Benha University

Faculty of Science

Arabic Sign Language Recognition and Generating Arabic Speech Using Convolutional Neural Network and YOLO

Submitted to Department of Mathematics
Faculty of Science , Benha University

Presented By

Youssef Hassan Mohamed
Mahmoud Ramadan Abdelwahab
Mostafa Khaled Ahmed
Fares Samy Mohamed
Wijdan Mostafa Mohamed
Nada Emad Mahmoud
Eman Nasr Elsayed
Shaima Hamido Abosreaa
Sohaila Ezzat Youssef
Rawan Emad Mohamed

Supervised By

Dr. Eman Ibrahim

2024

Abstract

Sign language is used by 70 million people around the world and there's a lack in communication between deaf and dump community and our community. Sign language encompasses the movement of the arms and hands as a means of communication for people with hearing disabilities. An automated sign recognition system requires two main courses of action: the detection of particular features and the categorization of particular input data. A vision-based system by applying CNN for the recognition of Arabic hand sign-based letters and translating them into Arabic speech is proposed in this paper. The proposed system will automatically detect hand sign letters and speaks out the result with the Arabic language with a deep learning model. This system gives 97% accuracy to recognize the Arabic hand sign-based letters which assures it as a highly dependable system. After recognizing the Arabic hand sign-based letters, the outcome will be fed to the text into the speech engine which produces the audio of the Arabic language as an output.

The cognitive process enables systems to think the same way a human brain thinks without any human operational assistance. The human brain inspires the cognitive ability On the other hand, deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled which is also known as deep neural learning or deep neural network .

In deep learning, CNN is a class of deep neural networks,
most commonly applied in the field of computer vision.

Table of Contents

List of figures

Chapter one

1.1 INTRODUCTION

Signing has always been part of human communications. Newborns use gestures as a primary means of communication until their speech muscles are mature enough to articulate meaningful speech. For thousands of years, deaf people have created and used signs among themselves. These signs were the only form of communication available for many deaf people. Within the variety of cultures of deaf people all over the world, signing evolved to form complete and sophisticated languages.

Sign language is a form of manual communication and is one of the most natural ways of communication for most people in deaf community. There has been a re-surging interest in recognizing human hand gestures. The aim of the sign language recognition is to provide an accurate and convenient mechanism to transcribe sign gestures into meaningful text or speech so that communication between deaf and hearing society can easily be made.

The significance of using hand gestures for communication becomes clearer when sign language is considered. Sign language is a collection of gestures, movements, postures, and facial expressions corresponding to letters and words in natural languages, so the sign language has more than one form because of its dependence on natural languages.

The sign language is the fundamental communication method between people who suffer from hearing impairments. In order for an ordinary person to communicate with deaf people, an interpreter is usually needed to translate sign language into natural language and vice versa.

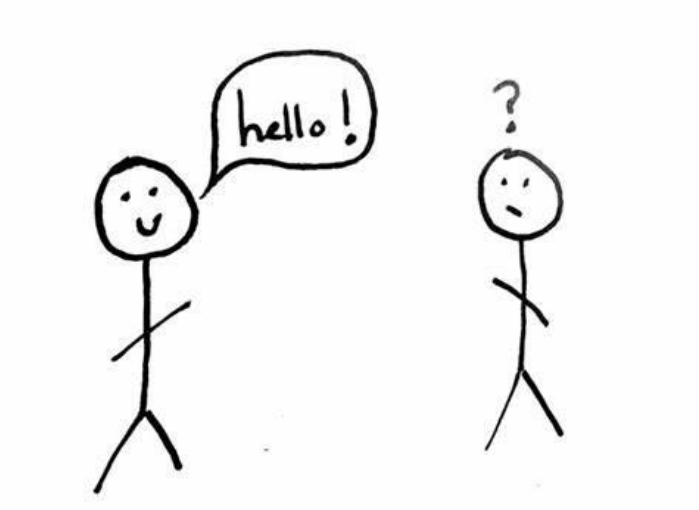
Human-Computer Interaction (HCI) is getting increasingly important as a result of the increasing significance of computer's influence on our lives. Researchers are trying to make HCI faster, easier, and more natural. To achieve this, Human-to-Human Interaction techniques are being introduced into the field of Human-Computer Interaction. One of the richest Human-to-Human Interaction fields is the use of hand gestures in order to express ideas.

1.2 Problem Statement

A person is said to have hearing loss if they are not able to hear as well as someone with normal hearing, meaning hearing thresholds of 20 dB or better in both ears. It can be mild, moderate, moderately severe, severe or profound, and can affect one or both ears. Major causes of hearing loss include congenital or early-onset childhood hearing loss, chronic middle ear infections, noise-induced hearing loss, age-related hearing loss, and ototoxic drugs that damage the inner ear.

The impacts of hearing loss are broad and can be profound. They include a loss of the ability to communicate with others delayed language development in children, which can lead to social isolation, loneliness, and frustration, particularly among older people with hearing loss. Many areas lack sufficient accommodations for hearing loss, which affects academic performance and options for employment. Children with hearing loss and deafness in developing countries rarely receive any schooling.

WHO estimates that unaddressed hearing loss costs the global economy US$ 980 billion annually due to health sector costs (excluding the cost of hearing devices), costs of educational support, loss of productivity, and societal costs.



1.3 Proposed Solution

We used the power of assistive technology to increase the quality of life by opening up new opportunities to disabled people and increasing the range of options open to them.

Our team developed an artificial intelligence (AI) powered system for the deaf and mute people, Implementing our solution on mobile and web applications will offer a low-cost superior approach to translating Arabic sign language into text and vice versa in real-time.

Introducing "SPEAKING FINGERS" the easy-to-use innovative digital interpreter dubbed as "Google Translate for the deaf and mute" works by placing a smartphone or webcam in front of the user while the app translates gestures or sign language into text. SIGNARA uses neural networks and computer vision to recognize the video of a sign language speaker and then smart algorithms translate it into speech. And the vise versa works by animating the input text into sign language.

The proposed Arabic Sign Language Translator system does not rely on using any gloves or visual markings to accomplish the recognition job. As an alternative, it deals with images of bare hands, which allows the user to interact with the system in a natural way.

In this technical documentation, we will show how we tackled our problem giving a detailed way of thinking and design approaches that allowed us to build this prototype.

1.4 Project scope

1- The main goal of our project is when we launch the application and the website, it achieves the target of our project (text or sign language will be translated to Arabic sound).

2- The website and app will help normal people in communication with deaf and dump.

3- Helps us understand their action by converts it to Arabic sound

4- Disable people can easily communicate with outside world.

5- Expected Outcome: Make communication between - Disable people and the rest of human beings easy

6- Project Deliverable: trained models, software, documentation

1.5. Work elements and techniques and Tools

1.5.1 YOLO

YOLOv8 is the newest state-of-the-art YOLO model that can be used for object detection, image classification, and instance segmentation tasks.

1.5.2 TensorFlow

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

1.5.3 Flutter

Flutter is an open-source UI software development kit created by Google. It can be used to develop cross platform applications from a single codebase for the web, Fuchsia, Android, iOS, Linux, macOS, and Windows. First described in 2015, Flutter was released in May 2017. Flutter is used internally by Google in apps such as Google Pay and Google Earthas well as by other software developers including ByteDance and Alibaba.

1.5.4 React

React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library for building user interfaces based on components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies.

1.5.5 Django

Django (/ˈdʒæŋgoʊ/ JANG-goh; sometimes stylized as Django is a free and open-source, Python-based web framework that runs on a web server. It follows the model–template–views (MTV) architectural pattern It is maintained by the Django Software Foundation (DSF), an independent organization established in the US as a 501(c) non-profit.

1.5.6 Php Laravel

Laravel is a free and open-source PHP web framework1234. It is used to develop complex web applications following the model–view–controller (MVC) architectural pattern12. Laravel is designed for developers who need a simple and elegant toolkit to create full-featured web.

1.6.1 Database engine

It is the basic software component that a database management system uses to read and search data from a database and it is the most frequentlyused interchangeably with database server and in our project, we will depend on SQL database engine.
We search about the video equals the sent text (static data).

1.6.2 Design user interfaces (UI)

The goal of user interface design is to produce a user interface which makes it easy, efficient, and enjoyable to operate a machine in the way which produces the desired result.
In our project we will use android, HTML, CSS and JavaScript to design thelayout.

1.6.3 Database server

A database server is a computer program that provides database services to other computer programs or computers as defined by the client–server model and most of the Database servers work with the base of query language.

1.6.4 Deep learning

Deep leaning is a machine learning approach depends on neuralnetworks that produce a very good accuracy and need huge dataset to learn and find a pattern between.

1.7 Stakeholders

1- Disable people.

2- Anyone can use it.

3- Developer.

4- System analyst.

5- Tester.

6- Sponsor.

7- All team members who participated in the work of the project.

1.8 Project Steps

1. Study and analysis the project idea from all aspects.
2. Search and choose the best algorithm that applies our project idea.
3. Modeling and simulating our system.
4. Solve problems and errors in the system.
5. Deploy and implement the system in real world.
6. Test the accuracy of the app and machine learning model.
7. Evolution the system in the future.
8. Integrate the models with mobile app and web site.
9. Deploy the web site and mobile app.

Chapter Two

Background

2.1 CNN

2.2 CNN Layers

2.3 Pooling

2.4 Convolution layer

2.5 Fully connected

2.6 CNN models such as AlexNet VGG16

2.1 CNN

Artificial Intelligence or AI is a monumental breakthrough that bridges the gap between what humans can do and what machine can do. One of many areas that was affected by the development of AI was Computer Vision. Those advancement created an algorithm for the Computer Vision domain that was known as Convolutional Neural Network or CNN for short.

CNNs, like neural networks, are made up of neurons with learnable weights and biases. Humm, wait what are neural networks? Yes, I hate to break it to you, but to understand CNN, you must know what a neural network is first. You can refer to this reference though . Each neuron in CNN will receive several inputs, takes a weighted sum over them, passes it through an activation function and responds with an output. The whole network has a loss function and all the tips and tricks that we developed for neural networks still apply on CNNs. Okay, that's enough of the hard part.

CNN is basically a deep learning algorithm that can take images as its input and by some process of learning able to differentiate one image from another. This result could be achieved by changing the parameters (learnable weights and biases) of the model.

