**Implementation of Drowsiness Detection System using Convolutional Neural Network**

**Abstract**:

Driver drowsiness has become one of the leading causes of car accidents in recent years resulting in serious physical injuries, fatalities and substantial financial losses. According to statistics, are liable driver drowsiness detection system is needed to warn the driver before a collision occurs. Behavioural, physiological and vehicle based measures are used by researchers to assess driver drowsiness. A thorough examination of these measures will shed light on the current systems, their problems, and the improvements that must be made in order to create a reliable system. By doings other measures will reveal existing systems, their flaws and the changes that must be made in order to produce a credible system.

**Introduction**

According to available statistics, each year people die on the road is nearly 1.3 million and around 50 million peoples are suffering from non-fatal injuries as a result of traffic accidents. At the wheel if a driver falls asleep, the car loses its control and crashes into another vehicle or stationary objects. The driver's level of drowsiness should be monitored to stop these fatal accidents. The following tools have been widely used to track drowsiness:

1)Vehicle based measures: Deviations from lanes, steering wheel rotation, accelerator pedal pressure, and other factors are continuously tracked, and any change that exceeds a predetermined level signals indicates that the driver is drowsy.

2) Behavioural measures: A camera tracks the driver's actions, such as eye closing, eye twitch, head posture, and yawning, and if any of these drowsiness signs are observed the driver is alerted.

3)Physiological measures: Many studies have looked into the relationship between physiological signals (ECG,EMG,EoG and EEG) and driver drowsiness. One of the important safety features is drowsiness detection that can avoid accidents caused by driver’s drowsiness.

Also, another common issue with the current machine vision models is the fact that most of these algorithms are bulky ( large in size) and require dedicated hardware to run the models that have been developed. They do not run efficiently on devices with low computational power

The aim of this study is to detect and alert the person when eyes are not opened for a particular duration with the use of efficient and fast CNN model. When drowsiness is detected, this system will notify the about the drowsiness of the driver.

**LITERATURE REVIEW**

As sleep-induced crashes represent a significant portion of vehicle crashes in the world, researchers and automotive companies have created different solutions ranging from finding patterns in driving habits to analyzing brain waves and vitals of the driver while driving. Most of these solutions are backed by some predictive algorithms powered by statistics and machine learning. The most common ones can be broadly divided into three categories as described in the following paragraphs. One approach is to find changes in vehicle behavior, such as the one proposed by McDonald et al. [6]. He created a contextual and temporal algorithm that utilizes the steering angle, vehicle speeds, and accelerator pedal positions. These values are passed into a Bayesian Network which determines if a driver shows characteristics of drowsy behavior. The algorithm was found to have lower false-positive rates than PERCLOS [7] methods, which predicts drowsiness based on eyelid movements and patterns. The takeaway from this study was that to predict correctly, the context of the situation is crucial. The data that it captures over a previous 10-second period is vital in understanding whether the person is at risk of drowsiness related lane departures. A second approach is based on studies focused on using the drivers’ vitals, brain waves, and readings from ElectroEncephalo Grams (EEG’s) to make predictions. Wei et al. [8] made comparisons between non-hair bearing EEG Brain Computer Interfaces that are easy to wear and less intrusive than the lab-based whole scalp EEG’s which are less comfortable. The study showed that non-hair bearing devices had no significant reduction in performance when compared to wholescalp EEG. Thereby with this finding one may develop less intrusive and comfortable headbands.

EEG alone is unable to detect all stages of drowsiness, so Kartsch et al. [9] used EEG with Inertial Measurements Units (IMU) sensors to detect 5 levels of drowsiness with about 95% accuracy. The team fused behavioral information from IMU and EEG information to detect drowsiness. Another drawback of the EEG system was the power requirements of these devices. Their technology has also facilitated the implementation of a parallel ultra-low power (PULP) platform on a microcontroller which extended the battery life to almost 46 hours, thereby creating devices that are always wearable and require low maintenance. Tateno et al. [10] developed a system that just uses heart rate monitoring to detect the respiration of a person and thereby detect drowsiness. The methodology was found to be an effective predictor for respiration and thereby drowsiness. Yet another technique is to utilize the power of computer vision. The recent breakthroughs in Deep Learning have provided new tools to computer vision for detection and classification. Computer vision-related applications utilize these methods in object detection, health and wellness, and even agricultural applications too [11]. A big impact in this space has been for imaging data. Since drivers facial features change significantly once he gets tired, computer vision scientists have attempted to capitalize on this and use it to provide solutions for drowsiness detection.

Tayab Khan et al. [12] proposed a solution to measure the angle of eyelid curvature and thereby identify if the eyes are closed or not. They achieved 95% accuracy with this method, but the limitation is that there needs to be enough light for the method to work as it functions poorly at night. Shakeel et al. [13] used MobileNet-SSD architecture to train a custom dataset of 350 images. The model was capable of achieving a Mean Average Precision of 0.84. The system was costeffective and efficient as the algorithm could be deployed in an Android device and the camera stream could be classified in real-time.

Celona et al. [14] proposed a vision-based Multi-Task Driver Monitoring Framework that analyzes eyes, mouth, and the pose of the head simultaneously to predict the level of the drowsiness. This study was conducted on the NTHU [15] dataset. Another study conducted by Xie et al. [16] used transfer learning and sequential learning from yawning video clips to detect yawning on the YawDD and NTHU-DDD database. This system was able to have higher precision and was robust to changes in the position and angle of the face to the camera. Mehta et al. [17] developed an Android application that is capable of detecting facial landmarks and then computing the Eye Aspect Ratio (EAR) and Eye Closure Ratio (ECR) to predict drivers drowsiness based on machine learning models with an accuracy of 84%

**Methodology**

Driver exhaustion and drowsiness are significant contributors to various automobile accidents. In the field of accident prevention systems, designing and maintaining technology that can effectively identify or avoid drowsiness at the wheel and warn the driver before a collision is a major challenge. We use OpenCV to take images from a webcam and these images given to a deep learning algorithm that can tell whether someone's eyes are closed or opened. In this case, we are looking for the persons face and eyes.

Step1: Image is taken as input from camera. We'll use a camera to capture photographs as input. But, in order to gain access to the webcam, we created an endless loop that captures each frame. We employ the cv2 method given by OpenCV. VideoCapture(0) (cap) is used to access the camera and capture the object. With cap.read(), each frame is read, and then image is saved in a variable.

Step 2:Create a ROI by detecting a face in the picture. To segment the face in the captured image, we first converted it to gray scale because, the OpenCV object detection algorithm only accepts grayscale images as input. To detect the objects, we don't need colour detail. We use the Haar cascade classifier to detect the face. The classifier face= cv2.Cas is set with this section. for (x,y,w,h)infaces,we use cv2.rectangle(frame, (x,y), (x+w, y+h), (100,100,100), 1 Step 3: Use the ROI to find the eyes and feed them to the classifier.

The technique for detecting eyes is the same as for detecting ears. Cascade classifier is used in left and right eyes. Then, use left\_eye=leye.detectMultiScale(gray) to detect the eyes. We extracted only the details of eyes from the captured image. This can be done by first removingthe eye's boundary box and then using this code to remove the eye image from the picture. l\_eye = frame[y : y+h, x : x+w] This information is given to CNN, which decides whether the eyes are closed or not. The right eye also detected in the above manner.

**FLOW DIAGRAM:**

**DATASET**

**IMAGE PREPROCESSING**

**DATA AUGMENTATION**

**NEURAL NETWORK TRAINING**

**TESTING TRAINED MODEL WITH VALUATION DATA**

**PROCESSES IN CNN:**

**Dataset collection:**

Appropriate datasets are required at all stages of object recognition research, starting from training phase to evaluating the performance of recognition algorithms. All the images collected for the dataset were downloaded from the Internet, searched by name on various sources in different languages.

**Image Preprocessing and Labelling:**

Images downloaded from the Internet were in various formats along with different resolutions and quality. In order to get better feature extraction, final images intended to be used as dataset for deep neural network classifier were preprocessed in order to gain consistency. Furthermore, procedure of image preprocessing involved cropping of all the images manually,in order to highlight the region of interest.

**Augmentation Process:**

The main purpose of applying augmentation is to increase the dataset and introduce slight distortion to the images which helps in reducing overfitting during the training stage.

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

Training deep learning neural network models on more data can result in more skillful models, and the augmentation techniques can create variations of the images that can improve the ability of the fit models to generalize what they have learned to new images.

