CV3: Object Detection, Optical Flow, Feature Tracking, Convolutional Neural Networks

StartOnAI
Felix Liu, Keshav Shah, Navein Suresh

19 February, 2021

Contents

1 Object Detection

1.1 What is Object Detection

Object detection is an AI technique in which computers can locate and identify images that are within an image or a video in order to perform data analysis on these images. Object detection also allows for classification of multiple types of objects within a certain image.

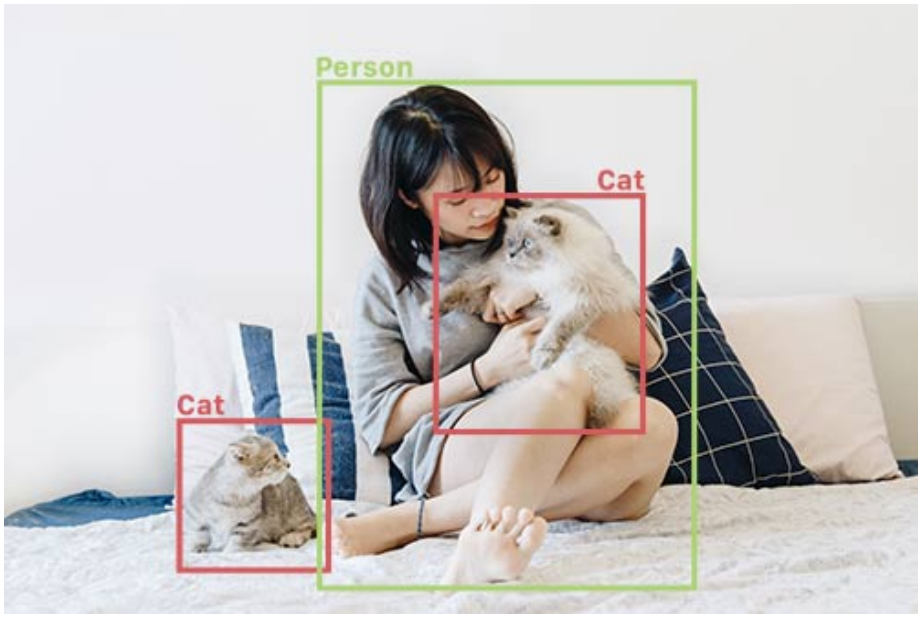


Figure 1: This image displays how an AI algorithm is able to identify two different objects within the frame (source: [6])

1.1.1 Object Detection vs Image Detection

Object detection is widely confused with image recognition as both seem to be similar concepts due to the tasks that they can accomplish. Image recognition tends to assign labels to an image such as if an image of a dog is viewed, that image will be labeled as "dog." On the other hand, object detection would draw a box around the dog and identify that the object does indeed exist in the image, and the model will also later pick which label should be applied to the image. It is important to make this distinction because it shows how object detection windows actually can provide more information than do image detection algorithms.

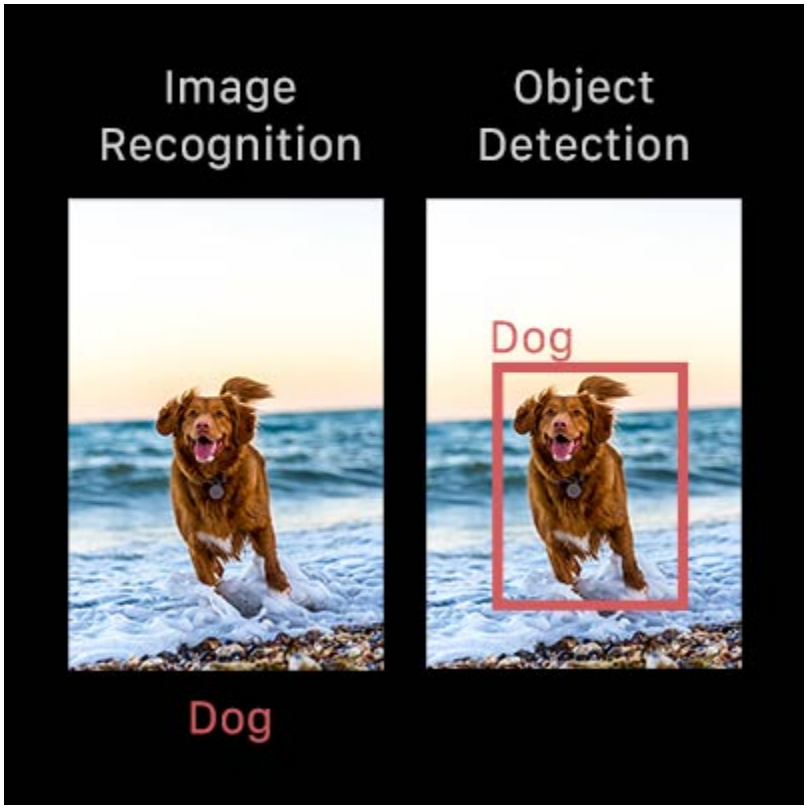


Figure 2: Illustration of how image detection algorithms work (left), compared to object detecting models (right) (source: [6])

1.1.2 Types of Object Detection

As you may have guessed, there are a variety of ways in which we can do object detection. Some commonly used techniques are looking at color histograms or edges of an image in order to identify certain pixel groups that may be part of an image, but there also are other techniques such as regression models that can predict where the object might be in the image along with what that object should be labelled.

These are machine learning based algorithms, but can we incorporate deep learning? To put it short, absolutely. A special type of neural network called Convolutional Neural Networks are the most effective way to go about object detection. We will discuss CNN's later on in this tutorial.

1.1.3 Why is Object Detection Important and What are Some Applications?

Object detection is significant to us as it provides a computer the ability to track objects within an image in a similar manner as the way a human perceives it. This can allow the image to track and even count the objects that are appearing within a window, and this can lead to many important

operations being done by computers that previously were only possible through the work of humans.

Some everyday applications that use object detection are:

- Crowd Counting
- Self-Driving Cars
- Video Surveillance Systems
- Face Detection
- Anomaly Detection

1.2 Theory of Object Detection

1.2.1 How Does Object Detection Work?

Now we will take a look at the state-of-the-art machine learning techniques that are used in the real world of object detection.

There are usually 2 parts to object detection models: an encoder and a decoder

Encoder: encoders can take an image in as inputs and it can run it through layers and blocks that basically extract essential information from the image in order to classify and label objects that are identified.

Decoder: A decoder, also known as a regressor in many instances, is connected to what the output of the input is and it predicts the locations and the sizes of the bounding box of the encoder. After this task is completed the machine can deliver an output of the model which is a set of x, y coordinates. This model has little complexity to it compared to other machine learning models, but there are some intricacies that make it a little impractical in some scenarios. For one, you must always be specifying the number of boxes (bounding boxes) beforehand. For example, if you have an image that has a total of two dogs but you told the model to only have one bounding box, the model will only detect one of the dogs. The only upside to this model is if you definitely know the number of object you need to predict in an image, this pure-regressor model is a very good option.