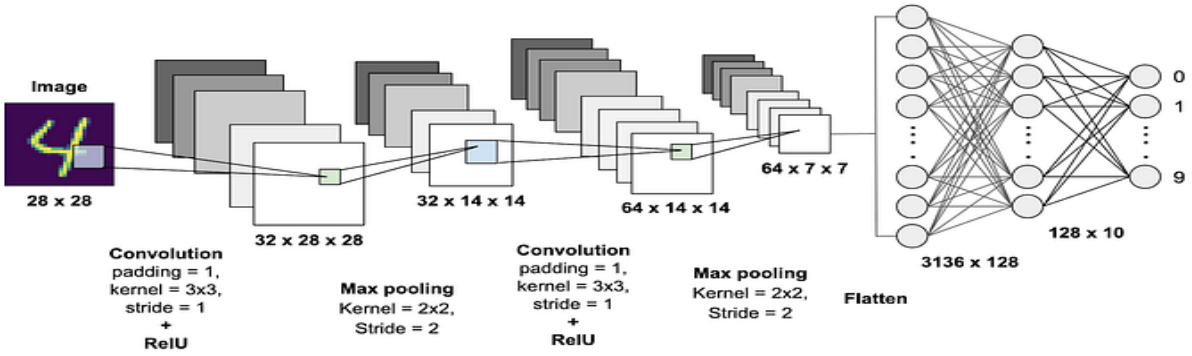


Figure 2.1:CNN Architecture

What makes CNN stand out from other classification techniques for classification of images is the total number of pre-processing required for the CNN / ConvNet is much lower as compared to other classification algorithms. Fun fact, the architecture of CNN and the majority of neural networks itself are really similar to the connectivity pattern inside of human brains and was inspired by the Visual Cortex of humans itself.

Image as an input

An image is a matrix of matrix value that indicates the pixel value for the image. One of the main reasons CNN is really good with classification based on image was because CNN is able to capture the Spatial and Temporal dependencies in an image through the application of relevant filters. Remember, the role of the CNN is to reduce the image into a form that is simpler for the algorithm to process, while still retaining the information of the images. Because of this, CNN takes less time to process images than other algorithms, making it one of the best algorithms to use when trying to tackle image problems.

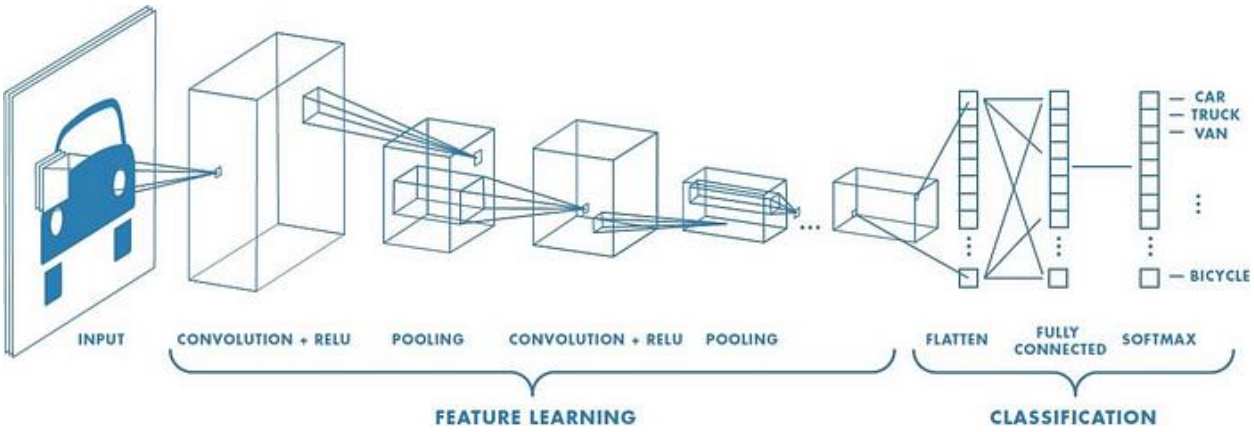
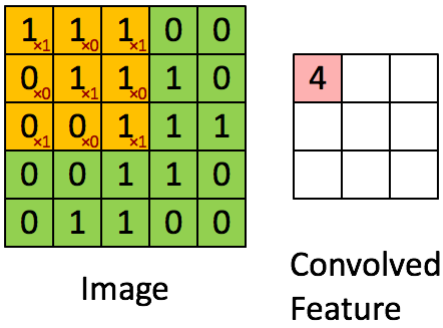


Figure 2.2: CNN Algorithm steps

2.2 CNN Layers

A convolutional layer is a filter (or kernel) in which is an integral component of the layered architecture of the CNN itself. Generally, it refers to the operation applied to the entirety of the inputs (in our case, the image) such that it transforms the information encoded in the pixels into much smaller details. In practice, a kernel is just a smaller-sized matrix from the input size matrix, that consist of real valued entries. Some people say that a picture is worth more than a thousand lectures. So here the representation of the convolutional layer and how it makes information smaller for the neural network to process later on, while still keeping all of the information.



The Kernel shifts 9 times because of Stride Length = 1 (Non-Strided), every time performing **a** matrix multiplication operation between K and the portion P of the image over which the kernel is hovering. Remember, this operation only applies on images with 1 (one) channel like grayscale images. In case of images that have multiple channel (like RGB images), Matrix Multiplication is performed between Kn and In stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.

Alright, it's definitely not easy to wrap all of those things in 1 read. Just remember that the objective of the convolutional layer is extract the high-level features such as edges, from the input image. With added layer onto the neural network architecture, the architecture will try to adapts to the High-Level features as well, which make it has the wholesome understanding of images in the dataset, similar to how we would when trying to identify images when we see it.

Pooling Layer:

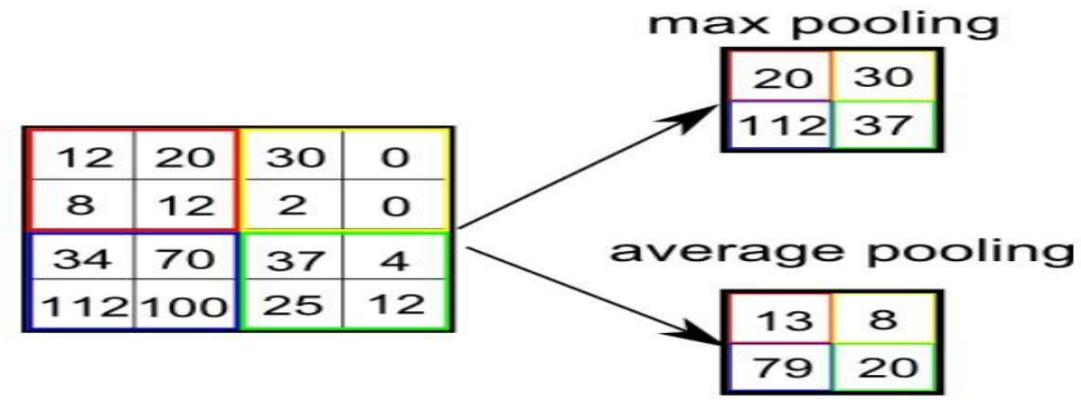


Figure 2.3: Pooling Layer

Alright, next we have a pooling layer. What's the difference between this layer and the previous one though? Well, the pooling layer is essentially the same as the convolutional layer. It is just another building block of a CNN and is responsible for reducing the spatial size of the Convolved Feature. This layer is useful when we are trying to decrease the computational power required to process the data through dimensionality reduction. Not only that, a pooling layer is also useful when trying to extract dominant features from the inputs.

- There are two types of pooling layers:
- Max Pooling.
- Average Pooling.

Max Pooling returns the maximum value from the portion of the image covered by the Kernel.

On the other hand, **Average Pooling** returns the average of all the values from the portion of the image covered by the Kernel. Because of the nature of max pooling layer, it also performs as Noise Suppressant, as it discards the noisy activations altogether and also performs de-noising along with dimensionality reduction.

The ReLu (Rectified Linear Unit) Layer:

So, the Convolutional Layer and the Pooling Layer are the main building blocks of the Convolutional Neural Network. But how do we connect each layer onto each other. Welcoming the ReLu Layer, as it is a layer of an activation function which is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input.

The ReLu layer is not specified only to the Convolutional Neural Network. It is a commonly used layer as an activation function needed in any neural network for transformation in neural networks. As this is a guide to understanding more on the CNN side, I won't explain much about this layer.

Fully Connected Layer:

Now, using both the Convolutional Layer and Pooling layer with the addition of ReLu Layer, we can convert our input image into a more suitable form for our Multi-Level Perceptron by flattening the image into a column vector. The flattened vector then fed to a feed-forward neural network and backpropagation applied to every iteration of training. Okay, this sounds a little bit hard to understand for a beginner.

Let me explain it step by step for you to understand what actually happens when training a neural network, which is the basic foundation for CNN. We train neural networks by repeating the learning process for the model then identify whether our model has already found the pattern of the image input and able to differentiate the image input. After each learning process is done, the model will reevaluate it's parameter on each layer to perform better on the next iteration. This action is called back-propagation and by doing this, the model will perform better and better each time it is trained (might need to watch out for overfitting tho, but this concept is out of this topic).

2.3 Pooling

What is Pooling layer and what does pooling layer do?

In simple words Pooling is used for dimensionality reduction in CNN. Why dimensionality reduction? For decreasing the computational power required to process the data. But pooling is not just for reducing the dimension only, it also helps in extracting the dominant features like edges in the image.

How pooling layer works?

So, now we know that pooling is used for dimensionality reduction but how pooling reduces dimension? Pooling works similar to filters. Consider the below image where we are using a filter of size 2 X 2. In case of filters, we used to multiply filter values to the input element wise and calculate the sum. In case of pooling, we still have the filter window but instead of multiplying we do some operation like taking max or average within the window. For example, in the below image having a 2 X 2 filter with stride 2, we have taken the max of the filter window and taken it as the activation function. Consider the green area where we have taken the max of the 4 values that is 7 and took it as the activation function then filter moves to the next position that is the red area and again it took the maximum value that is 10. Similarly, the filter moves to the orange and blue area and takes the max value from those areas.

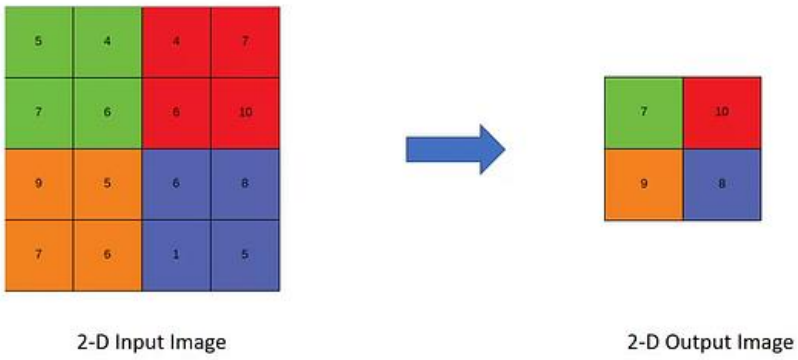


Figure 2.4: Max Pooling Layer 2d

Now we can see that the dimension has been reduced from a 4 X 4 input to a 2 X 2 output. But how pooling extract the dominant features.

In the above image we took the max value of the filter window and if the filter has detected something then the maximum value represents those detections hence taking that value only and throwing away the redundant information.

Now let's see how pooling works with the 3d input image.

Consider the below image with a 3d input image having RGB channels. So, when we apply pooling to 3d image it works independently on each channel. The filter will first go through the red channel and takes the max values for each window and then similarly to the green and blue channels. So, for a 3d input we get a 3d output with reduced dimensions.