**Neural Network Training:**

The main goal of training the network is for neural network to learn the features that distinguish one class from the others. Therefore, when using more augmented images, the chance for the network to learn the appropriate features has been increased.

**Testing Trained Model With Valuation Data:**

Finally the trained network is used to detect the classes by processing the input images in valuation dataset and results are processed.

**SYSTEM SPECIFICATION**

**Software Used:**

* Operating System : Windows 7 / 8/ 10
* Language : Python
* IDE : Anaconda, Notebook

**Hardware Used:**

* Processor : Intel core i3
* Ram : 8 GB
* Hard Disk : 120 GB

**Anaconda (Python distribution)**

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution includes data-science packages suitable for Windows, Linux, and MacOS.

**Overview**

Anaconda distribution comes with more than 1,500 packages as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface (CLI).

The big difference between conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason conda exists.

When pip installs a package, it automatically installs any dependent Python packages without checking if these conflict with previously installed packages. It will install a package and any of its dependencies regardless of the state of the existing installation. Because of this, a user with a working installation of, for example, Google Tensorflow, can find that it stops working having used pip to install a different package that requires a different version of the dependent numpy library than the one used by Tensorflow. In some cases, the package may appear to work but produce different results in detail.

In contrast, conda analyses the current environment including everything currently installed, and, together with any version limitations specified (e.g. the user may wish to have Tensorflow version 2,0 or higher), works out how to install a compatible set of dependencies, and shows a warning if this cannot be done.

Open source packages can be individually installed from the Anaconda repository[8], Anaconda Cloud (anaconda.org), or your own private repository or mirror, using the conda install command. Anaconda Inc compiles and builds all the packages in the Anaconda repository itself, and provides binaries for Windows 32/64 bit, Linux 64 bit and MacOS 64-bit. Anything available on PyPI may be installed into a conda environment using pip, and conda will keep track of what it has installed itself and what pip has installed.

Custom packages can be made using the conda build command, and can be shared with others by uploading them to Anaconda Cloud,[9] PyPI or other repositories.

The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, it is possible to create new environments that include any version of Python packaged with conda.

**Anaconda Navigator**

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

The following applications are available by default in Navigator:

JupyterLab

Jupyter Notebook

QtConsole

Spyder

Glue

Orange

RStudio

Visual Studio Code

Conda:

Conda is an open source, cross-platform, language-agnostic package manager and environment management system that installs, runs, and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language (e.g., R), including multi-language projects. The conda package and environment manager is included in all versions of Anaconda, Miniconda, and Anaconda Repository.

**Anaconda Cloud**

Anaconda Cloud is a package management service by Anaconda where you can find, access, store and share public and private notebooks, environments, and conda and PyPI packages.[20] Cloud hosts useful Python packages, notebooks and environments for a wide variety of applications. You do not need to log in or to have a Cloud account, to search for public packages, download and install them.

You can build new packages using the Anaconda Client command line interface (CLI), then manually or automatically upload the packages to Cloud.

**Jupyter Notebook**

This tutorial explains how to install, run, and use Jupyter Notebooks for data science, including tips, best practices, and examples.

As a web application in which you can create and share documents that contain live code, equations, visualizations as well as text, the Jupyter Notebook is one of the ideal tools to help you to gain the data science skills you need.

This tutorial will cover the following topics:

* A basic overview of the Jupyter Notebook App and its components,
* The history of Jupyter Project to show how it's connected to IPython,
* An overview of the three most popular ways to run your notebooks: with the help of a Python distribution, with pip or in a Docker container,
* A practical introduction to the components that were covered in the first section, complete with examples of Pandas DataFrames, an explanation on how to make your notebook documents magical, and answers to frequently asked questions, such as "How to toggle between Python 2 and 3?", and
* The best practices and tips that will help you to make your notebook an added value to any data science project!

## What Is A Jupyter Notebook?

In this case, "notebook" or "notebook documents" denote documents that contain both code and rich text elements, such as figures, links, equations, ... Because of the mix of code and text elements, these documents are the ideal place to bring together an analysis description, and its results, as well as, they can be executed perform the data analysis in real time.

The Jupyter Notebook App produces these documents.

We'll talk about this in a bit.

For now, you should know that "Jupyter" is a loose acronym meaning Julia, Python, and R. These programming languages were the first target languages of the Jupyter application, but nowadays, the notebook technology also supports many other languages.

And there you have it: the Jupyter Notebook.

As you just saw, the main components of the whole environment are, on the one hand, the notebooks themselves and the application. On the other hand, you also have a notebook kernel and a notebook dashboard.

Let's look at these components in more detail.

### What Is The Jupyter Notebook App?

As a server-client application, the Jupyter Notebook App allows you to edit and run your notebooks via a web browser. The application can be executed on a PC without Internet access, or it can be installed on a remote server, where you can access it through the Internet.

Its two main components are the kernels and a dashboard.

A kernel is a program that runs and introspects the user’s code. The Jupyter Notebook App has a kernel for Python code, but there are also kernels available for other programming languages.

The dashboard of the application not only shows you the notebook documents that you have made and can reopen but can also be used to manage the kernels: you can which ones are running and shut them down if necessary.

## The History of IPython and Jupyter Notebooks

To fully understand what the Jupyter Notebook is and what functionality it has to offer you need to know how it originated.

Let's back up briefly to the late 1980s. Guido Van Rossum begins to work on Python at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Fast forward two years: the IPython team had kept on working, and in 2007, they formulated another attempt at implementing a notebook-type system. By October 2010, there was a prototype of a web notebook, and in the summer of 2011, this prototype was incorporated, and it was released with 0.12 on December 21, 2011. In subsequent years, the team got awards, such as the Advancement of Free Software for Fernando Pérez on 23 of March 2013 and the Jolt Productivity Award, and funding from the Alfred P. Sloan Foundations, among others.

Lastly, in 2014, Project Jupyter started as a spin-off project from IPython. IPython is now the name of the Python backend, which is also known as the kernel. Recently, the next generation of Jupyter Notebooks has been introduced to the community. It's called JupyterLab.

After all this, you might wonder where this idea of notebooks originated or how it came about to the creators.

A brief research into the history of these notebooks learns that Fernando Pérez and Robert Kern were working on a notebook just at the same time as the Sage notebook was a work in progress. Since the layout of the Sage notebook was based on the layout of Google notebooks, you can also conclude that also Google used to have a notebook feature around that time.

For what concerns the idea of the notebook, it seems that Fernando Pérez, as well as William Stein, one of the creators of the Sage notebook, have confirmed that they were avid users of the Mathematica notebooks and Maple worksheets. The Mathematica notebooks were created as a front end or GUI in 1988 by Theodore Gray.

The concept of a notebook, which contains ordinary text and calculation and/or graphics, was definitely not new.

Also, the developers had close contact with one another and this, together with other failed attempts at GUIs for IPython and the use of "AJAX" = web applications, which didn't require users to refresh the whole page every time you do something, were two other motivations for the team of William Stein to start developing the Sage notebooks.

**Convolutional Neural Networks Working:**

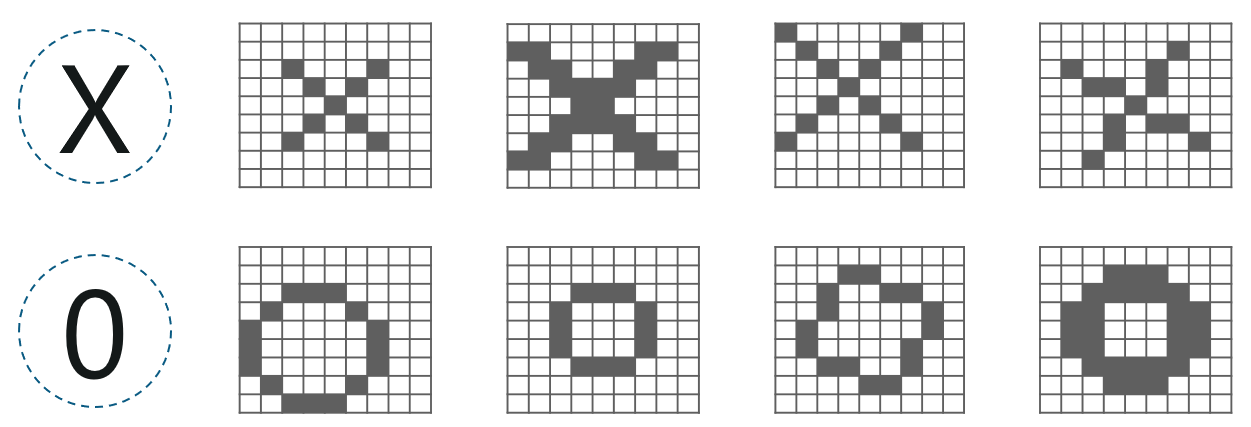
There are **four** layered **concepts** we should understand in Convolutional Neural Networks:

1. Convolution,
2. ReLu,
3. Pooling and
4. Full Connectedness (Fully Connected Layer).

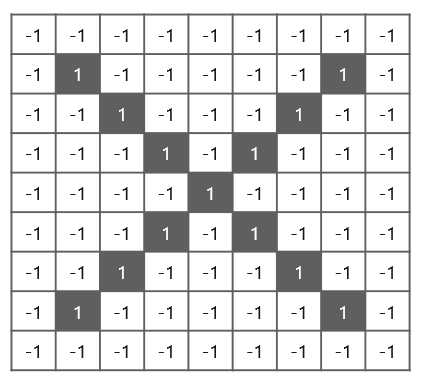
Let’s begin by checking out a **simple example:**

**Example of CNN:**

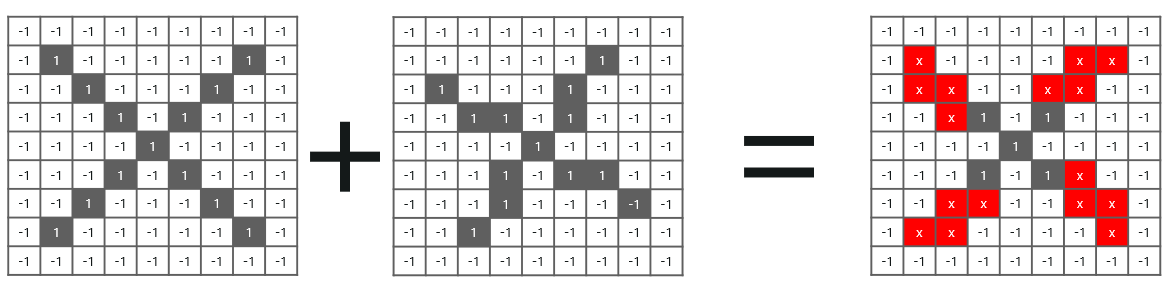
Consider the image below:



Here, there are multiple renditions of X and O’s. This makes it tricky for the computer to recognize. But the goal is that if the **input signal** looks like **previous**images it has seen before, the **“image” reference** signal will be mixed into, or **convolved** with, the **input** signal. The resulting **output** signal is then passed on to the **next layer.**

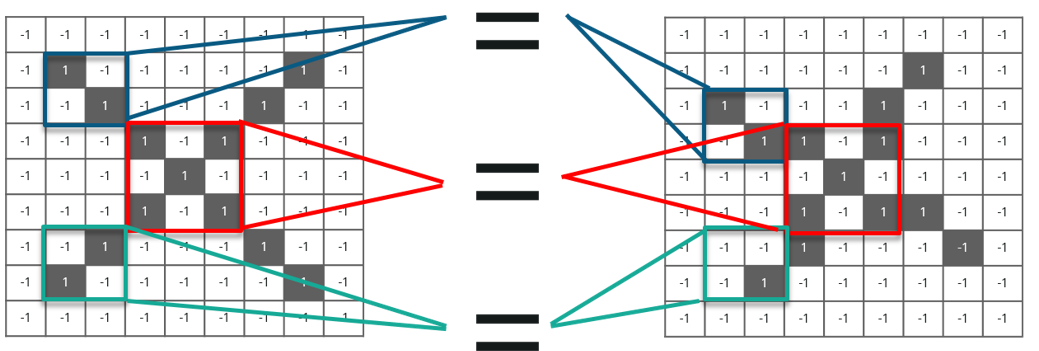


So, the **computer understands** every pixel. In this case, the **white** pixels are said to be **-1** while the **black** ones are **1.** This is just the way we’ve implemented to **differentiate the pixels** in a basic binary classification.



Now if we would just **normally search** and **compare** the **values** between a normal image and another **‘x’ rendition,** we would get a **lot** of **missing pixels.**

**So, how do we fix this?**



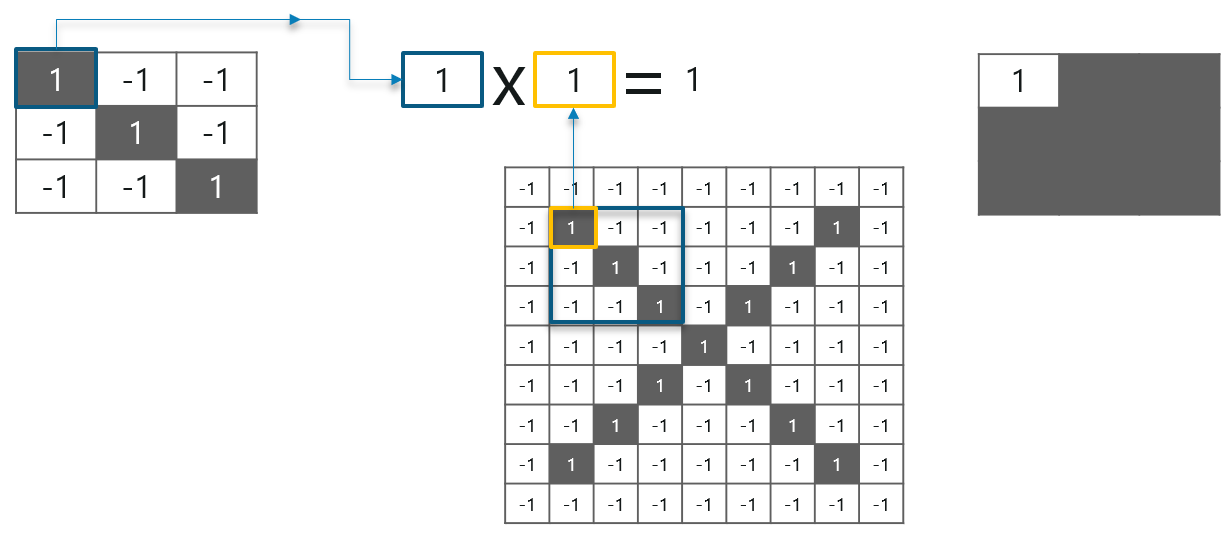
We take **small patches** of the pixels called **filters** and try to **match** them in the corresponding **nearby** locations to see if we get a **match.** By doing this, the Convolutional Neural Network **gets a lot better** at seeing **similarity** than directly trying to match the **entire image.**

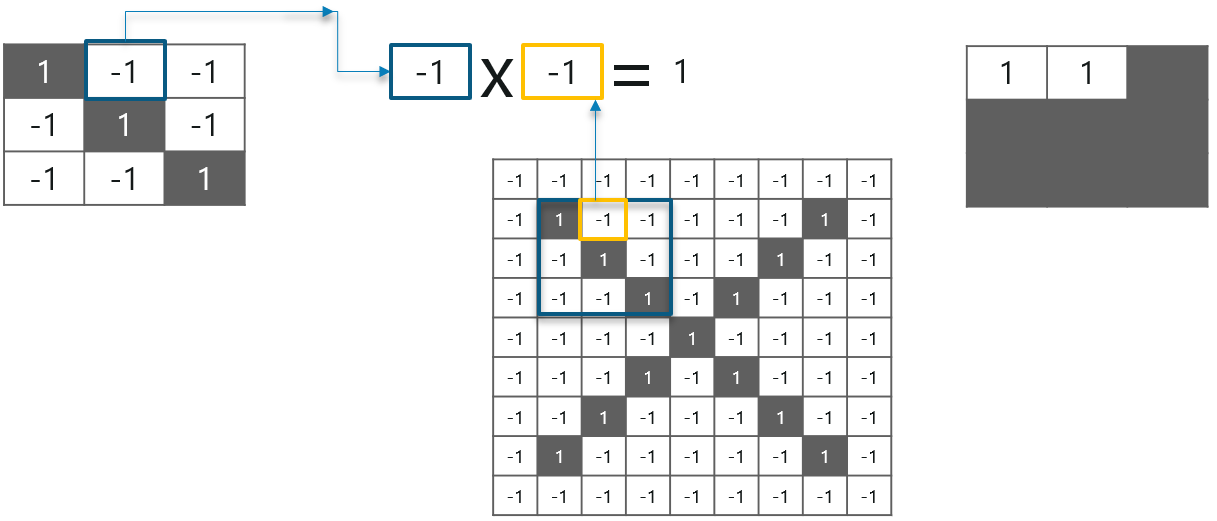
**Convolution Of An Image**

Convolution has the nice property of being **translational invariant**. Intuitively, this means that **each** convolution filter represents a **feature** of interest (e.g **pixels in letters)** and the Convolutional Neural Network **algorithm** learns which **features** comprise the **resulting reference** (i.e. alphabet).