1.2.2 Extensions of the Regressor Approach

Expanding on the model that we have just discussed is a model called the **region proposal network** (RPN). This is a type of decoder where the model can provide regions of an image where it believes there might be an existing object. The pixels in these regions would then be fed into a classification network to determine labels if the model made accurate estimations. This method becomes beneficial as it ends up providing a more accurate model that is flexible with a number of regions that might contain an image or bounding box. Of course with every benefit, there is some sort of trade off. In this model, the computational efficiency is greatly hurt in exchange for such high accuracy.

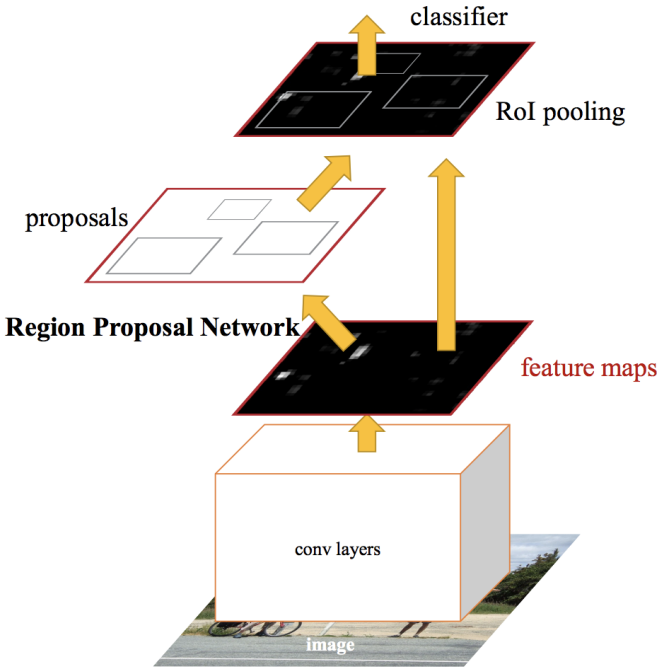


Figure 3: Region proposal networks being incorporated in an object detection system (source: [1])

1.2.3 Fixing the Issue of Computational Efficiency in RPN's

Single shot detectors or SSD's are a middle ground to region proposal networks. Instead of using subnetworks to estimate regions that might contain images, SSD's actually run on sets of predetermined regions. Anchor points in the form of a grid are laid over the input image and each anchor point contains boxes of multiple shapes and sizes which are the regions of the image. For each box in the anchor, the model will output a prediction of whether or not an object exists in that region. Modifications to that box's location and size are made to make it fit the object more closely. Due to multiple boxes being present at each anchor point, SSDs can produce detectors that overlap and for this reason, post-processing must be applied to the SSD model into order to prune many predictions that would overlap so that the best one can be chosen. A common post-processing technique used to select only the best prediction is called **non-maximum suppression**.

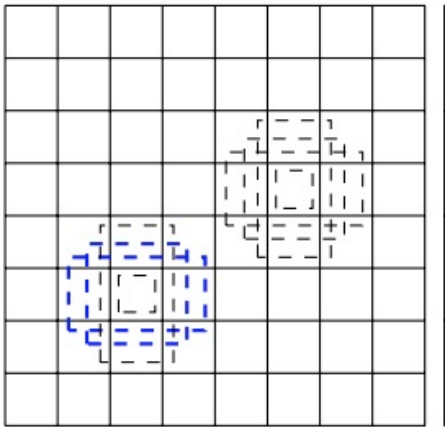


Figure 4: The anchor points and regions of a Single shot detector (SSD) (source: [6])

1.2.4 Accuracy of an Object Detecting Model

Object detectors output locations and labels for objects but there needs to be a way in which the performance of the model can be checked. To check the efficiency of effectively predicting the object's location, a commonly used metric is called **intersection-over-union** (IOU). This method says that given two bounding boxes, you can compute the area of the intersecting region and divide by the area of the union, or total area. This value will range from 0 to 1 where closer to 0 represents no interactions, and a value closer to 1 represents perfectly overlapping regions. For labels, we can use "percent correct" methods to determine the overall accuracy of our model.

1.2.5 Using a Combination of Object Detection Tools

Say you have a problem where you want to detect humans as they enter a store and determine the likelihood of them interacting with a specific fixture as you move it from place to place. You can apply three separate tools here: object detection, object tracking, and object counting.

Object Detection: Object detection: Writing and training a human interaction model as described above to detect the fixture and humans to help differentiate customers from associates. Human interaction models allow economists in wholesale industries and other fields to monitor sales of products based off of customer interactions.

Object Tracking: Use a tracking function to follow unique customers around the store to record details of the interactions they had with the display

Object Counting: Counting individual objects in a class/frame, such as the number of people or number of a specific product

Using a combination of these techniques would allow a retailer to analyze how effective store displays truly are by counting the number of customers who interact with the display and how long they spend looking at it. They would also be able to tell how many of the customers who looked at the display would actually put that product in their cart, which goes to show the grand spectra of how object detection can be used across many disciplines.

1.3 Machine Learning Pipeline for Object Detection

We have now discovered many attributes of object detection, so now it is time to summarize how the machine learning algorithm completes object detection by learning about the pipeline in which this information is relayed.

You first start with a collection of images and find the relevant features of each image in the preprocessing stage. For example, you might use a feature extraction algorithm to take edge and corner features to differentiate between classes of certain images.

These captured features can now be added to a machine learning model which will separate these features into categories in order to analyze them and classify new objects in the future.

Using feature extraction and machine learning algorithms, we can use combinations of the acquired data in order to create accurate object recognition models.

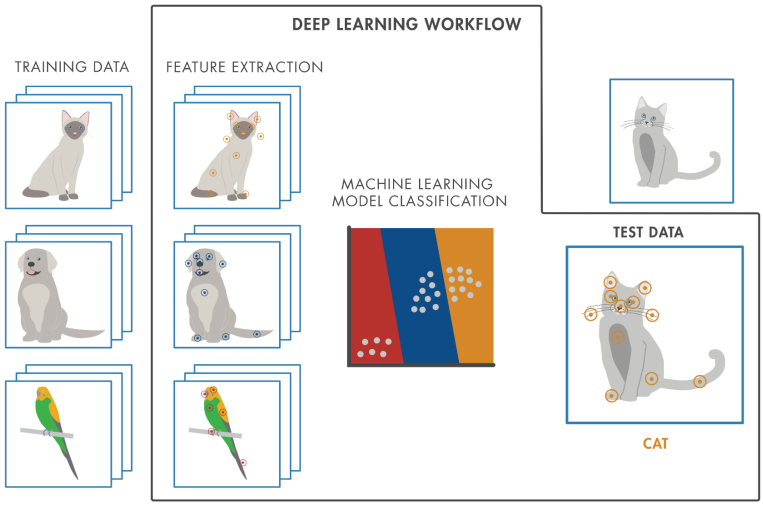


Figure 5: Machine Learning pipeline of object detection (source: [8])

In summary, using machine learning for this object detection process allows for the best combination of features and classifiers for learning to be used. It can calculate accurate and efficient results with the use of a very minimal quantity of data.

2 Optical Flow

2.1 Introduction

Optical flow refers to how various objects or visual features (like edges or surfaces) move within a video. These movements are represented by vectors (called **optical flow vectors**) that represent the direction and speed of movement (through length or color), and when these vectors are drawn throughout an image (more specifically, a frame in the video), we get a vector field known as the **optical flow field**. An example of this is shown below in Fig 6.