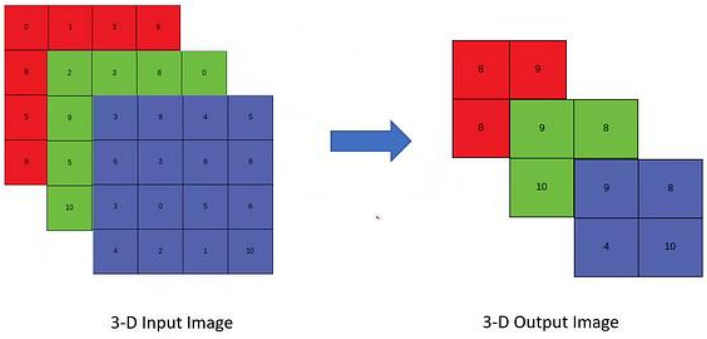


Figure 2.5: Max Pooling Layer 3d

Average pooling:

It is similar in operation as max pooling the only difference is instead of taking the maximum of the values, we take the average of the values in the window.

Earlier we have seen that if we have a 3D input image we will get a 3d output. But in global average pooling if we pass a 3D input, we get a 1D output. This is used when we want the CNN or the feature extraction part to connect to the fully connected part. In global average pooling the filter size is equal to the size of the entire image. So instead to a 2 X 2 filter that we used in max pooling and average pooling the filter size will be 4 X 4 that is equal to the size of the images. It takes the average of the entire channel. For example, it will first take the average of entire red channel then green and then blue and convert it into a 1D vector.

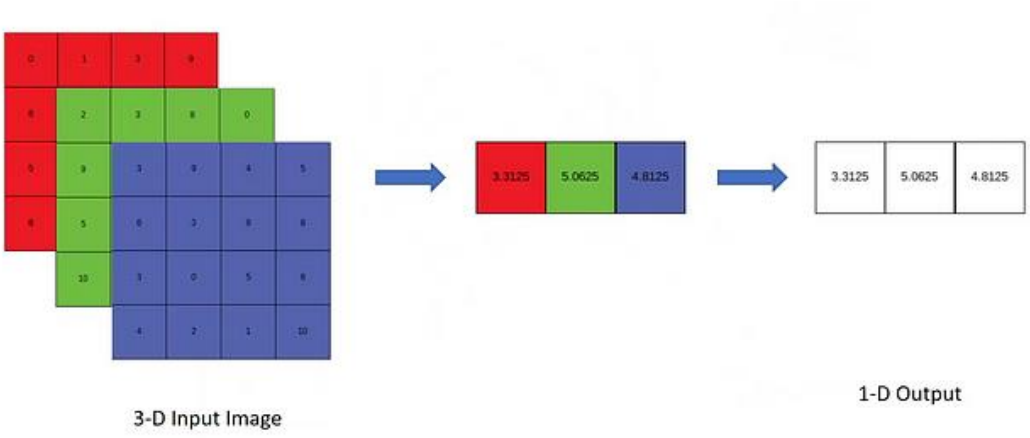


Figure 2.6: Average Pooling Layer 3d

2.4 Convolution layer

The effect of the convolution layer is such that we get a summary of the presence of all the features in this layer. In the convolution layer the input is convolved with a set of weights/filters.

What is Convolution?

Convolution is a linear operation that computes the dot product of the local receptive field and the filter matrix. As the filter matrix is slid over the input, multiple convolved values are produced resulting in a matrix of convolved values. Nonlinearity may be introduced to the output from the convolution layer using a nonlinear activation function.

- **Convolution and Discrete Signals**:

Convolution is a mathematical operation borrowed from digital signal processing to determine the amount of correlation between two signals. The concept is extended to digital images to find the correlation between two images. If one of the images is smaller, then it determines the correlation of the smaller image with every possible window of same size in the other image. Given a discrete signal x(n) of length N and filter f(n) of length M , the amount of correlation between them is determined as

$$(x * f)[n] = \sum_{a=-M/2}^{M/2} x[n-a]\, f[a]$$

Here, the filter is defined over a set of integers from -M/2 and M/2.

- **Cross-correlation**

Convolution operation assumes that the filter f is flipped and shifted across x to get (x*f)[n]. Another terminology used in the similar context is that of **cross-correlation**. The only difference between convolution and cross correlation is that the filter is not flipped in cross correlation.

$$(x \otimes f)[n] = \sum_{a=-M/2}^{M/2} x[n+a]\, f[a]$$

So, what we usually call as convolution when we apply CNN is cross correlation. Given an 2D input image X of size M x M and 2D kernel W of size N x N, the convolution operation can be generalized and written as in the following equation. Feature response at pixel position (i,j) is obtained by computing the dot product between a window of size N x N centered at pixel (i,j) and the filter.

$$(x * w)[i,j] = \sum_{a=-N/2}^{N/2} \sum_{b=-N/2}^{N/2} x(i-a, j-b) w(a,b)$$

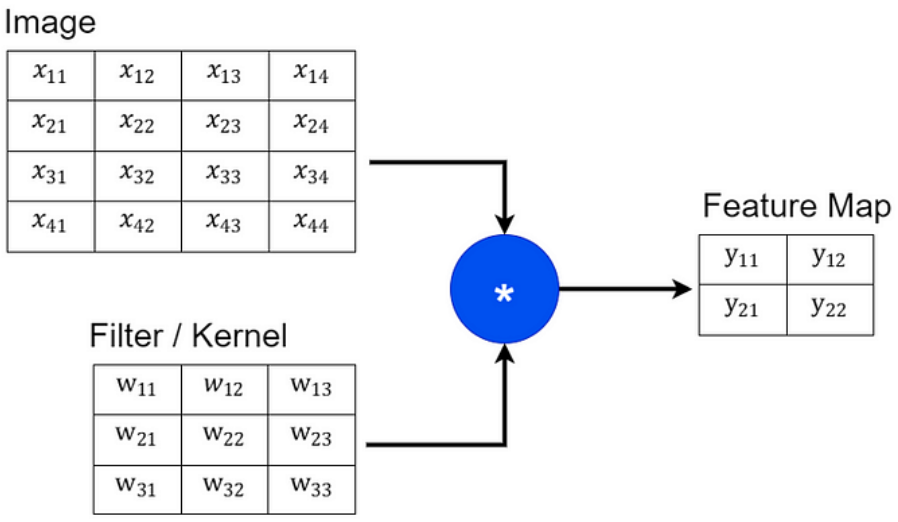


Figure 2.7: shows the convolution between a image of size 4x4 and filter of size 3x3.

Table 1: shows the computation of filter responses. The receptive field (portion of image) which constitutes to the feature response is highlighted.

Feature responses	Receptive field	Computation
y_{11}	x_{11} x_{12} x_{13} x_{14} x_{21} **x_{22}** x_{23} x_{24} x_{31} x_{32} x_{33} x_{34} x_{41} x_{42} x_{43} x_{44}	$x_{11}w_{11} + x_{12}w_{12} + x_{13}w_{13} + x_{21}w_{21} + x_{22}w_{22} + x_{23}w_{23} + x_{31}w_{31} + x_{32}w_{32} + x_{33}w_{33}$
y_{12}	x_{11} x_{12} x_{13} x_{14} x_{21} x_{22} **x_{23}** x_{24} x_{31} x_{32} x_{33} x_{34} x_{41} x_{42} x_{43} x_{44}	$x_{12}w_{11} + x_{13}w_{12} + x_{14}w_{13} + x_{22}w_{21} + x_{23}w_{22} + x_{24}w_{23} + x_{32}w_{31} + x_{33}w_{32} + x_{34}w_{33}$
y_{21}	x_{11} x_{12} x_{13} x_{14} x_{21} x_{22} x_{23} x_{24} x_{31} **x_{32}** x_{33} x_{34} x_{41} x_{42} x_{43} x_{44}	$x_{21}w_{11} + x_{22}w_{12} + x_{23}w_{13} + x_{31}w_{21} + x_{32}w_{22} + x_{33}w_{23} + x_{41}w_{31} + x_{42}w_{32} + x_{43}w_{33}$
y_{22}	x_{11} x_{12} x_{13} x_{14} x_{21} x_{22} x_{23} x_{24} x_{31} x_{32} **x_{33}** x_{34} x_{41} x_{42} x_{43} x_{44}	$x_{22}w_{11} + x_{23}w_{12} + x_{24}w_{13} + x_{32}w_{21} + x_{33}w_{22} + x_{34}w_{23} + x_{42}w_{31} + x_{43}w_{32} + x_{44}w_{33}$

2.5 Fully connected

A fully connected neural network (FCNN) is a type of neural network architecture where each input node is connected to every output node in the subsequent layer. Let's break down how it works:

1. **Neurons and Layers**:
    - In neural networks, individual functions are represented by neurons (also known as perceptrons).
    - A fully connected layer consists of neurons that apply a linear transformation to the input vector using a weights matrix.
    - The output of this transformation is then passed through a non-linear activation function.
2. **Mathematical Representation**:
    - Suppose we have an input vector **x** and a weights matrix **W** for a fully connected layer.
    - The output of the layer can be expressed as:

      $f(x)=W\cdot x$

    - Here, the dot product between **W** and **x** represents the linear transformation.
    - The bias term (often denoted as **W0**) can be added inside the non-linear function, but for simplicity, let's ignore it for now.

3. **Visualizing Fully Connected Layers**:
    - Consider a fully connected layer with an input size of nine and an output size of four.
    - The operation can be visualized as follows: Fully Connected Layer
    - The activation function f wraps the dot product between the input vector and the weights matrix.
4. **Comparison with Convolutional Layers**:
    - Fully connected layers connect all input nodes to all output nodes.
    - In contrast, convolutional layers (used in CNNs) have a local receptive field and only connect a subset of input nodes to each output node.
    - CNNs are particularly effective for tasks like image recognition due to their ability to capture spatial hierarchies.
5. **Deep Learning Architectures**:
    - FCNNs and CNNs serve as the basis for many deep learning architectures.
    - Understanding these fundamental layers helps in designing more complex neural networks.

2.6 CNN models

Problems ranging from image recognition to image generation and tagging have benefited greatly from various deep learning (DL) architectural advancements. Understanding the intricacies of different DL models will help you understand the evolution of the field, and find the right fit for the problems you're trying to solve.

Over the past couple of years many architectures have sprung up varying in many aspects, such as the types of layers, hyperparameters, etc. In this series we'll review several of the most notable DL architectures that have defined the field and redefined our ability to tackle critical problems.

In the first part of this series we'll cover "earlier" models that were published from 2012 to 2014. This includes:

I. **AlexNet (2012)**
II. **VGG 16 (2014)**
III. **GoogleNet**

l. AlexNet (2012)

AlexNet is one of the most popular neural network architectures to date. It was proposed by Alex Krizhevsky for the ImageNet Large Scale Visual Recognition Challenge (ILSVRV), and is based on convolutional neural networks. ILSVRV evaluates algorithms for Object Detection and Image Classification. In 2012, Alex Krizhevsky et al. published *ImageNet Classification with Deep Convolutional Neural Networks*. This is when AlexNet was first heard of.