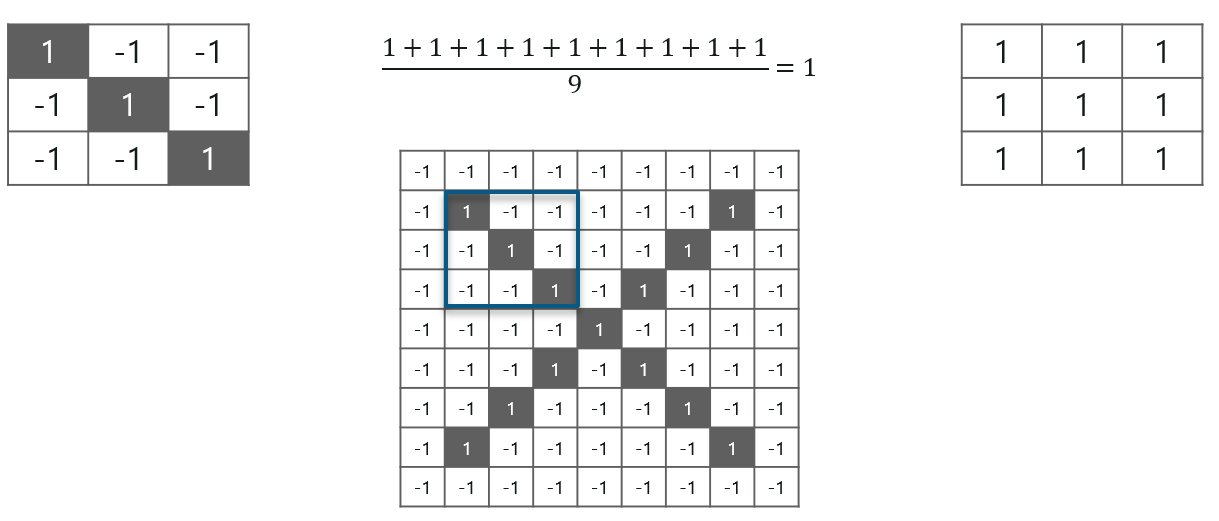
We have **4 steps** for convolution:

* **Line up** the feature and the image
* **Multiply** each **image** pixel by corresponding **feature** pixel
* **Add** the values and find the **sum**
* **Divide** the sum by the **total** number of pixels in the **feature**

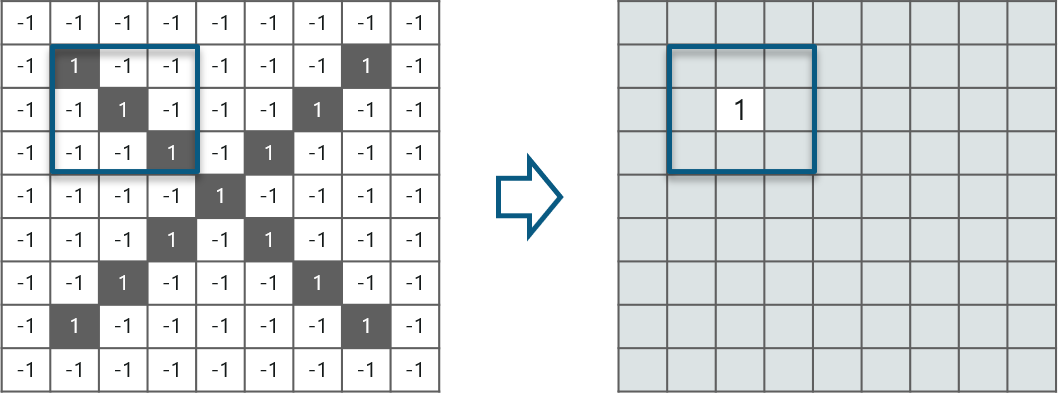




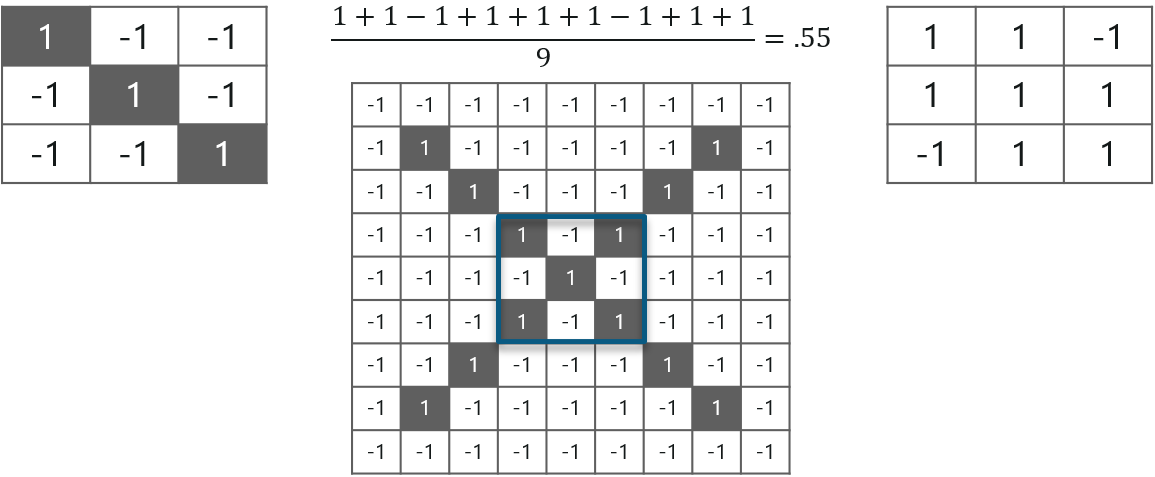
Consider the above image – As you can see, we are **done** with the first**2 steps**. We considered a **feature image** and **one pixel** from it. We **multiplied** this with the**existing image** and the product is stored in another **buffer feature image**.



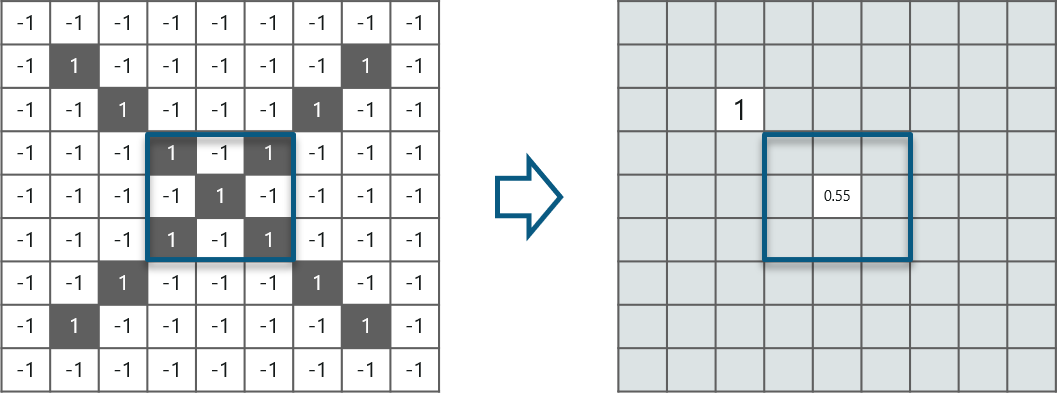
With this **image,** we completed the l**ast 2 steps.** We added the **values** which led to the **sum.** We then, **divide** this **number** by the **total** number of pixels in the **feature image.** When that is done, the **final value** obtained is placed at the **center** of the **filtered image** as shown below:



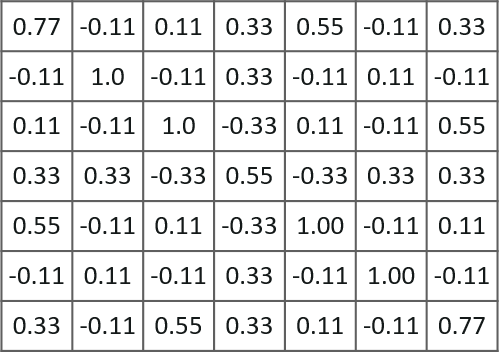
 Now, we can **move** this **filter** around and do the **same** at **any pixel** in the image. For **better clarity,** let’s consider **another example:**



As you can see, here after performing the first 4 steps we have the value at 0.55! We take this value and place it in the image as explained before. This is done in the following image:



Similarly, we move the feature to every other position in the image and see how the feature matches that area. So after doing this, we will get the output as:



Here we considered just one filter. Similarly, we will perform the same convolution with every other filter to get the convolution of that filter.