Figure 6: The red lines are optical flow vectors that form the optical flow field (source: [3])

2.1.1 Applications

Optical flow has many applications and the following are just a few:

- Image stabilization
- Traffic analysis
- Recognizing movements

Image Stabilization: works by first analyzing the movement of background pixels, then canceling out that movement (determined by the optical flow vectors) to make a video seem less wobbly.

Traffic Analysis: can be used to analyze traffic by determining regions of heavy traffic and the direction that traffic is moving.

Recognizing movements: optical flow can be used in conjunction with machine learning methods to determine which actions are being performed (ex. gaming consoles like the Xbox can recognize a jump, duck, etc.).

2.1.2 Intuition

When we are trying to find the optical flow field of a field, we must first lay out a condition known as the **brightness constancy constraint**. To understand what this states, we begin by examining the brightness of an arbitrary pixel a at position (x, y) at time t that has moved to position $(x + dx, y + dy)$ at time $t + dt$. Clearly, if we wish to even track the movement of pixel a between these two times (in a video with a finite number of frames, these two times would refer to consecutive frames), pixel a must have a constant brightness in the short run. Otherwise, pixel a would be impossible to track.

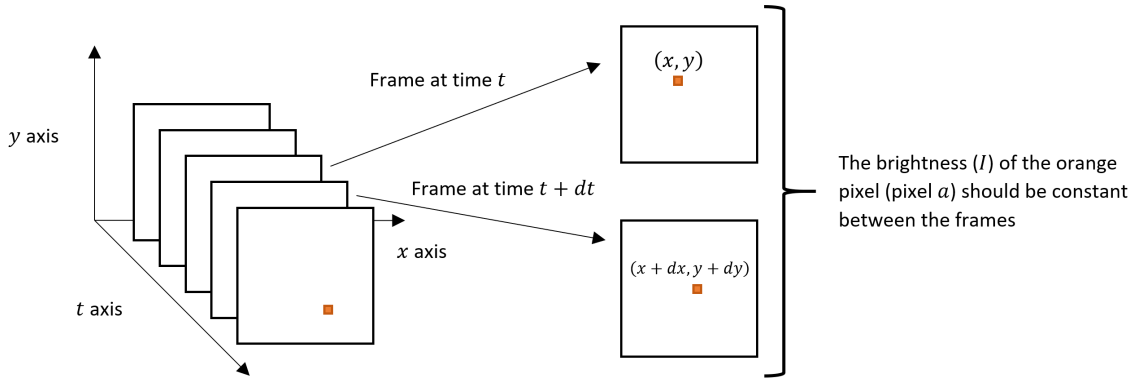


Figure 7: A visual representation of what the brightness constancy constraint states (source: StartOnAI)

Mathematically, we can express the brightness constancy constraint with the following equation (Eq. 1), where $I(x, y, t)$ is a function denoting the brightness of a pixel located at position (x, y) at time t:

$$I(x, y, t) \approx I(x + dx, y + dy, d + dt) \tag{1}$$

Notice the use of the $\approx$ symbol, since it is still okay for the brightness of a pixel to change over a long period of time; it just has to be constant in the short run. However, it is also okay to just outright use the $=$ as many other texts do.

Another important assumption for optical flow is one of **small movements**. This means that pixels don't jump halfway across the image between consecutive frames, which would make find optical flow vectors a headache.

2.2 Optical Flow Equation

2.2.1 Statement

Recall that the optical flow vector marked the velocity with which a pixel moved. Using knowledge from linear algebra, we can express this velocity as a vector with x and y components which we

will denote as u and v respectively. Therefore, the optical flow vector becomes $\begin{bmatrix} u \\ v \end{bmatrix}$.

Since optical flow is highly geared around math, we will present the **optical flow equation** first (Eq. 2), then provide a proper proof for it.

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0 \tag{2}$$

The first step in analyzing this equation is to notice that we already know I and its partial derivatives. This is because I is merely the brightness of a pixel in the video, something we can determine by tracking each pixel within the video and taking measurements. This leaves u and v as our unknown variables.

However, as we know from basic algebra, solving an equation with two unknowns isn't possible since there are infinitely many solutions. For instance, the line $ax + by = c$ is one such equation with two unknowns x and y, and constants a, b, and c. This is the basis of what is known as the **aperture problem**, which we will cover in the next section and figure out how to work around.

2.2.2 Proof

Proving the optical flow equation requires working from the brightness constancy constraint, then applying the Taylor series expansion (1st order approximation suffices) and simplifying from there. The main insight for this step is that, if the time step between frames is extremely small, the brightness function can be linearized.

As a foreword, recall how u and v were defined as the x and y components of velocity respectively. This means that $u = \frac{dx}{dt}$ and $v = \frac{dy}{dt}$ by the definition of velocity.

$$\begin{aligned} I(x,y,t) &= I(x+dx, y+dy, d+dt) && \text{brightness constancy constraint} \\ &= I(x,y,t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt && \text{Taylor Series Expansion} \end{aligned}$$

$$\begin{aligned} 0 &= \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt && \text{subtract } I(x,y,t) \text{ from both sides} \\ &= \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} && \text{divide } dt \text{ from both sides} \\ &= \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} && \text{substitute definitions of } u \text{ and } v \end{aligned}$$

This concludes the proof.

2.3 The Aperture Problem

2.3.1 Intuition

Let us begin with the following question (look at Fig 8 for reference): Determine where the yellow pixel ends up after the line moves from the initial position (marked in green) to the final position (marked in red) while looking through an aperture (small hole).

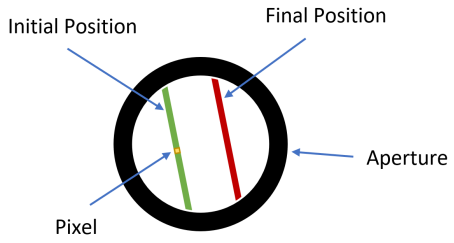


Figure 8: Problem asks where the yellow pixel ends up on the red line (source: StartOnAI)

Answer: It's ambiguous! Using our intuition, we see that there is no way to determine how far the pixel has traveled parallel to the line (visualized in Fig 9). This is the essence of the aperture problem.

(a) Line shifts down a little

(b) Line shifts up a little

Figure 9: Shifting of the line makes the final position of the pixel ambiguous (source: StartOnAI)

In order to formalize this concept a little more, we state that the aperture problem **prevents us from determining optical flow perpendicular to the image gradient** (image gradient can refer to the gradient vectors of something like an edge). As a consequence of this problem, we are forced to settle for **normal flow**, which is the optical flow parallel to the image gradient (refer to Fig 10).

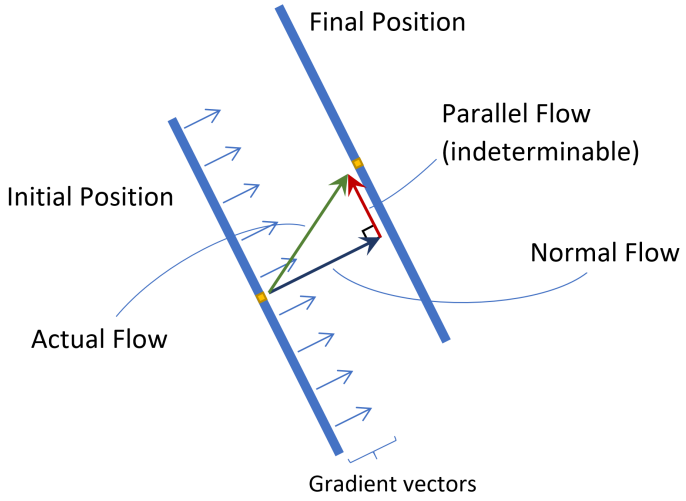


Figure 10: only the normal flow can be computed (source: StartOnAI)

2.3.2 "Solution"

Since the optical flow vector is ambiguous, our first instinct is to try and add additional constraints such that we eliminate ambiguity.

