Network”, International Journal on Soft Computing, 2010

[3] Santaji Ghorpade, Jayshree Ghorpade and Shamla Mantri, "Pattern Recognition Using Neural Networks", International Journal of Computer Science & Information Technology, 2010

[4] Andrej Karpathy, Convolutional Neural Networks for Visual Recognition, Stanford University, 2016

[5] Amritpal Kaur, Madhavi Arora, "Neural network based Numerical digits Recognization using NNT in Matlab", International Journal of Computer Science, 2013

[6] Matija Folnović, "Deep neural architectures for object localization", Sveučilište u Zagrebu, 2015

[7] Aravindh Mahendran, Andrea Vedaldi, "Understanding Deep Image Representations by Inverting Them", Computer Vision Foundation, 2015

[8] https://www.coursera.org/specializations/deep-learning

[9] https://deeplearning4j.org/

[10] http://benchmark.ini.rub.de/?section=gtsrb&subsection=news

AUTHORS

Nemanja Veličković is currently pursuing a M.Tech. degree in Computer Science and Engineering at Računarski fakultet, University Union, School of Computing, Belgrade, Serbia and has completed a B.E in Computer Science and Engineering also at Računarski fakultet. Currently working as a full stack developer at Ticketmaster.