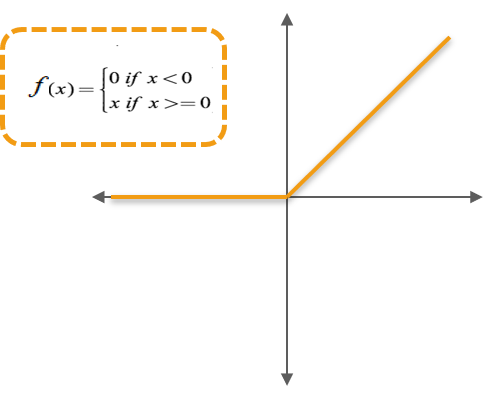
The **output** signal **strength** is not dependent on where the **features** are located, but simply whether the **features** are **present.** Hence, an alphabet could be sitting in **different positions** and the **Convolutional Neural Network** algorithm would still be able to **recognize it.**

**ReLU Layer**

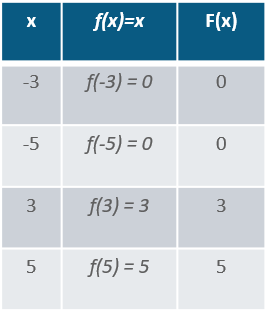
ReLU is an activation function. But, what is an activation function?

**Rectified Linear Unit**(ReLU) transform function only activates a node if the input is above a certain quantity, while the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable.

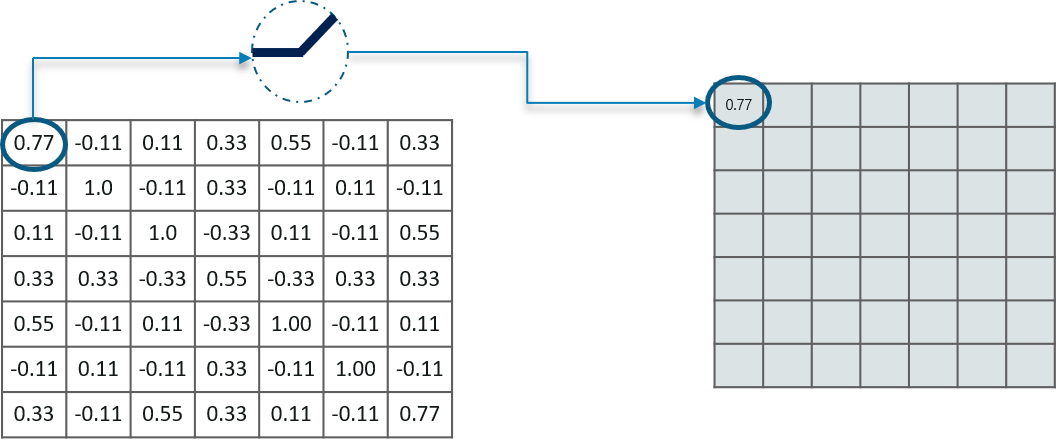
Consider the below example:



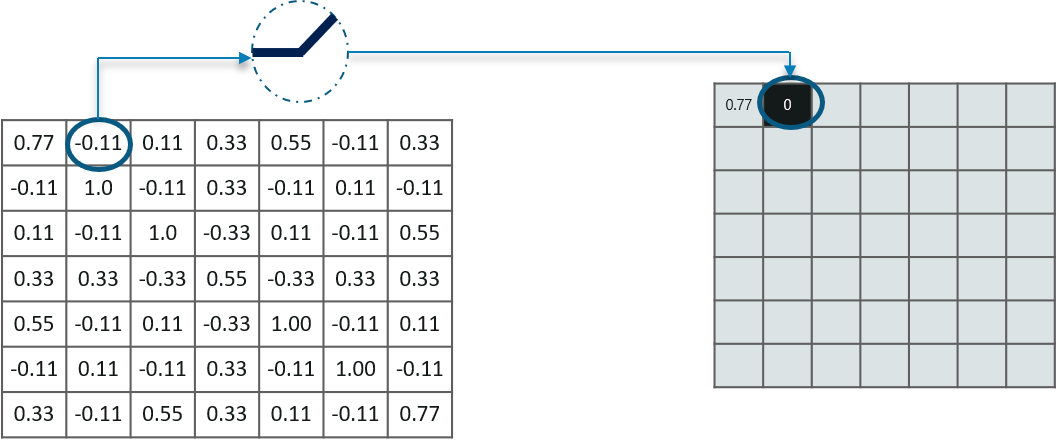
We have considered a simple function with the values as mentioned above. So the function only performs an operation if that value is obtained by the dependent variable. For this example, the following values are obtained:

****

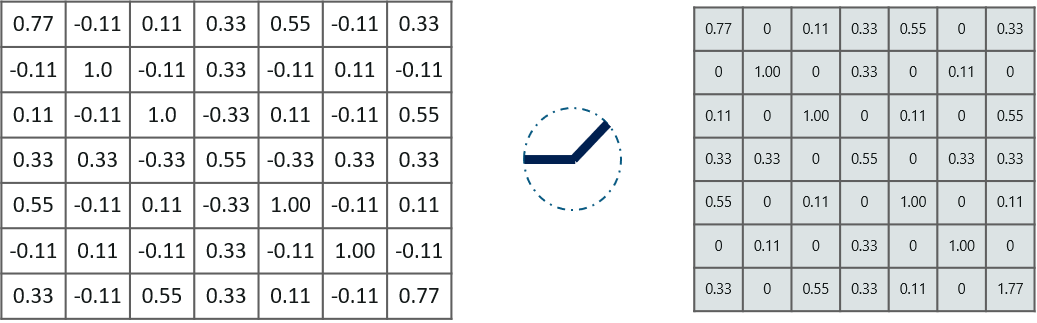
**Why do we require ReLU here?**

****

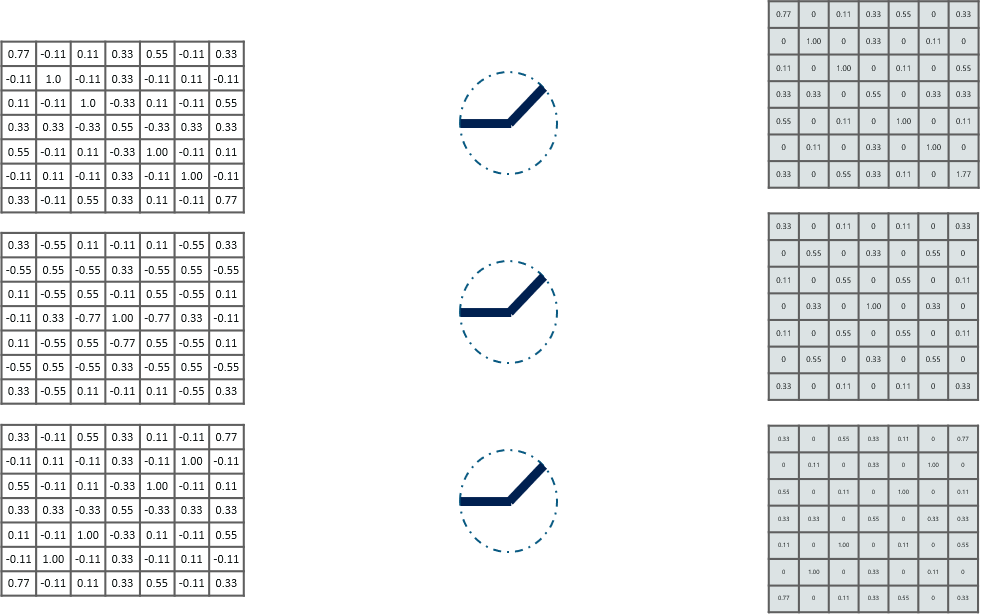
The main aim is to remove all the negative values from the convolution. All the positive values remain the same but all the negative values get changed to zero as shown below:

****

So after we process this particular feature we get the following output:

****

Now, similarly we do the same process to all the other feature images as well:



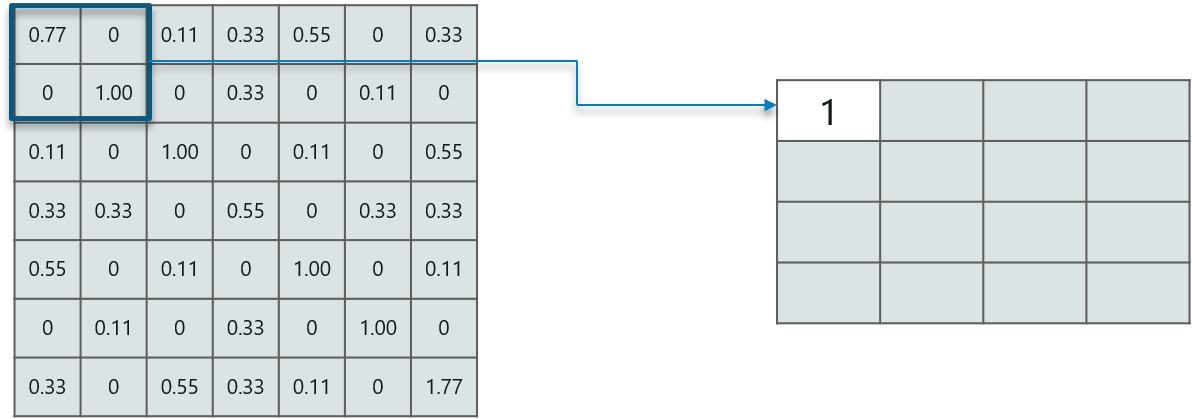
**Inputs** from the convolution layer can be **“smoothened”** to **reduce** the **sensitivity** of the **filters** to **noise** and **variations.** This smoothing process is called **subsampling** and can be **achieved** by taking **averages** or taking the **maximum** over a **sample** of the signal.

**Pooling Layer**

In this layer we **shrink** the **image** stack into a **smaller size.** Pooling is done **after passing** through the **activation** layer. We do this by implementing the following 4 steps:

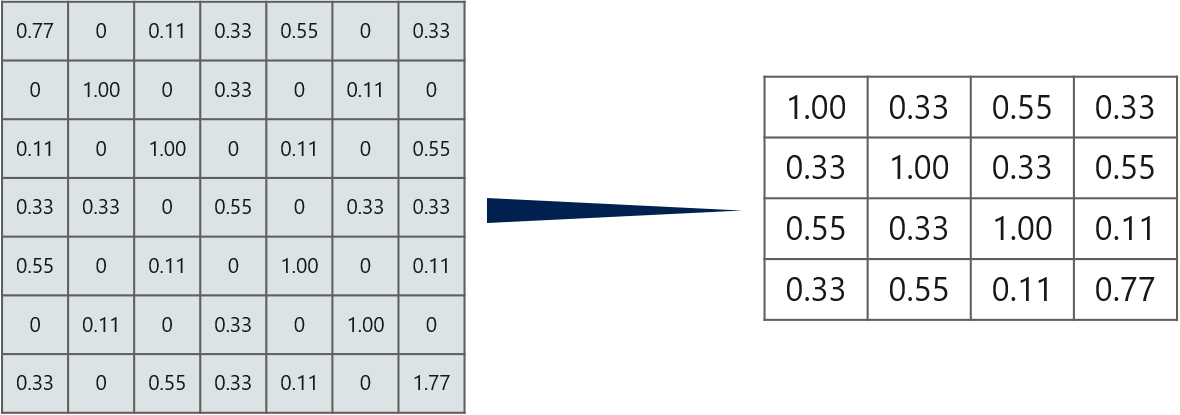
* Pick a **window size** (usually 2 or 3)
* Pick a **stride** (usually 2)
* **Walk** your window **across** your **filtered** images
* From each **window,** take the **maximum** value

Let us understand this with an example. Consider performing pooling with a window size of 2 and stride being 2 as well.

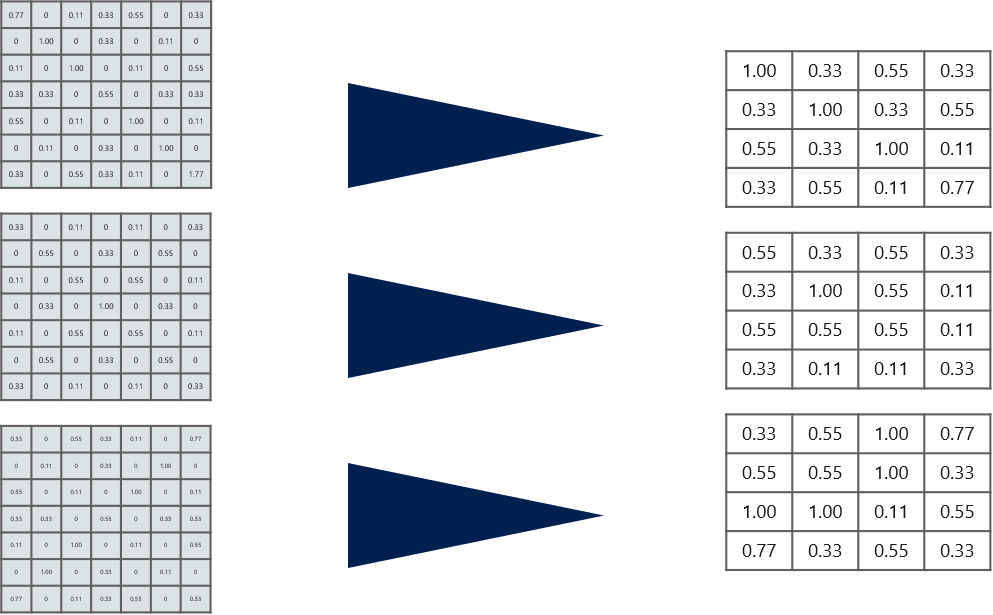


So in this case, we took **window size** to be **2** and we got **4** **values** to choose from. From those 4 values, the **maximum value** there is 1 so we pick 1. Also, note that we **started out** with a **7×7** matrix but now the same matrix after **pooling** came down to **4×4.**

But we need to **move** the **window across** the **entire** image. The procedure is exactly as same as above and we need to repeat that for the entire image.



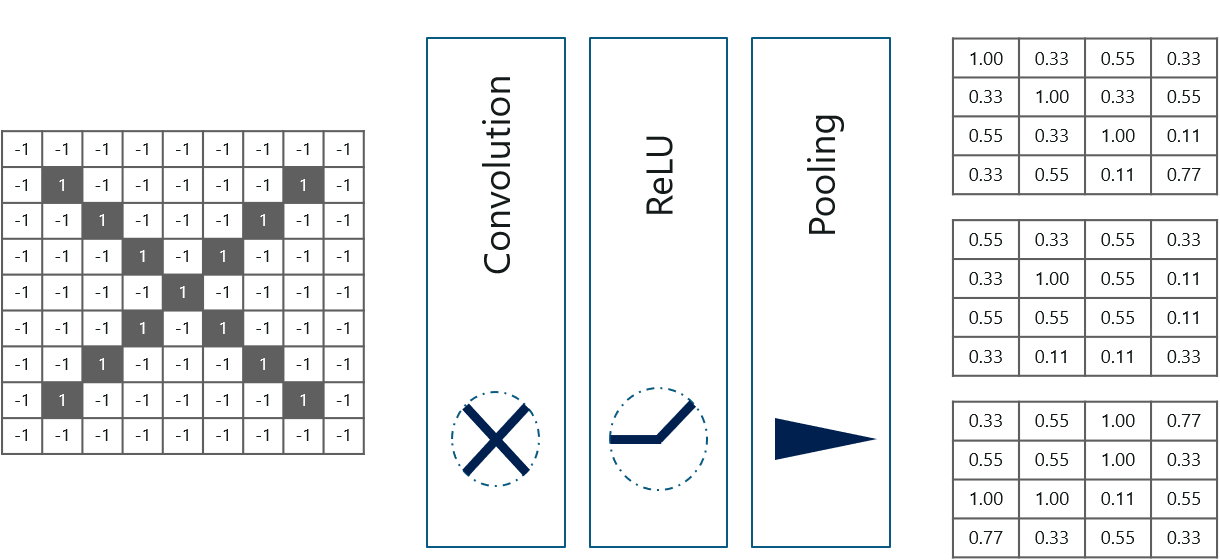
Do note that this is for **one filter.** We need to do it for 2 other filters as well. This is done and we arrive at the following result:



Well the **easy part** of this **process** is **over.** Next up, we need to **stack up all these layers!**

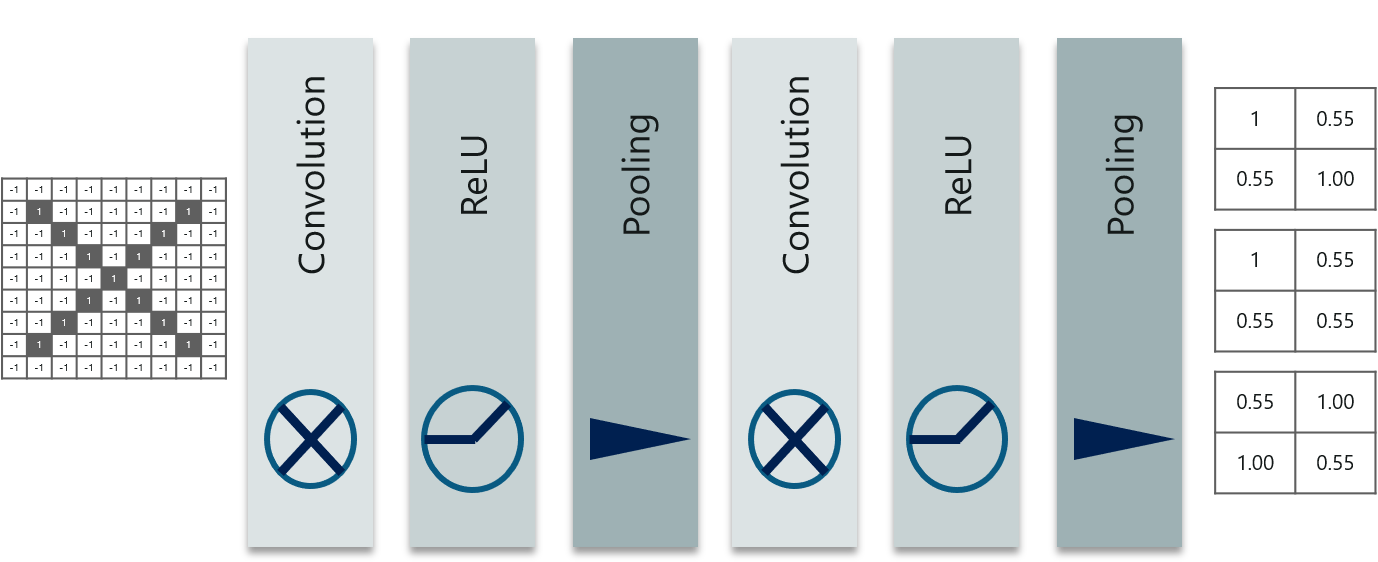
**Stacking Up The Layers**

So to get the **time-frame** in one picture we’re here with a **4×4** matrix from a **7×7** matrix after passing the input through 3 layers – **Convolution, ReLU** and **Pooling** as shown below:



But can we **further reduce** the image from **4×4** to **something lesser?**

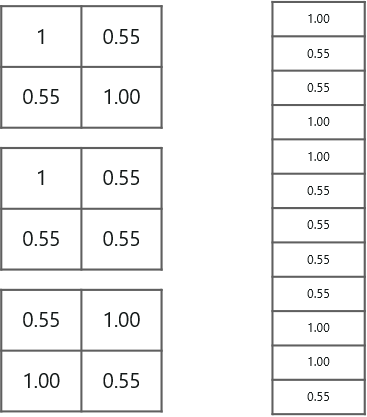
**Yes, we can!** We need to perform the 3 operations in an iteration after the first pass. So after the second pass we arrive at a 2×2 matrix as shown below:



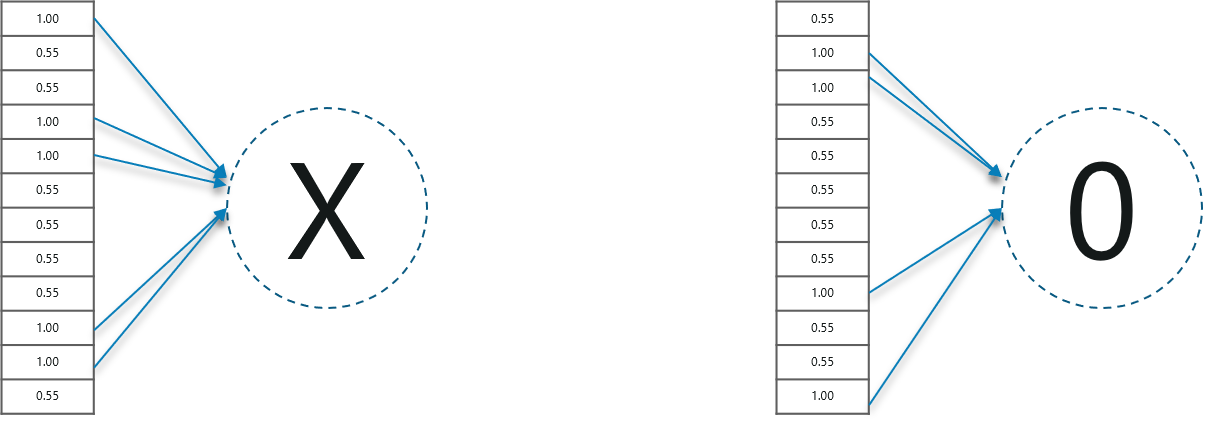
The last layers in the network are **fully connected,** meaning that neurons of preceding layers are **connected** to **every neuron** in **subsequent** layers.

This **mimics high level reasoning** where all possible **pathways** from the **input** to **output** are considered.

Also, fully connected layer is the final layer where the classification actually happens. Here we take our filtered and shrinked images and put them into one single list as shown below:



So **next,** when we feed in, **‘X’** and **‘O’** there will be **some element** in the vector that will be **high.** Consider the image below, as you can see for ‘X’ there are **different elements** that are **high** and **similarly,** for **‘O’** we have **different elements** that are **high:**



Well, what did we **understand** from the **above image?**

When the **1st, 4th, 5th, 10th** and **11th** values are **high,** we can classify the image as **‘x’.** The concept is similar for the other **alphabets** as well – when certain **values** are arranged the way they are, they can be **mapped** to an **actual** letter or a **number** which we **require**

## What is Dropout in Neural Networks? *The term “dropout” refers to dropping out units (both hidden and visible) in a neural network.*

Simply put, dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. By “ignoring”, I mean these units are not considered during a particular forward or backward pass.

More technically, At each training stage, individual nodes are either dropped out of the net with probability 1-p or kept with probability p, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed.

## Why do we need Dropout?

Given that we know a bit about dropout, a question arises — why do we need dropout at all? Why do we need to literally shut-down parts of a neural networks?

**The answer to these questions is “to prevent over-fitting”.**

A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data.

# Dropout — Revisited

Now that we know a little bit about dropout and the motivation, let’s go into some detail. If you just wanted an overview of dropout in neural networks, the above two sections would be sufficient. In this section, I will touch upon some more technicality.

In machine learning, regularization is way to prevent over-fitting. Regularization reduces over-fitting by adding a penalty to the loss function. By adding this penalty, the model is trained such that it does not learn interdependent set of features weights. Those of you who know Logistic Regression might be familiar with L1 (Laplacian) and L2 (Gaussian) penalties.

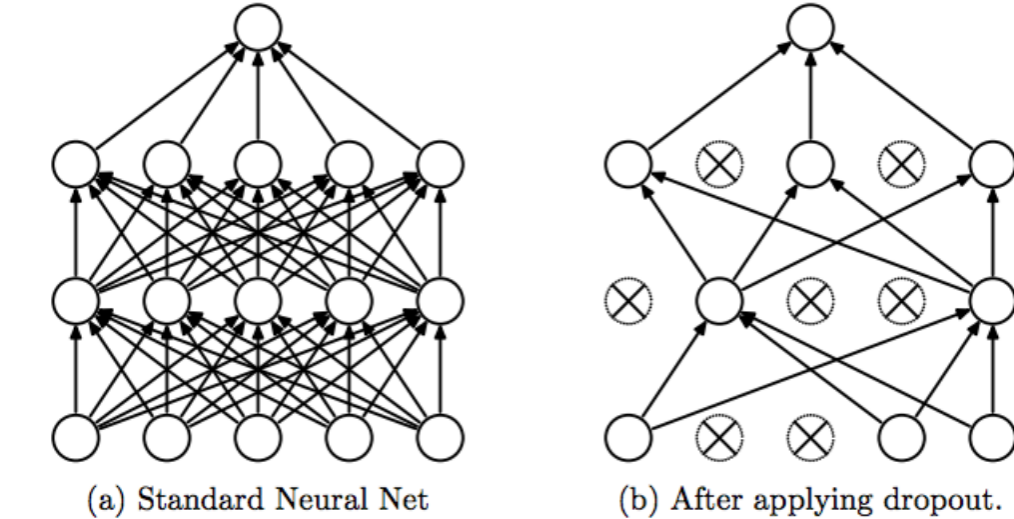
Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons.