In fact, the various ways of adding these constraints is what makes each method of find optical flow unique from one another.

One such method of adding additional constraints is to apply the optical flow equation to a small $k \times k$ window around a given pixel. Solving this system would then give you the optical flow vector for that pixel. This particular method of determining a local constraint around a pixel is known as the **Lucas-Kanade Method**, which is covered in detail in the next section.

2.4 Lucas-Kanade Method

Once again, most of these methods are highly mathematical, so **the following derivation is presented for completeness and is completely optional**. If you are not interested in the derivation, feel free to skip ahead to Sec 2.4.2.

With that out of the way, let us explore the mathematical concept needed for this derivation: **the Normal Equation**

2.4.1 The Normal Equation

For those disinclined to read through a proof, I will first present the equation and state its meaning, then provide a proof later. For those unfamiliar with notation, adding a ^{T} to a matrix or vector flips the matrix/vector along its main diagonal. For instance, $\begin{bmatrix} a \\ b \end{bmatrix}^T = \begin{bmatrix} a & b \end{bmatrix}$, and $\begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}^T = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$.

$$A^T\vec{b} = A^T A\vec{x} \tag{3}$$

Eq 3 is what is known as the **normal equation** in linear algebra and is used to approximate solutions to the equation $A\vec{x} = \vec{b}$. This situation arises when $\vec{b}$ lies outside the column space of matrix A, denoted $C(A)$, (which is basically the set of all vectors $A\vec{x}$ can possibly produce for an arbitrary $\vec{x}$).

The best way to think about this is to visualize $C(A)$ as some arbitrary hyperplane in space and $\vec{b}$ as some vector not on this plane. Clearly, there is no way for $A\vec{x}$ to equal $\vec{b}$, since the definition of the column space states that $A\vec{x}$ can only produce vectors that lie on hyperplane $C(A)$. Therefore, we settle for an approximation where the best solution for $\vec{x}$ would be the one such that $A\vec{x}$ lies "under" $\vec{b}$ (visualized in Fig 11).

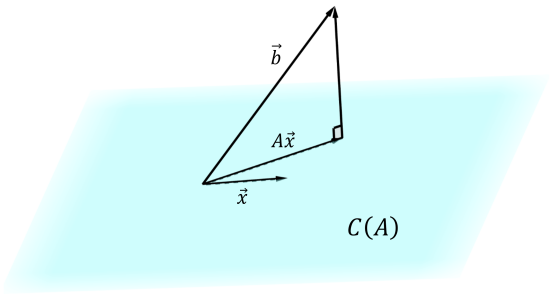


Figure 11: $A\vec{x}$ lies "under" $\vec{b}$ (source: StartOnAI)

Thus, the key takeaway for this equation is that it gives us conditions to approximate the best possible solution to the equation $A\vec{x} = \vec{b}$ when there are no actual solutions.

Proof: First, we begin by labeling a diagram (Fig 12).

We then proceed by listing what we know:

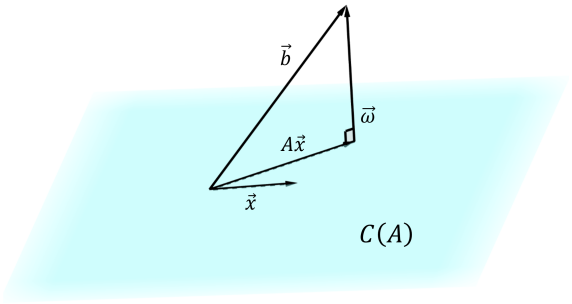


Figure 12: Problem setup (source: StartOnAI)

1. $C(A)$ and $\vec{\omega}$ are orthogonal (perpendicular), meaning all basis vectors of A are orthogonal to $\vec{\omega}$ as well
2. $\vec{b}$ can be obtained from $A\vec{x}$ and $\vec{\omega}$ by vector addition

Formalizing, we get the following equations:

$$A^T\vec{\omega} = 0 \tag{4}$$

$$A\vec{x} + \vec{\omega} = \vec{b} \tag{5}$$

From here, we rearrange Eq 5 into $\vec{\omega} = \vec{b} - A\vec{x}$ and substitute into Eq 4. Expanding from here gets the normal equation.

$$\begin{aligned} 0 &= A^T\vec{\omega} && \text{from Eq 4} \\ &= A^T(\vec{b} - A\vec{x}) && \text{substitution} \\ &= A^T\vec{b} - A^TA\vec{x} && \text{expand} \\ &\downarrow \\ A^T\vec{b} &= A^TA\vec{x} && \text{rearrange} \end{aligned} \tag{6}$$

This concludes the proof for the normal equation.

2.4.2 The Method

As discussed earlier, the aperture problem presented a situation where we need to introduce additional constraints in order to find the optical flow vector. In the case of the Lucas-Kanade Method, **constraints are added by applying the optical flow equation to a $k \times k$ window around pixel a under the assumption that each pixel in this window moves in the same way as pixel a.** Solving from there gets the optical flow vector.

With the proper intuition at hand, we must now proceed by formalizing this constraint with math.

Since we are applying the optical flow equation to a $k \times k$ window of pixels, there will be a total of k^2 equations, since there are k^2 pixels and one equation for each pixel. Moreover, since we assume that each pixel moves in approximately the same way as the center pixel (pixel a), all pixels in this window will have the same optical flow vector. This system of k^2 equations is shown below, where the k^2 pixels are labeled $p_1, p_2, p_3, \ldots, p_{k^2}$ (note: we use the subscript notation to denote a derivative to reduce clutter; also, $I(p_1)$ denotes the brightness of pixel p_1):

$$\begin{aligned} I_x(p_1)u + I_y(p_1)v &= -I_t(p_1) \\ I_x(p_2)u + I_y(p_2)v &= -I_t(p_2) \\ I_x(p_3)u + I_y(p_3)v &= -I_t(p_3) \\ &\vdots \\ I_x(p_{k^2})u + I_y(p_{k^2})v &= -I_t(p_{k^2}) \end{aligned}$$

Those familiar with elementary linear algebra will know that we can express such a system with the matrix multiplication $A\vec{x} = \vec{b}$ as follows:

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ I_x(p_3) & I_y(p_3) \\ \vdots & \vdots \\ I_x(p_{k^2}) & I_y(p_{k^2}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -I_t(p_1) \\ -I_t(p_2) \\ -I_t(p_3) \\ \vdots \\ -I_t(p_{k^2}) \end{bmatrix} \tag{7}$$

Since there are way more equations that unknowns (unknowns are u and v, and there are k^2 equations), this is a suitable situation to apply the normal equation, as it almost guaranteed no solution exists. As a refresher, the normal equation states that the best approximation for $\vec{x}$ in the equation $A\vec{x} = \vec{b}$ satisfies $A^T\vec{b} = A^TA\vec{x}$. We can rewrite this in terms of $\vec{x}$ as $\vec{x} = (A^TA)^{-1}A^T\vec{b}$.