The challenge was to develop a Deep Convolutional Neural Network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 dataset into more than 1000 different categories. The architecture achieved a top-5 error rate (the rate of not finding the true label of a given image among a model's top-5 predictions) of 15.3%. The next best result trailed far behind at 26.2%.

AlexNet Architecture

The architecture is comprised of eight layers in total, out of which the first 5 are convolutional layers and the last 3 are fully-connected. The first two convolutional layers are connected to overlapping max-pooling layers to extract a maximum number of features. The third, fourth, and fifth convolutional layers are directly connected to the fully-connected layers. All the outputs of the convolutional and fully-connected layers are connected to ReLu non-linear activation function. The final output layer is connected to a softmax activation layer, which produces a distribution of 1000 class labels.

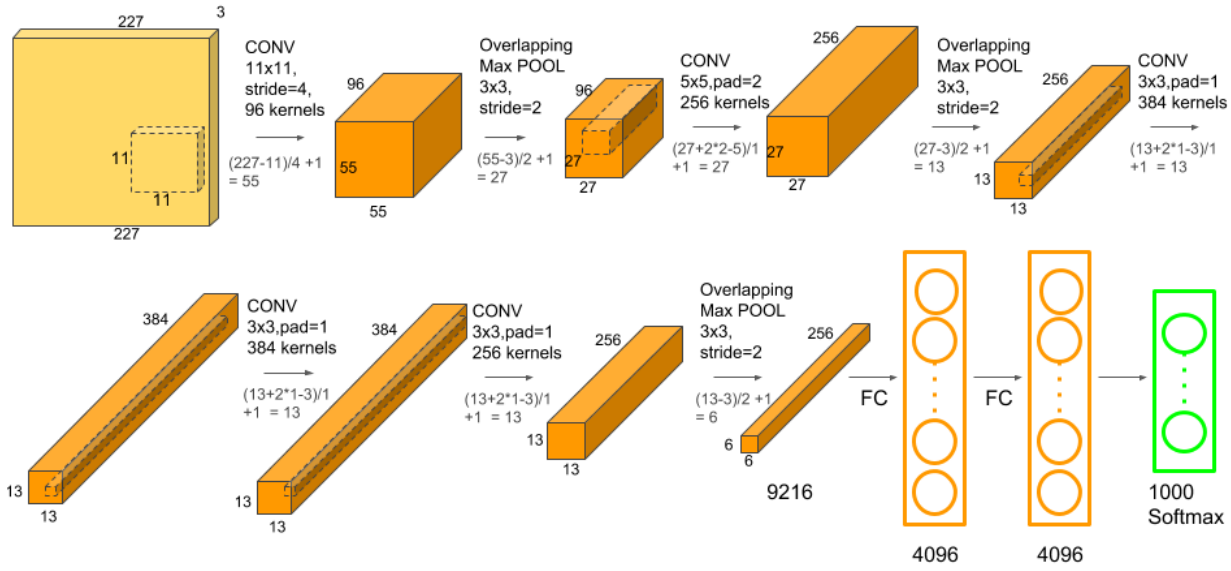


Figure 2.8: AlexNet Architecture

The input dimensions of the network are (256 × 256 × 3), meaning that the input to AlexNet is an RGB (3 channels) image of (256 × 256) pixels. There are more than 60 million parameters and 650,000 neurons involved in the architecture. To reduce overfitting during the training process, the network uses dropout layers. The neurons that are “dropped out” do not contribute to the forward pass and do not participate in backpropagation. These layers are present in the first two fully-connected layers.

AlexNet Training and Results

The model uses a stochastic gradient descent optimization function with batch size, momentum, and weight decay set to 128, 0.9, and 0.0005 respectively. All the layers use an equal learning rate of 0.001. To address overfitting during training, AlexNet uses both data augmentation and dropout layers. It took approximately six days to train on two GTX 580 3GB GPUs for 90 cycles.

Below is a screenshot of the results that were obtained using the AlexNet Architecture:

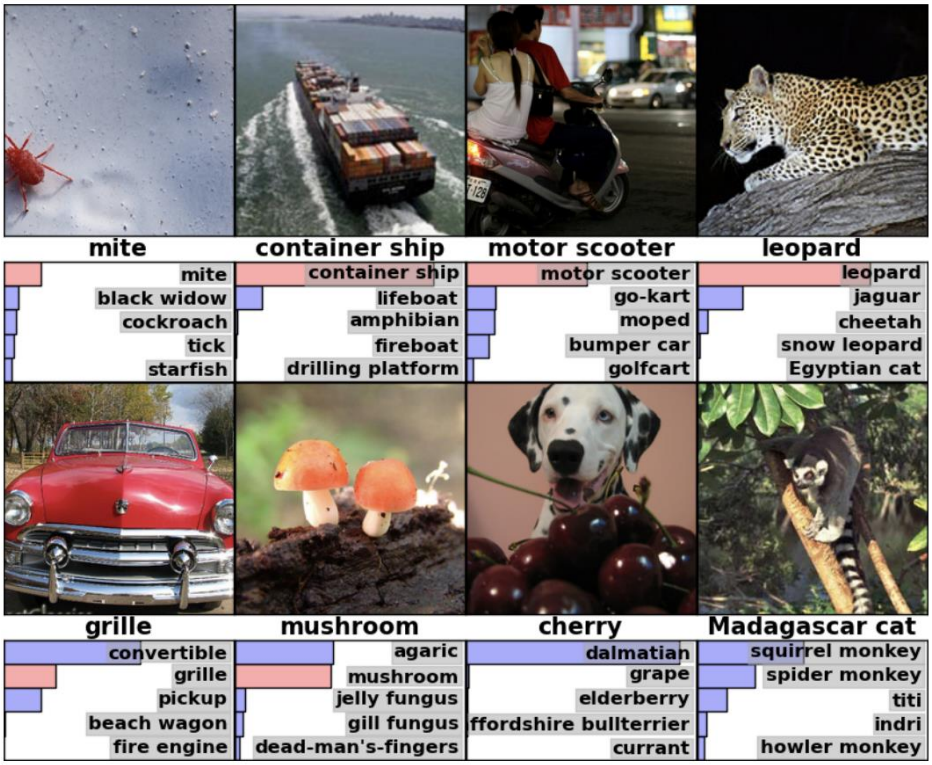


Figure 2.9: Results Using AlexNet on the ImageNet Dataset

Regarding the results on the ILSVRC-2010 dataset, AlexNet achieved top-1 and top-5 test set error rates of 37.5% and 17.0% when the competition was held.

Popular deep learning frameworks like PyTorch and TensorFlow now have the basic implementation of architectures like AlexNet. Below are a few relevant links for implementing it on your own.

II. VGG 16 (2014)

VGG is a popular neural network architecture proposed by Karen Simonyan & Andrew Zisserman from the University of Oxford. It is also based on CNNs, and was applied to the ImageNet Challenge in 2014. The authors detail their work in their paper, *Very Deep Convolutional Networks for large-scale Image Recognition*. The network achieved 92.7% top-5 test accuracy on the ImageNet dataset.

Major improvements of VGG, when compared to AlexNet, include using large kernel-sized filters (sizes 11 and 5 in the first and second convolutional layers, respectively) with multiple (3×3) kernel-sized filters, one after another.

VGG Architecture:

The input dimensions of the architecture are fixed to the image size, (244 × 244). In a pre-processing step the mean RGB value is subtracted from each pixel in an image.

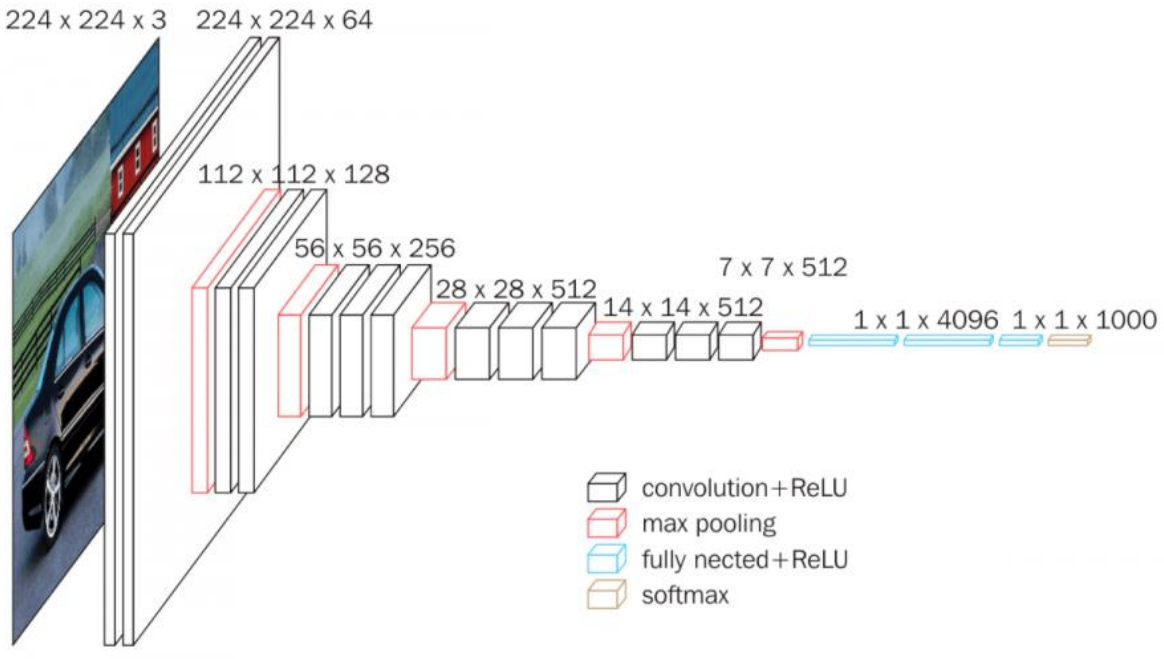


Figure 2.10: VGG Architecture

After the pre-processing is complete the images are passed to a stack of convolutional layers with small receptive-field filters of size (3×3). In a few configurations the filter size is set to (1 × 1), which can be identified as a linear transformation of the input channels (followed by non-linearity).

The stride for the convolution operation is fixed to 1. Spatial pooling is carried out by five max-pooling layers, which follow several convolutional layers. The max-pooling is performed over a (2 × 2) pixel window, with stride size set to 2.

The configuration for fully-connected layers is always the same; the first two layers have 4096 channels each, the third performs 1000-way ILSVRC classification (and thus contains 1000 channels, one for each class), and the final layer is the softmax layer. All the hidden layers for the VGG network are followed by the ReLu activation function.

VGG Configuration, Training, and Results:

The VGG network has five configurations named A to E. The depth of the configuration increases from left (A) to right (B), with more layers added. Below is a table describing all the potential network architectures:

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

All configurations follow the universal pattern in architecture and differ only in depth; from 11 weight layers in network A (8 convolutional and 3 fully-connected layers), to 19 weight layers in network E (16 convolutional and 3 fully-connected layers).