## Training Phase:

Training Phase: For each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction, p, of nodes (and corresponding activations).

## **Testing Phase:**

Use all activations, but reduce them by a factor p (to account for the missing activations during training).



# Some Observations:

1. Dropout forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.
2. Dropout roughly doubles the number of iterations required to converge. However, training time for each epoch is less.
3. With H hidden units, each of which can be dropped, we have  
   2^H possible models. In testing phase, the entire network is considered and each activation is reduced by a factor p.

# Flatten Layers:

**Flatten** is used to flatten the input. For example, if flatten is applied to layer having input shape as **(batch\_size, 2,2)**, then the output shape of the layer will be **(batch\_size, 4)**

**dense layer:**

The **dense layer** is a neural network **layer** that is connected deeply, which means each neuron in the **dense layer** receives input from all neurons of its previous **layer**. The **dense layer** is found to be the most commonly used **layer** in the models. In the background, the **dense layer** performs a matrix-vector multiplication.

**OpenCV:**

**OpenCV** (*Open source computer vision*) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.

OpenCV supports some models from deep learning frameworks like TensorFlow, Torch, PyTorch (after converting to an ONNX model) and Caffe according to a defined list of supported layers.

**History**

Officially launched in 1999 the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls.The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

* Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.
* Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
* Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

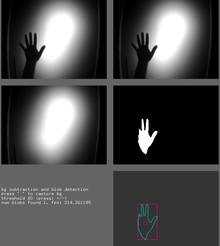
The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 "pre-release" was released in October 2008.

The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the C++ interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six months and development is now done by an independent Russian team supported by commercial corporations.

In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site.

On May 2016, Intel signed an agreement to acquire Itseez, a leading developer of OpenCV.

**Applications:**

[](https://en.wikipedia.org/wiki/File:OfxOpenCV.png)

openFrameworks running the OpenCV add-on example

OpenCV's application areas include:

* 2D and 3D feature toolkits
* Egomotion estimation
* Facial recognition system
* Gesture recognition
* Human–computer interaction (HCI)
* Mobile robotics
* Motion understanding
* Object identification
* Segmentation and recognition
* Stereopsis stereo vision: depth perception from 2 cameras
* Structure from motion (SFM)
* Motion tracking
* Augmented reality

To support some of the above areas, OpenCV includes a statistical machine learning library that contains:

* Boosting
* Decision tree learning
* Gradient boosting trees
* Expectation-maximization algorithm
* k-nearest neighbor algorithm
* Naive Bayes classifier
* Artificial neural networks
* Random forest
* Support vector machine (SVM)

**Conclusion**

The Drowsiness Detection System, which is based on the driver's eye closure, can discern between normal eye twitch and drowsiness, as well as detect drowsiness when driving. The suggested scheme will help deter injuries caused by drowsy driving. Using a Haar cascade classifier, OpenCV was used to detect faces and eyes, and then a CNN model to predict the status. Continuous eye closures are used to assess the driver's alertness level. For the future work, this detection system can be made into hardware with advanced features.

**REFERENCES**

[1] N. C. for Statistics and Analysis, “Crash Stats: Drowsy Driving 2015,” October 2017. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/ Public/ViewPublication/812446

[2] M. Walker, Why We Sleep: Unlocking the Power of Sleep and Dreams. Scribner, 2017. [Online]. Available: https://books.google.com.qa/books? id=8bSuDgAAQBAJ

[3] W. H. Organization, “The top 10 causes of death.” [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/ the-top-10-causes-of-death

[4] M. Benz, “Mercedes benz safety - s class.” [Online]. Available: https://www.mercedes-benz.co.uk/passengercars/mercedes-benz-cars/ models/s-class/saloon-w222/safety/intelligent-drive.module.html

[5] Muze, “Eyesight.” [Online]. Available: https://www.eyesight-tech.com/

[6] A. D. McDonald, J. D. Lee, C. Schwarz, and T. L. Brown, “A contextual and temporal algorithm for driver drowsiness detection,” Accident Analysis & Prevention, vol. 113, pp. 25–37, Apr. 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0001457518300058>

[7] U. S. F. M. C. S. A. T. Division, “PERCLOS: A Valid Psychophysiological Measure of Alertness As Assessed by Psychomotor Vigilance,” October 1998.

[8] C. S. Wei, Y. T. Wang, C. T. Lin, and T. P. Jung, “Toward Drowsiness Detection Using Non-hair-Bearing EEG-Based Brain-Computer Interfaces,” IEEE Transactions on Neural Systems and Rehabilitation Engineering, 2018.

[9] V. J. Kartsch, S. Benatti, P. D. Schiavone, D. Rossi, and L. Benini, “A sensor fusion approach for drowsiness detection in wearable ultra-low-power systems,” Information Fusion, vol. 43, pp. 66–76, Sep. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/ pii/S1566253517306942

[10] S. Tateno, X. Guan, R. Cao, and Z. Qu, “Development of Drowsiness Detection System Based on Respiration Changes Using Heart Rate Monitoring,” in 2018 57th Annual Conference of the Society of Instrument and Control Engineers of Japan, SICE 2018, 2018, pp. 1664–1669.

[11] A. Kamilaris and F. X. Prenafeta-Bold, “Deep learning in agriculture: A survey,” Computers and Electronics in Agriculture, vol. 147, pp. 70 – 90, 2018. [Online]. Available: http://www.sciencedirect.com/science/ article/pii/S0168169917308803

[12] M. Tayab Khan, H. Anwar, F. Ullah, A. Ur Rehman, R. Ullah, A. Iqbal, B.-H. Lee, and K. S. Kwak, “Smart Real-Time Video Surveillance Platform for Drowsiness Detection Based on Eyelid Closure,” Wireless Communications and Mobile Computing, vol. 2019, pp. 1–9, 2019.

[13] M. F. Shakeel, N. A. Bajwa, A. M. Anwaar, A. Sohail, A. Khan, and H. ur Rashid, “Detecting Driver Drowsiness in Real Time Through Deep Learning Based Object Detection,” 2019, pp. 283–296. [Online]. Available: http://link.springer.com/10.1007/978-3-030-20521-8{ }24

[14] L. Celona, L. Mammana, S. Bianco, and R. Schettini, “A multi-task CNN framework for driver face monitoring,” IEEE International Conference on Consumer Electronics - Berlin, ICCE-Berlin, vol. 2018-Septe, pp. 1–4, 2018.

[15] C.-H. Weng, Y.-H. Lai, and S.-H. Lai, “Driver drowsiness detection via a hierarchical temporal deep belief network,” in Asian Conference on Computer Vision. Springer, 2016, pp. 117–133.

[16] Y. Xie, K. Chen, and Y. L. Murphey, “Real-time and Robust Driver Yawning Detection with Deep Neural Networks,” Proceedings of the 2018 IEEE Symposium Series on Computational Intelligence, SSCI 2018, pp. 532–538, 2019.

[17] S. Mehta, S. Dadhich, S. Gumber, and A. Jadhav Bhatt, “Real-Time Driver Drowsiness Detection System Using Eye Aspect Ratio and Eye Closure Ratio,” SSRN Electronic Journal, pp. 1333–1339, 2019.

[18] P. Peyrard, “Personal system for the detection of a risky situation and alert,” Feb. 28 2019, uS Patent App. 16/178,365.

[19] R. Jabbar, K. Al-Khalifa, M. Kharbeche, W. Alhajyaseen, M. Jafari, and S. Jiang, “Real-time driver drowsiness detection for android application using deep neural networks techniques,” Procedia computer science, vol. 130, pp. 400–407, 2018.

[20] R. Jabbar, K. Al-Khalifa, M. Kharbeche, W. Alhajyasen, M. Jafari, and S. Jiang, “Applied internet of things iot: Car monitoring system for modeling of road safety and traffic system in the state of qatar,” Qatar Foundation Annual Research Conference Proceedings, vol. 3, p. ICTPP1072, 2018.