Then, applying the normal equation to solve for u and v results in the following:

$$\begin{aligned}
\begin{bmatrix} u \\ v \end{bmatrix} &= \left(\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ I_x(p_3) & I_y(p_3) \\ \vdots & \vdots \\ I_x(p_{k^2}) & I_y(p_{k^2}) \end{bmatrix}^T \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ I_x(p_3) & I_y(p_3) \\ \vdots & \vdots \\ I_x(p_{k^2}) & I_y(p_{k^2}) \end{bmatrix} \right)^{-1} \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ I_x(p_3) & I_y(p_3) \\ \vdots & \vdots \\ I_x(p_{k^2}) & I_y(p_{k^2}) \end{bmatrix}^T \begin{bmatrix} -I_t(p_1) \\ -I_t(p_2) \\ -I_t(p_3) \\ \vdots \\ -I_t(p_{k^2}) \end{bmatrix} \\
&= \left(\begin{bmatrix} \sum_{i=1}^{k^2} \big(I_x(p_i)\big)^2 & \sum_{i=1}^{k^2} \big(I_x(p_i) I_y(p_i)\big) \\ \sum_{i=1}^{k^2} \big(I_x(p_i) I_y(p_i)\big) & \sum_{i=1}^{k^2} \big(I_x(p_i)\big)^2 \end{bmatrix} \right)^{-1} \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ I_x(p_3) & I_y(p_3) \\ \vdots & \vdots \\ I_x(p_{k^2}) & I_y(p_{k^2}) \end{bmatrix}^T \begin{bmatrix} -I_t(p_1) \\ -I_t(p_2) \\ -I_t(p_3) \\ \vdots \\ -I_t(p_{k^2}) \end{bmatrix} \\
&= \left(\begin{bmatrix} \sum_{i=1}^{k^2} \big(I_x(p_i)\big)^2 & \sum_{i=1}^{k^2} \big(I_x(p_i) I_y(p_i)\big) \\ \sum_{i=1}^{k^2} \big(I_x(p_i) I_y(p_i)\big) & \sum_{i=1}^{k^2} \big(I_x(p_i)\big)^2 \end{bmatrix} \right)^{-1} \begin{bmatrix} -\sum_{i=1}^{k^2} \big(I_x(p_i) I_t(p_i)\big) \\ -\sum_{i=1}^{k^2} \big(I_y(p_i) I_t(p_i)\big) \end{bmatrix}
\end{aligned}$$

From here, we can let computers, which are well equipped for matrix operations, continue the computation and return values for u and v that allow us to construct the optical flow vector. Applying this same process to other pixels will then allow us to construct the optical flow field.

2.5 Horn-Schunck Method

Since this method is even more mathematical in nature, only a brief introduction will be given.

This method introduces a constraint known as **smoothness**. Essentially, it tries to minimize distortions in the optical flow field and prefers solutions with smoother flow.

In order to describe "smoothness," we formulate it as a kind of global energy, where greater energy means less smooth and less energy means more smooth. This is known as a **functional**, which maps a space X (X is an image in this case) to the real numbers. Contextually, the energy functional E takes some image as input and spits out some real number describing the energy of the image.

Thus, if we wish to find the best (smoothest) optical flow field, we must minimize the energy functional E, which is given by the following, where α is some regularization constant determined by the programmer:

$$E = \iint \Big[(I_x u + I_y v + I_t)^2 + \alpha(||\nabla u||^2 + ||\nabla v^2||) \Big] dx\, dy \tag{8}$$

As a side note, larger values of α lead to a smoother flow. Furthermore, u and v are treated as functions of both x and y, so taking the gradient (denoted ∇) is an appropriate operation.

3 Feature Tracking

Feature Tracking, also known as object tracking, is a discipline within computer vision which aims to track objects, items, people, etc. within frames of vision. Several different algorithms are implemented in order to perform such a task at varying different efficiencies and accuracies. This section will explore the fundamentals, theory, and applications of this growing field.

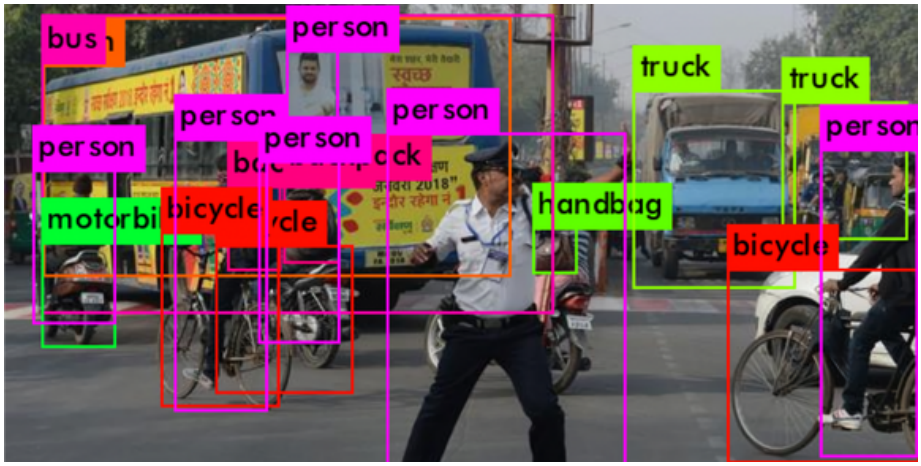


Figure 13: A machine learning model identifying various different objects on a busy street

3.1 What Exactly is Feature Tracking

As AI explorers, we must really understand what feature tracking aims to accomplish. Imagine yourself in a crowded traffic jam. As a human, you are able to identify, with your own eyes, various movements, objects, animals, clusters, etc. But for a machine, this task is difficult to do and near impossible without the aid of algorithms, which draw from deep learning principles such as SORT or GOTURN. Before getting a grasp of object tracking, it's important to analyze what object detection does in a particular frame.

When an object is detected by the machine, a box, as well as an ID, is associated with that particular object. After this is finished, the process moves forward into feature tracking. At this point in the process, there are two potential paths to be taken, **offline object tracking** vs **online object tracking**.

Offline Object Tracking: The feature tracking is accomplished on a previously recorded video. In this case, all of the frames are existing and can easily be worked with.

Online Object Tracking: This type of tracking deals with live footage in which the frames are not pre-recorded and instead produced live. Future frames are not available for use and so algorithms are restricted in their capacity.