The number of channels of convolutional layers is rather small, starting from 64 in the first layer and then increasing by a factor of 2 after each max-pooling layer, until reaching 512. Below is an image showing the total number of parameters (in millions):

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Training an image on the VGG network uses techniques similar to Krizhevsky et al., mentioned previously (i.e. the training of AlexNet). There are only a few exceptions when multi-scale training images are involved. The entire training process is carried out by optimizing the multinomial logistic regression objective using mini-batch gradient descent based on backpropagation. The batch size and the momentum are set to 256 and 0.9, respectively. The dropout regularization was added for the first two fully-connected layers setting the dropout ratio to 0.5. The learning rate of the network was initially set to 0.001 and then decreased by a factor of 10 when the validation set accuracy stopped improving. In total, the learning rate was reduced 3 times, and the learning was stopped after 370,000 iterations (74 epochs).

VGG16 significantly outperformed the previous generation of models in both the ILSVRC-2012 and ILSVRC-2013 competitions. Concerning the single-net performance, the VGG16 architecture achieved the best result (7.0% test error). Below is a table showing the error rates.

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3

Regarding the hardware and training time, the VGG network took weeks of training using NVIDIA's Titan Black GPUs.

There are two key drawbacks worth noting if you're working with a VGG network. First, it takes a lot of time to train. Second, the network architecture weights are quite large. Due to its depth and number of fully-connected nodes, the trained VGG16 model is over 500MB. VGG16 is used in many deep learning image classification problems; however, smaller network architectures are often more desirable (such as SqueezeNet, GoogleNet, etc.)

Popular deep learning frameworks like PyTorch and TensorFlow have the basic implementation of the VGG16 architecture. Below are a few relevant links.

Chapter Three

3.1 Datasets

we have two Datasets:

I. ArASL: Arabic Alphabets Sign Language Dataset

Abstract:

A fully-labelled dataset of Arabic Sign Language (ArSL) images is developed for research related to sign language recognition. The dataset will provide researcher the opportunity to investigate and develop automated systems for the deaf and hard of hearing people using machine learning, computer vision and deep learning algorithms. The contribution is a large fully-labelled dataset for Arabic Sign Language (ArSL) which is made publically available and free for all researchers.

Data, The dataset which is named ArSL2018 consists of 54,049 images for the 32 Arabic sign language sign and alphabets collected from 40 participants in different age groups. Different dimensions and different variations were present in images which can be cleared using pre-processing techniques to remove noise, center the image, etc. The dataset is made available publicly at https://data.mendeley.com/datasets/y7pckrw6z2/1.

Value of the data

- The current trend of machine learning and deep learning in developing applications helpful in our daily lives such as fingerprint or face recognition and other application in fields such as healthcare, assistive technology, and others. The main core of these applications is image pre-processing, classification and recognition to automate tasks usually done by humans. The ArSL2018 dataset is a valuable resource for researchers in the machine learning and deep learning community for development of assistive technology applications for persons with disability.

- The ArSL2018 dataset collected in Al Khobar, Saudi Arabia is a collection of 54,000 images of the 32 Arabic Sign Language Signs and Alphabet.

- The ArSL2018 is a comprehensive Arabic Sign Language Image repository fully-labelled for purposes of classification and recognition, and for the purpose of applications automating the recognition of sign language for Arabic deaf and hard of hearing individuals.

- The ArSL2018 dataset would assist researchers and allow for faster application development, and faster prototyping of different applications and devices in the assistive technology field.

- The ArSL2018 is a base for the research community to build on this dataset to produce a dataset with more image variations.

Data

The ArSL2018 is a new comprehensive fully labelled dataset of Arabic Sign Language images launched in Prince Mohammad Bin Fahd University, Al Khobar, Saudi Arabia to be made available for researchers in the field of Machine Learning and Deep Learning. It is useful for application and device development in the assistive technology field for the benefit of the deaf and hard of hearing individuals. Examples of related datasets can be found in Refs. The ArSL2018 dataset is unique in the sense that it is the first large comprehensive dataset for Arabic Language Sign Language according to the author(s) knowledge. There is a large potential for this dataset to be used by researchers to both increase accuracies of classification and recognition and for development of prototypes useful for the deaf community.

The ArSL2018 dataset is compiled of 54,049 images in gray scale with 64 × 64 dimension. Variations of images were introduced with different lighting and different background. shows a sample of the pictures of the Arabic Sign Language signs and alphabets in the dataset. In order to assist researchers to access the ArSL2018 dataset for classification and recognition, we have collected, labelled, generated and published the ArSL2018 dataset shows the classification of the Arabic Alphabet signs, with labels and number of images. The dataset has been identified to be sufficient for both training and classification, and has been tested as such. The dataset can be used as is and maybe increased with more variations in the second version of the dataset.

Figure 3.1: Representation of the Arabic Sign Language for Arabic Alphabets.

#	Letter name in English Script	Letter name in Arabic script	# of Images	#	Letter name in English Script	Letter name in Arabic script	# of images
1	Alif	أ(ألِف)	1672	17	Zā	ظ(ظَاء)	1723
2	Bā	ب (بَاء)	1791	18	Ayn	ع(عَين)	2114
3	Tā	ت (أَتَاء)	1838	19	Ghayn	غ(غَين)	1977
4	Thā	ث (ثَاء)	1766	20	Fā	ف(فَاء)	1955
5	Jīm	ج (جِيمْ)	1552	21	Qāf	ق (قَاف)	1705
6	Hā	ح (حَاء)	1526	22	Kāf	ك(كَاف)	1774
7	Khā	خ (خَاء)	1607	23	Lām	ل(لاَمْ)	1832
8	Dāl	د (دَالْ)	1634	24	Mīm	م(مِيمْ)	1765
9	Dhāl	ذ (ذَال)	1582	25	Nūn	ن(نُون)	1819
10	Rā	ر (رَاء)	1659	26	Hā	ه(هَاء)	1592
11	Zāy	ز (زَاي)	1374	27	Wāw	و(وَاو)	1371
12	Sīn	س (سِينْ)	1638	28	Yā	ئ (يَا)	1722
13	Shīn	ش (شِينْ)	1507	29	Tāa	ة(ة)	1791
14	Sād	ص(صَادْ)	1895	30	Al	ال(ال)	1343
15	Dād	ض(ضَاد)	1670	31	Laa	لا(لا)	1746
16	Tā	ط(طَاء)	1816	32	Yāa	يَاء (يَاء)	1293

Table 2. Input Arabic Alphabet Sign classes with their labels and number of images.

Experimental design, materials, and methods

The ArSL2018 dataset images were taken at Prince Mohammad Bin Fahd University and in the Khobar Area, Kingdom of Saudi Arabia from volunteers of different age groups. A smart Camera attached to tripod was used to capture the images. Volunteers were made to stand around 1 m away from the camera. Variations of images were introduced with different lighting, angles, timings and different background. The total number of images per alphabet varies, however, the total number of images compiled for the dataset were 54,049 images. The images were taken in RGB format with different dimensions and variations, which required pre-processing the images to make the suitable for classification and recognition. The collected images were resized to a fixed dimension 64×64 and converted to grayscale images, with a range of pixel values 0 to 255.

II. RGB Arabic Alphabet Sign Language (AASL) dataset Image Dataset

The contribution is a large fully-labelled dataset for Arabic Sign Language (ArSL) which is made publically available and free for all researchers. The dataset which is named RGB Arabic Alphabet Sign Language (AASL) dataset Image Dataset consists of 21868 images for the 31 Arabic sign language sign and alphabets collected from 40 participants in different age groups. Different dimensions and different variations were present in images which can be cleared using pre-processing techniques to remove noise.

1. Data split

TRAIN	TEST	VALID
19496	1384	988

2. Preprocessing

- **Preprocessing steps :**
  - Auto-Orient: Applied
  - Resize: Stretch to 640x640

- **Augmentations :**
  - Rotation: Between -4° and +4°
  - Cutout: 3 boxes with 10% size each
  - Grayscale: Applied .

3. Augmentations

- Outputs per training example: 1
- Rotation: Between -4° and +4°
- Cutout: 3 boxes with 10% size each

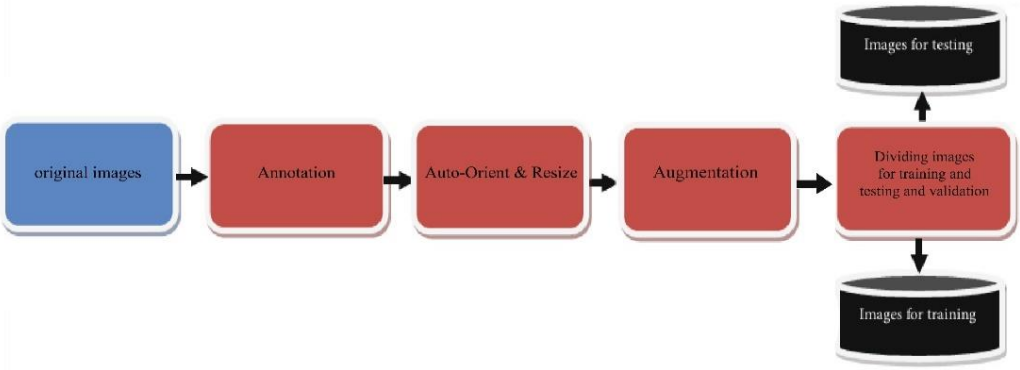


Figure 3.2: Augmentations data

4. Example of images in dataset:

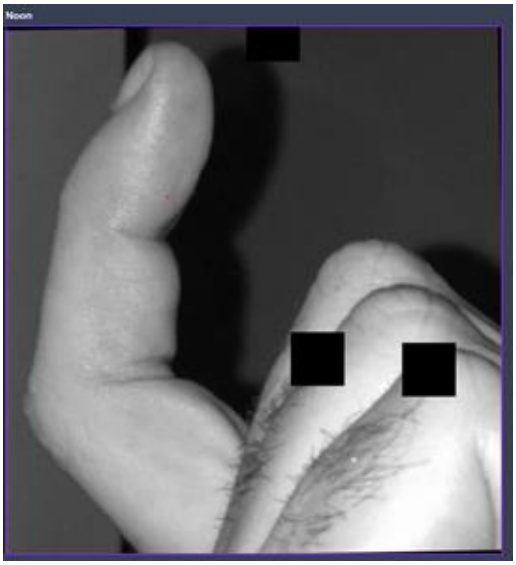


Figure 3.3: example of images in dataset

5. Classes:

#	Letter name in English Script	Letter name in Arabic script	# of Images	#	Letter name in English Script	Letter name in Arabic script	# of images
1	Alef	أ(أَلِف)	2800	17	Thal	ظ(ظَاء)	136
2	Beh	ب (بَاء)	343	18	Ayn	ع(عَين)	251
3	The	ت (أَتَاء)	397	19	Ghain	غ(غَين)	627
4	Thā	ث (ثَاء)	2168	20	Feh	ف(فَاء)	975
5	Jeem	ج (جِيمْ)	217	21	Qāf	ق (قَاف)	426
6	Hā	ح (حَاء)	398	22	Kāf	ك(كَاف)	186
7	Khah	خ (خَاء)	96	23	Lām	ل(لَامْ)	513
8	Dal	د (دَالْ)	476	24	Meem	م(مِيمْ)	3060
9	Thāl	ذ (ذَال)	881	25	Noon	ن(نُون)	164
10	Reh	ر (رَاء)	266	26	Hah	ه(هَاء)	210
11	Zain	ز (زَاي)	478	27	Wāw	و(وَاو)	155
12	Seen	س (سِينْ)	464	28	Teh_Marbuta	ة(ة)	156
13	Sheen	ش (شِينْ)	296	29	Al	ال(ال)	2077
14	Sād	ص(صَادْ)	698	30	Laa	لا(لا)	1598
15	Dād	ض(ضَاد)	88	31	Yeh	ي (يَاء)	160
16	Tā	ط(طَاء)	1105				

CNN Model:

When working with images, the best approach is a CNN (Convolutional Neural Network) architecture.