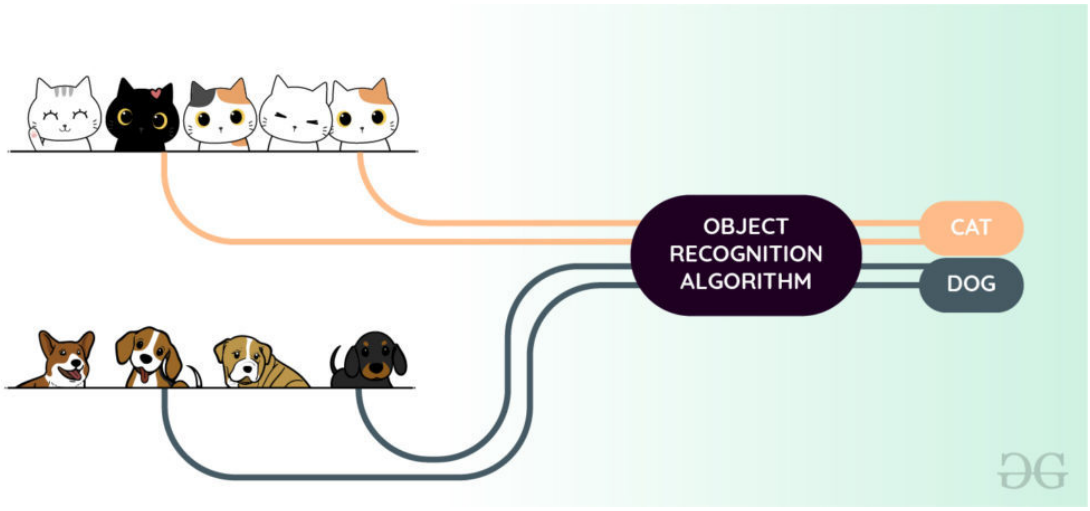


Figure 14: (source: [7])

Although there is a good portion of overlap between object detection and object tracking it is important to note the differences between the two. The following is a list of possible challenges one could encounter in the field of object tracking which would rather not be pertinent in the field of object detection: **Re-Identification**, **Appearance/disappearance**, **Occlusion**, **Motion Blur**, **Scale Change**, etc.

Re-Identification: This is a challenge related to whether the machine is able to re-identify an object from a previous frame.

Appearance/Disappearance: This deals with objects moving in and out of frames and re-identification of these objects.

Occlusion: At times, some objects are clouded or blocked by other objects and can cause machine confusion in that case.

Motion Blur: While objects are in motion, their identification by machines may be blurred as is usual with any motion identification.

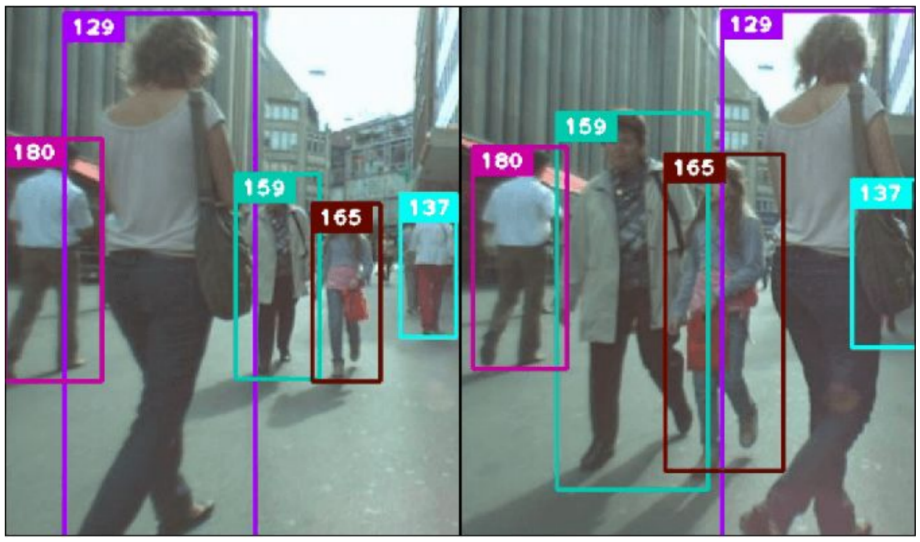
Scale Change: With effects of zooming in and out in play, the scale of perspective also changes, which affects the tracking of objects through the frame.

3.2 Major Object Tracking Algorithms (SORT, GOTURN, MDNet)

All of the following algorithms that we explore will have direct relations to several deep learning techniques.

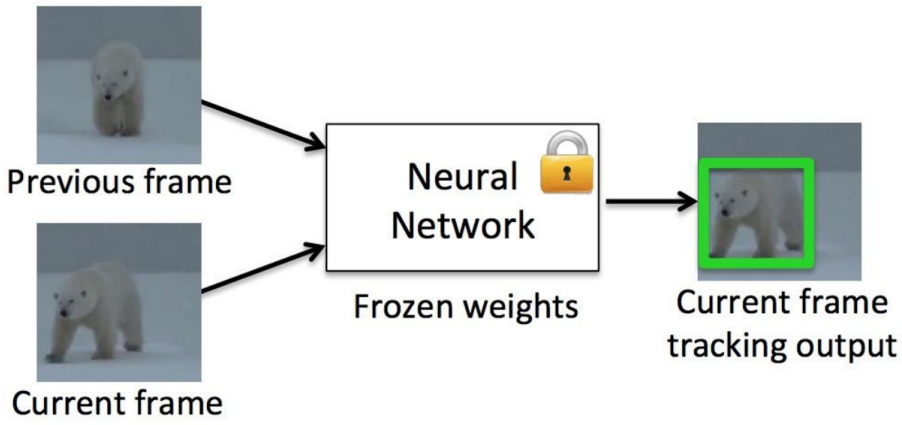
3.2.1 SORT

This particular algorithm is able to recognize and generate boxes for multiple different objects in a single frame.



3.2.2 GOTURN

With the GOTURN algorithm, the main process is combining two different frames of view, each capturing different pieces of information, and then combining them with a system of neural network weights to correctly track the object along its motion.



3.2.3 MDNet

This algorithm is part of the CNN class in which the training phase is aimed to speed up to produce faster results with great efficiency.

3.3 Applications

The most common and widely known application is self-driving cars. Object tracking is vital when these types of cars steer on their own and have to pay attention to the other cars near, in front, or behind them. CCTV, as well as surveillance cameras, are also another place in which object tracking algorithms can be implemented for increased efficiency and accuracy for facial recognition purposes.

4 Convolutional Neural Networks

4.1 Overview

4.1.1 Introduction to CNN's

Convolutional neural networks are particular algorithms that are allowing for advancements in Media Recreation, NLP, Image Analysis, and Classification techniques.

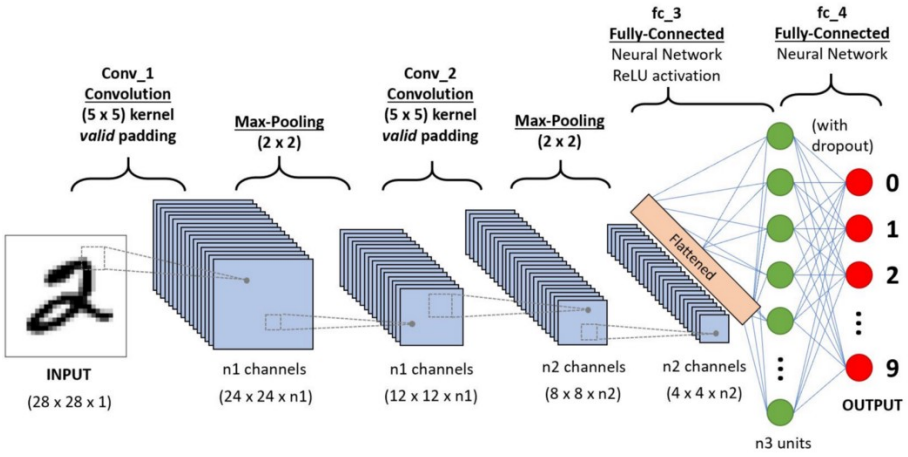


Figure 15: A look at the pathways of convolutional neural nets (source: [12])

CNNs are Deep Learning algorithms that can take images as an input, assign importance to them based on weights and biases, and then differentiate images from one another based on this process. There is much less preprocessing required in CNNs than compared to general classification algorithms.