The image passes through Convolutional Layers, in which several filters extract important features.

After passing some convolutional layers in sequence, the output is connected to a fully-connected Dense network.

I. Model Architecture:

The CNN architecture outlined in the provided summary consists of several layers organized in a sequential manner:

1. Convolutional Layers:

- The first convolutional layer (conv2d_13) has 64 filters with a kernel size of 3x3, resulting in an output shape of 128x128x64.
- The subsequent convolutional layer (conv2d_14) also has 64 filters with a kernel size of 3x3, maintaining the output shape of 128x128x64.
- Max pooling (max_pooling2d_5) is applied with a pool size of 2x2, reducing the output shape to 64x64x64.
- This pattern of alternating convolutional layers and max pooling layers is repeated to extract hierarchical features at different scales.

2. Additional Convolutional Layers:

- The network further deepens with more convolutional layers, increasing the number of filters and complexity of features learned at each stage.
- Each convolutional layer is followed by another convolutional layer to capture more intricate patterns in the data.
- Max pooling is applied after every two convolutional layers to down sample the feature maps.

3. Fully Connected Layers:

- After the convolutional layers, the feature maps are flattened into a vector using the Flatten layer.
- Two dense (fully connected) layers (dense_3, dense_4) with 256 and 64 neurons, respectively, are added to transition from spatial hierarchies to class scores.
- The final dense layer (dense_5) with 32 neurons represents the output classes corresponding to the sign language gestures.

Overall, this deep CNN architecture leverages multiple layers of convolutions and pooling operations to extract intricate features from input images and classify them into one of the 32 sign language classes. The dense layers at the end help in mapping these features to the respective classes for accurate recognition.

Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 128, 128, 64)	1792
conv2d_14 (Conv2D)	(None, 128, 128, 64)	36928
max_pooling2d_5 (MaxPooling2D)	(None, 64, 64, 64)	0
conv2d_15 (Conv2D)	(None, 64, 64, 128)	73856
conv2d_16 (Conv2D)	(None, 64, 64, 128)	147584
max_pooling2d_6 (MaxPooling2D)	(None, 32, 32, 128)	0
conv2d_17 (Conv2D)	(None, 32, 32, 256)	295168
conv2d_18 (Conv2D)	(None, 32, 32, 256)	590080
conv2d_19 (Conv2D)	(None, 32, 32, 256)	590080

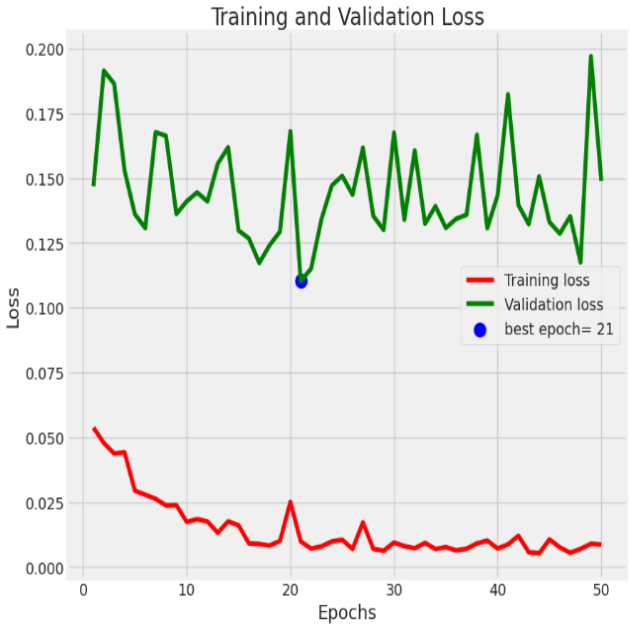
```
 max_pooling2d_7 (MaxPoolin  (None, 16, 16, 256)       0
 g2D)

 conv2d_20 (Conv2D)          (None, 16, 16, 512)       1180160

 conv2d_21 (Conv2D)          (None, 16, 16, 512)       2359808

 conv2d_22 (Conv2D)          (None, 16, 16, 512)       2359808

 max_pooling2d_8 (MaxPoolin  (None, 8, 8, 512)         0
 g2D)

 conv2d_23 (Conv2D)          (None, 8, 8, 512)         2359808

 conv2d_24 (Conv2D)          (None, 8, 8, 512)         2359808

 conv2d_25 (Conv2D)          (None, 8, 8, 512)         2359808

 max_pooling2d_9 (MaxPoolin  (None, 4, 4, 512)         0
 g2D)

 flatten_1 (Flatten)         (None, 8192)              0

 dense_3 (Dense)             (None, 256)               2097408

 dense_4 (Dense)             (None, 64)                16448

 dense_5 (Dense)             (None, 32)                2080

=================================================================
Total params: 16830624 (64.20 MB)
Trainable params: 16830624 (64.20 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

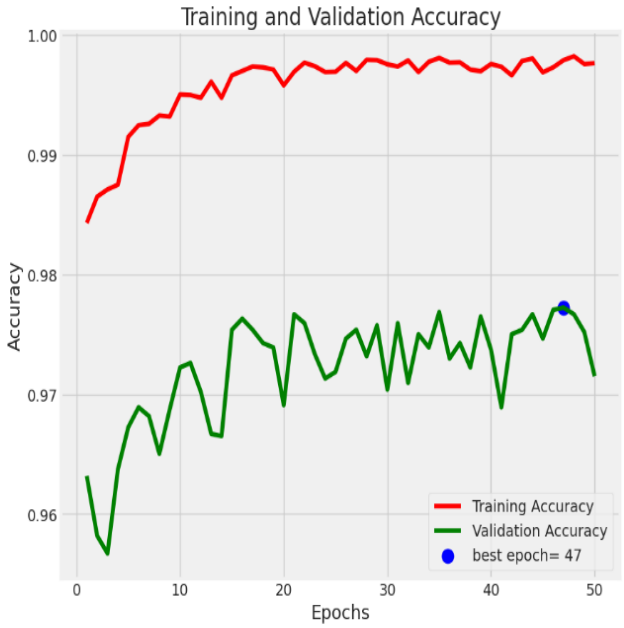
II. Results from the (CNN) model:

Results from the Convolutional Neural Network (CNN) model for the Sign Language Recognition project were highly encouraging, achieving an accuracy of 97% on the test dataset. The model was trained on a dataset of 54,049 images with 32 classes representing different sign language gestures. To enhance the model's performance, data augmentation techniques were applied during training.

Training Result:

This Figure take training model and plot history of accuracy and losses with the best epoch in both of them.





- Training Accuracy = 0.9977 Validation Accuracy = 0.9715
- Training Loss = 0.0087 Validation Loss = 0.1489

Testing Result:

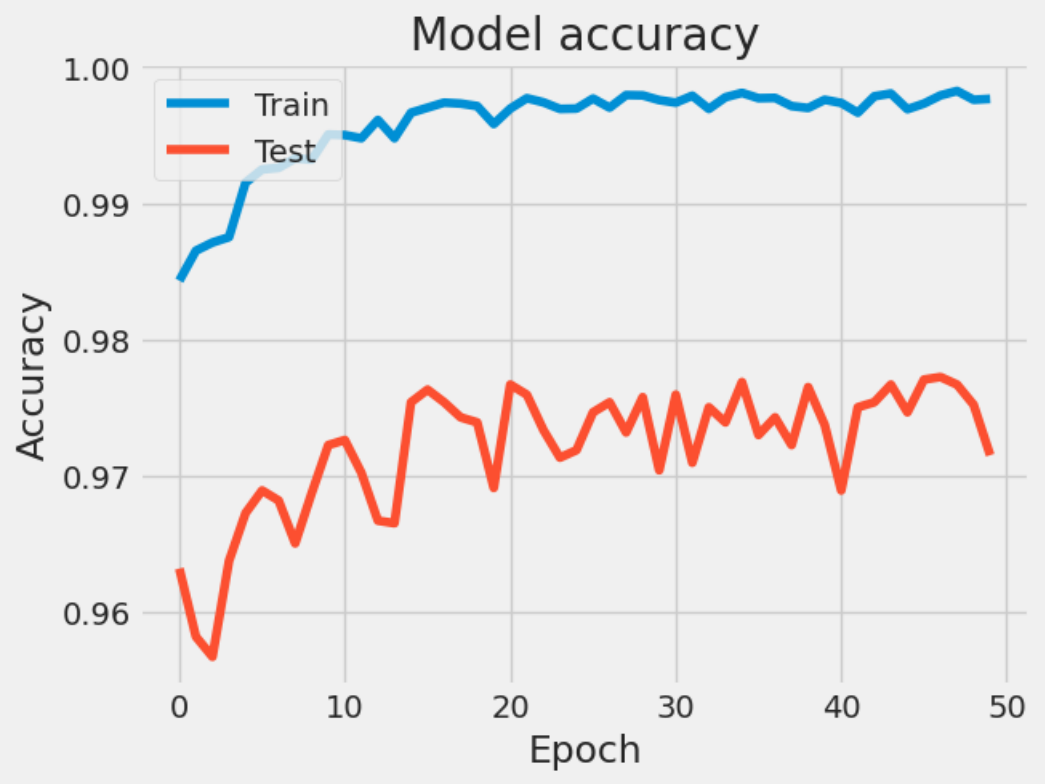


Figure3.4 : Plot training & testing accuracy values

- Training Accuracy = 0.9977
- Testing Accuracy = 0.9715

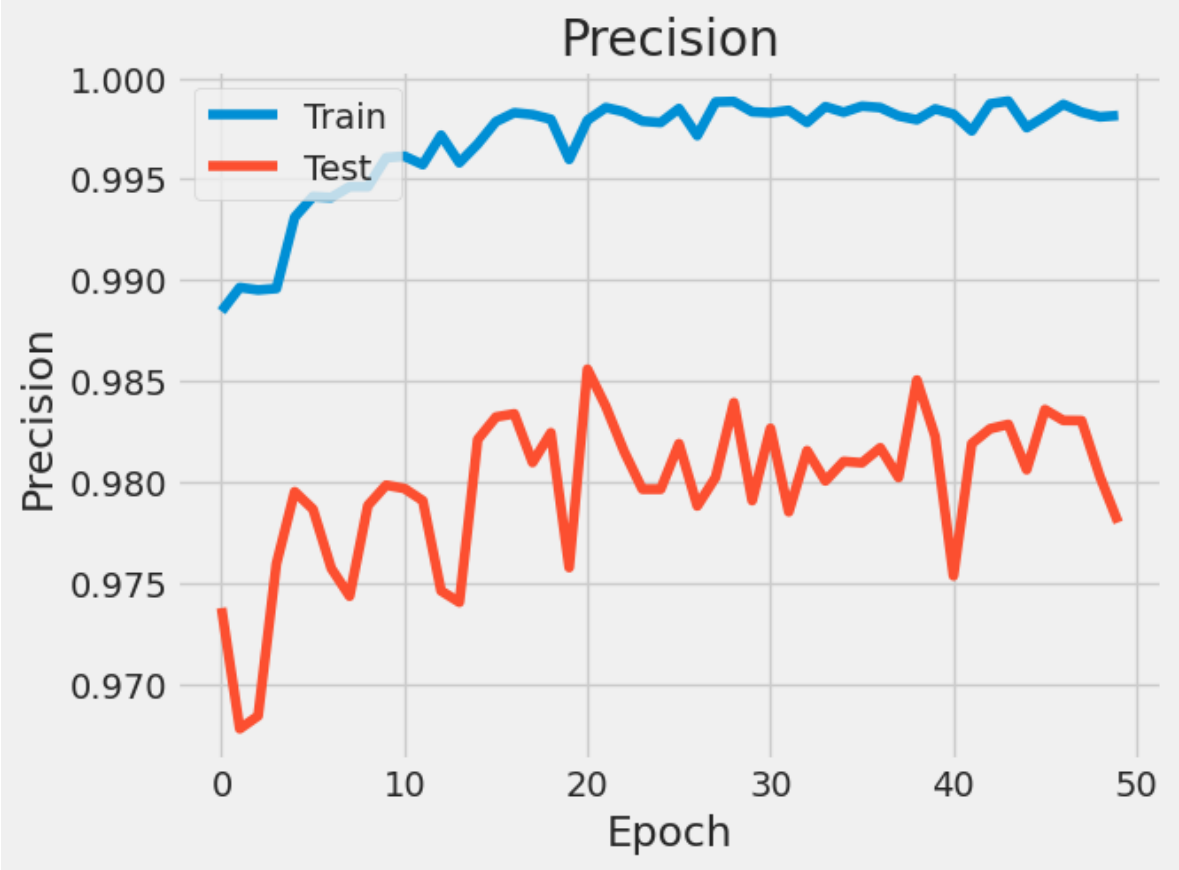


Figure 3.5: Training & Validation precision values

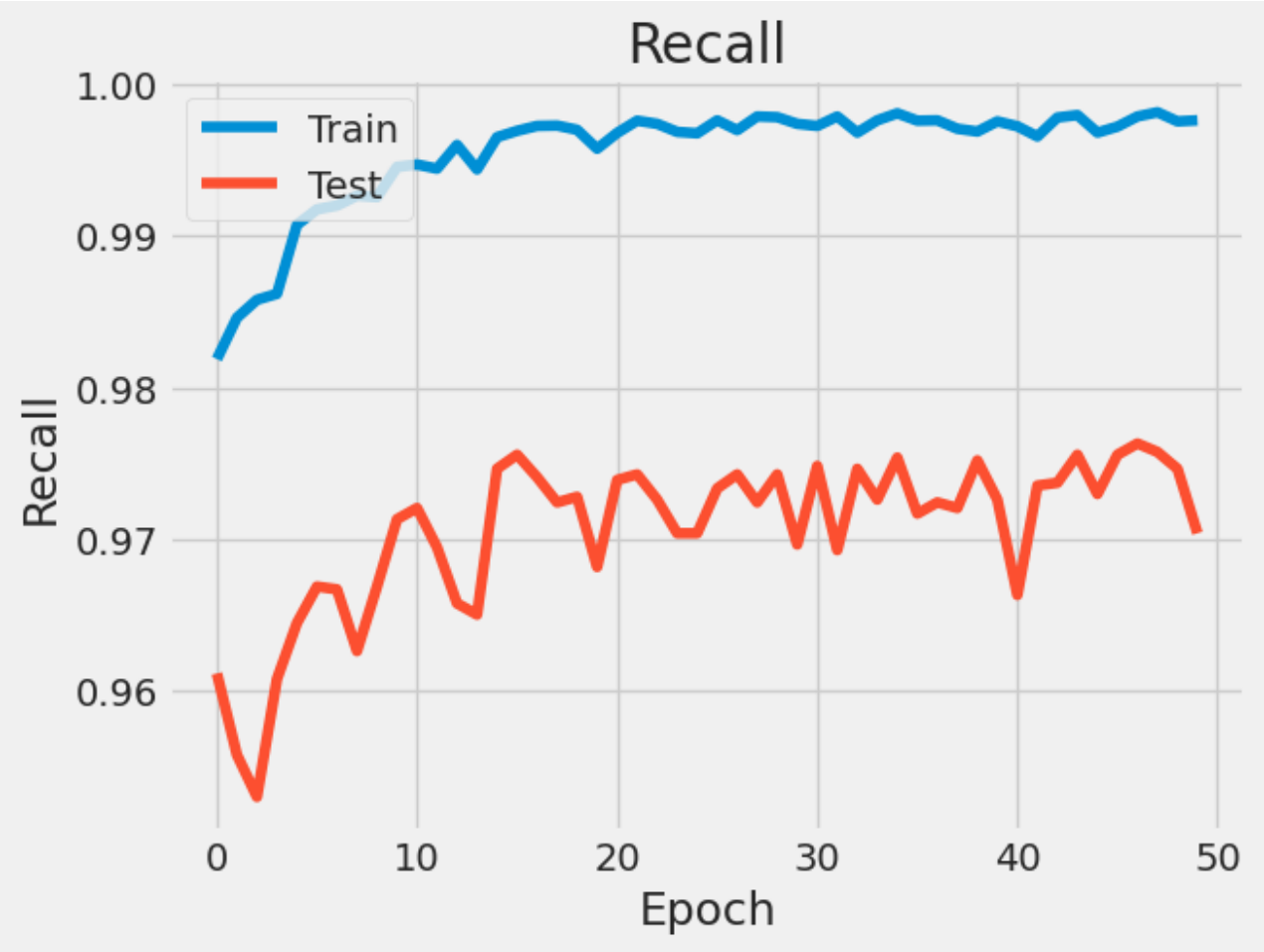


Figure 3.6. Plot training & validation recall values

Confusion Matrix:

The confusion matrix revealed that the model performed well across all 32 classes, with accuracy ranging from 95% to 97.15% for individual classes. The precision and recall scores were calculated for each class, demonstrating an average precision of 0.978 and an average recall of 0.97.