CNNs take on the pathways of the human brain in terms of their connectivity as they are inspired by the organization of the visual cortex in the human brain. Neurons respond to stimuli from images in the environment in only a specific region of the visual field known as the receptive field. A collection of these fields can overlap in order to cover the entire visual area.

4.1.2 Why Use CNNs over Feed-Forward neural Networks

Since images are just pixel values, it does not make sense to flatten a 3×3 image matrix into a 9×1 vector. In basic binary imaging, this method might show a reasonably high precision score, but the actual prediction of image classes would be extremely inaccurate for more complex images.

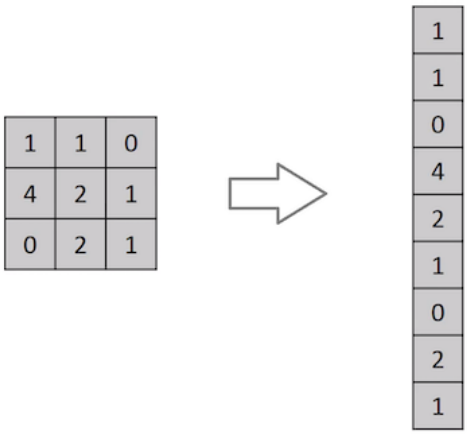


Figure 16: Flattening a 3×3 matrix into a 9×1 vector (source: [12])

CNNs allow for the spatial and temporal dependencies of an image to be captured so that there is a better fitting of the image to the dataset, since there are reduced parameters and weights can be reused. This network essentially can track the complexity of the image to a better degree.

4.1.3 Inputting into a CNN

In the figure below (Fig 17), we have a 3 dimensional matrix that is representative of a RGB matrix. In the real world, there is an incredible amount of complexity in images beyond these images as 8k images are 7680×4320, which means that the sophistication of the image is exponentially greater; this is the reason why CNNs are implemented. Essentially, the convolutional neural net is used to reduce the images into a form that is easier to process so that there can be an accurate prediction of what the image actually is.

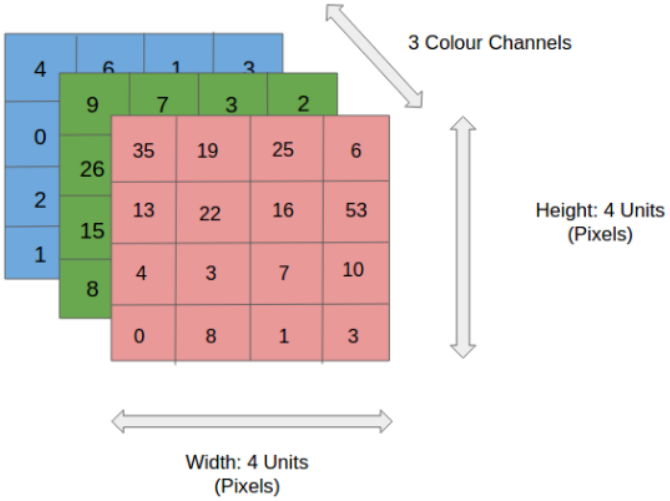


Figure 17: RGB Image Matrix Representation (source: [12])

4.1.4 The Convolution Layer and the Kernel

The image below (Fig 18) represents the $5 \times 5 \times 1$ image that is being processed. The way that this convolution operation is carried out is through something called the kernel, which essentially acts as a filter for the image. In this scenario, the Kernel K, is a $3 \times 3 \times 1$ matrix. The kernel will shift a total of 9 times in order to cover all the squares and it will move from the top left on the screen all the way down to the bottom right to traverse the entirety of the image.

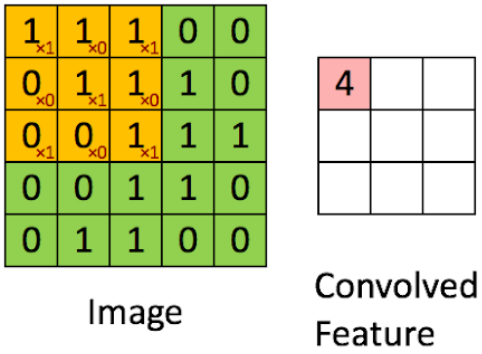


Figure 18: Kernel within a matrix (source: [12])

The Convolution part of the CNN refers to the mathematical calculator that is pulled from the values of the pixels and can be seen below (Fig 19) for a particular RGB image.

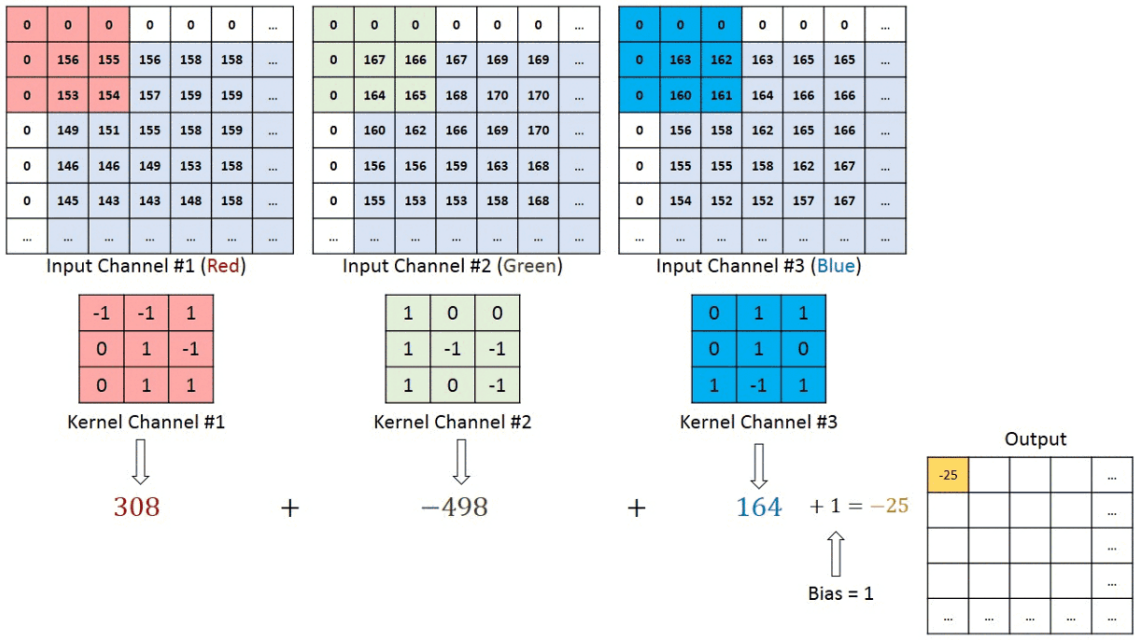


Figure 19: Convolution calculation for a RGB image (source: [12])

4.2 Output of CNNs

4.2.1 Objective of CNN Operations

The objective of the CNN is to extract the high-level features which includes the edges and corners of the input image. CNNs do not always have to only have one Convolutional layer, but it is generally thought out where the first Convolution layer can capture the lesser low-level features such as the edges, color and gradient orientations. With more layers added, then higher level features will be added to the network in order to better the algorithm's understanding of the image.

4.2.2 Result of CNN Operation

There are two possible results that can come out of a CNN operation using a technique called **padding**. One is called **Valid Padding**, where the convolved feature is reduced in dimensionality with techniques such as Principal Component Analysis when compared to the input. The second, called **Same Padding**, is where the dimensionality is increased or it does not change at all.

When a $5 \times 5 \times 1$ image is turned into a $6 \times 6 \times 1$ image, you see that the convolved matrix turns out with dimensions $5 \times 5 \times 1$, hence why this method is named Same Padding. However, if we do this same operation without padding, we are given a matrix with the dimensions smaller than the original ($3 \times 3 \times 1$), hence this is an instance of Valid Padding.

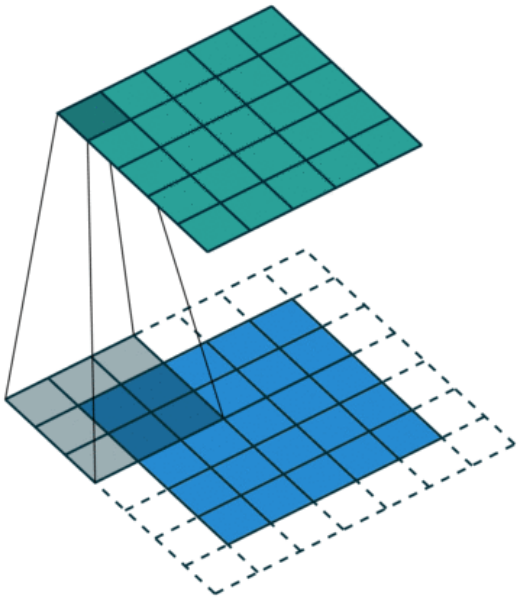


Figure 20: Depiction of padding in a matrix (green matrix is original image padded onto the new image represented by the blue matrix) (source: [12])

4.2.3 Pooling Layer

The final major component of a CNN is the **Pooling layer**. Pooling layers are responsible for reducing the size (dimensionality) of the convolved feature in order to decrease the computational power required to process the data. Furthermore, it is useful for the extraction of dominant features, such as rotational and position invariants (constants in the image) that maintain the effectiveness of the trained model.

4.2.4 Types of Pooling

There are two general types of pooling: **max pooling** and **average pooling**. Max pooling returns the maximum value from the image region that is covered by the kernel, while the average pooling, as its name suggests, returns the average of all the values from the portion of the image covered by the kernel.

Max pooling also is a noise suppressant, which means that it gets rid of noisy activations and also de-noises when dimensions are reduced. On the other hand, average pooling only performs dimensionality reduction as the noise suppressing mechanism. This proves that max pooling actually works out better than average pooling on average.

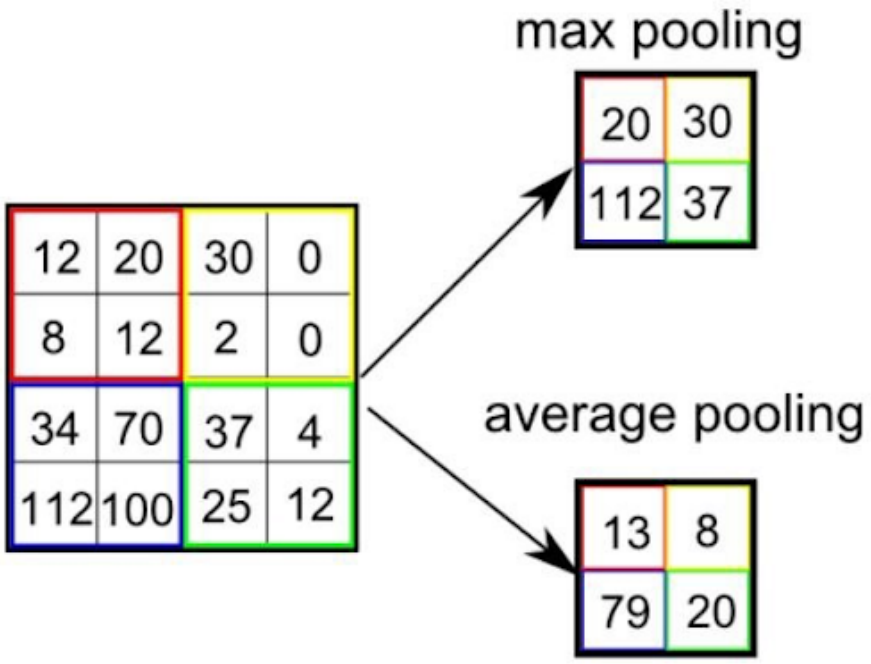


Figure 21: Image showing max and average pooling and how the values compare (source: [12])

4.2.5 Summary of Pooling

The Convolutional layer and the pooling layer together form the ith layer of a CNN, which means that an increased numbers of layers might capture low level details even further, albeit at the cost of more computational power.

The process above displays to us that we have allowed the model to understand the features of an image through the use of CNNs in order to find its most specific features necessary for calculating the intricacies of an image.

References

[1] Rohith Gandhi. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO - Object Detection Algorithms*. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e. July 2018.

[2] *Horn–Schunck method*. https://en.wikipedia.org/wiki/Horn%E2%80%93Schunck_method. Dec. 2020.

[3] *How to Build a Robot Tutorials*. https://www.societyofrobots.com/programming_computer_vision_tutorial_pt4.shtml.

[4] *Lecture 7: Optical Flow and Tracking*. https://web.stanford.edu/class/cs231m/lectures/lecture-7-optical-flow.pdf.

[5] *Lucas–Kanade method*. https://en.wikipedia.org/wiki/Lucas%E2%80%93Kanade_method. Dec. 2020.

[6] *Object Detection Guide*. https://www.fritz.ai/object-detection/#:~:text=Object%20detection%20is%20a%20computer,in%20an%20image%20or%20video.&text=Imagine%2C%20for%20example%2C%20an%20image,of%20them%20within%20the%20image..

[7] *Object Detection vs Object Recognition vs Image Segmentation*. Feb. 2020. URL: %5Curl%7Bhttps://www.geeksforgeeks.org/object-detection-vs-object-recognition-vs-image-segmentation/%7D.

[8] *Object Recognition: 3 Things You Need to Know*. https://www.edge-ai-vision.com/2020/04/object-recognition-3-things-you-need-to-know/. Apr. 2020.

[9] *Object Tracking in Deep Learning*. https://missinglink.ai/guides/computer-vision/object-tracking-deep-learning/.

[10] *Optical flow*. https://en.wikipedia.org/wiki/Optical_flow. Jan. 2021.

[11] Ben Reid. *Memia 2020.10: ACBC // welcome to the new economy // planet healing // think BIG // "precision-ready" digital infrastructure // bubble living on Mars*. https://memia.substack.com/p/memia-202010-acbc-welcome-to-the. Apr. 2020.

[12] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks-the ELI5 way*. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Dec. 2018.