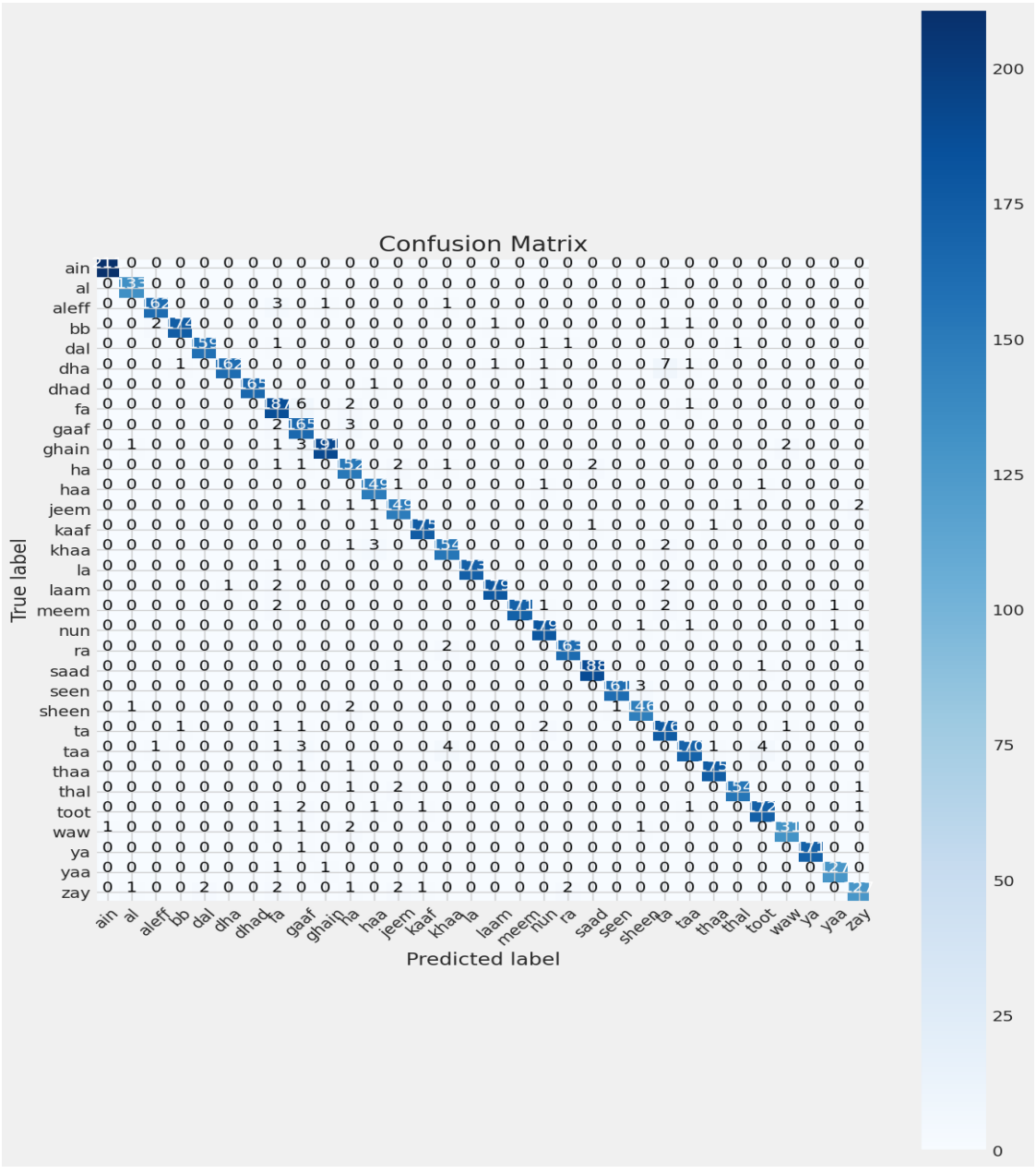


Figure 3.7: Confusion matrix

3.3 YOLO Model

I. What is YOLOv8?

YOLOv8 is the newest state-of-the-art YOLO model that can be used for object detection, image classification, and instance segmentation tasks. YOLOv8 was developed by Ultralytics, who also created the influential and industry-defining YOLOv5 model. YOLOv8 includes numerous architectural and developer experience changes and improvements over YOLOv5.

YOLOv8 is under active development as of writing this post, as Ultralytics work on new features and respond to feedback from the community. Indeed, when Ultralytics releases a model, it enjoys long-term support: the organization works with the community to make the model the best it can be.

II. YOLOv8 Architecture: A Deep Dive

YOLOv8 does not yet have a published paper, so we lack direct insight into the direct research methodology and ablation studies done during its creation. With that said,

Here we provide a quick summary of impactful modeling updates and then we will look at the model's evaluation, which speaks for itself.

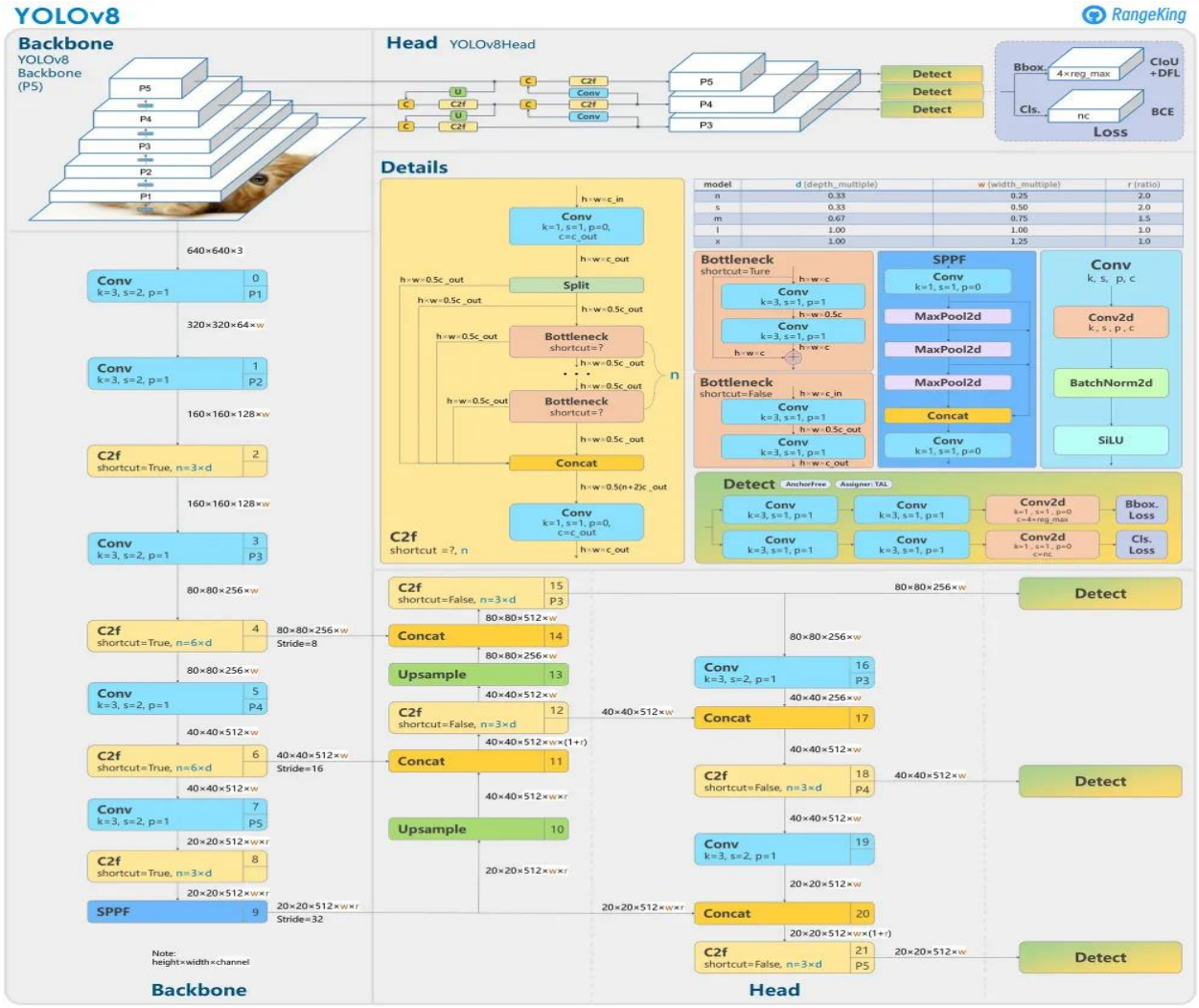


Figure 3.7: YOLOv8 Architecture

III. Anchor Free Detection

YOLOv8 is an anchor-free model. This means it predicts directly the center of an object instead of the offset from a known anchor box.

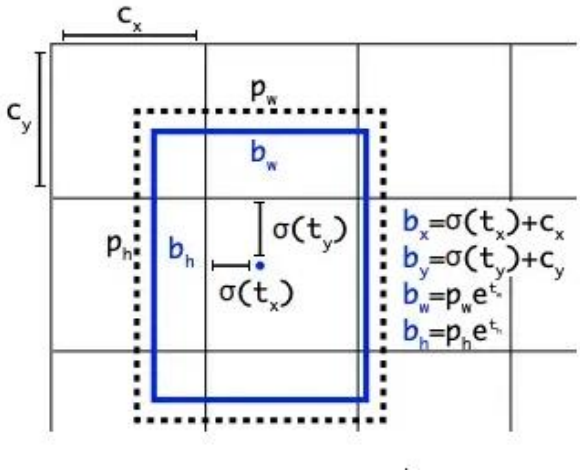
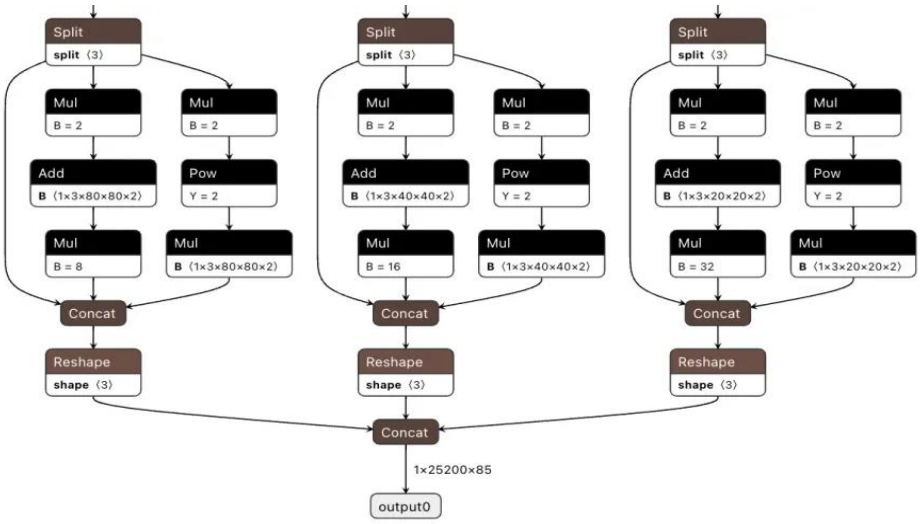


Figure 3.8. Anchor Free Detection

Anchor boxes were a notoriously tricky part of earlier YOLO models, since they may represent the distribution of the target benchmark's boxes but not the distribution of the custom dataset.

Figure 3.9



Anchor free detection reduces the number of box predictions, which speeds up Non-Maximum Suppression (NMS), a complicated post processing step that sifts through candidate detections after inference.

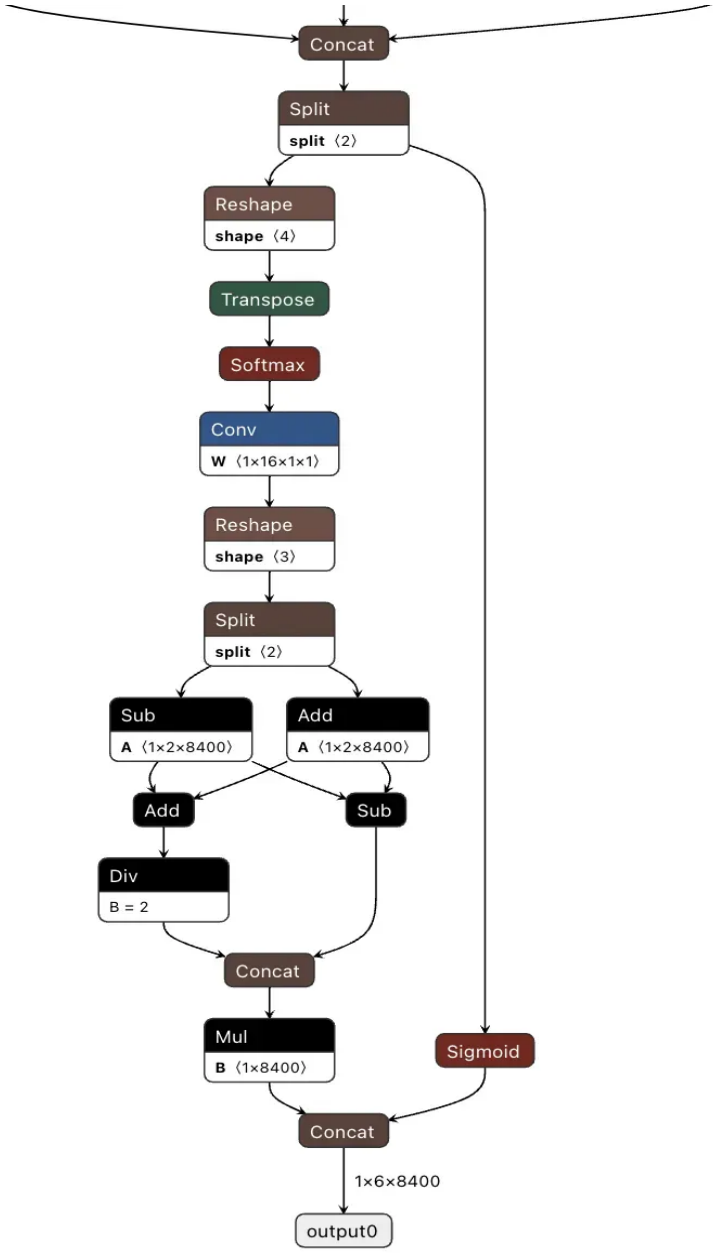


Figure 3.10

IV. New Convolutions

The stem's first 6x6 conv is replaced by a 3x3, the main building block was changed, and C2f replaced C3. The module is summarized in the picture below, where "f" is the number of features, "e" is the expansion rate and CBS is a block composed of a Conv, a BatchNorm and a SiLU later.

In C2f, all the outputs from the Bottleneck (fancy name for two 3x3 convs with residual connections) are concatenated. While in C3 only the output of the last Bottleneck was used.

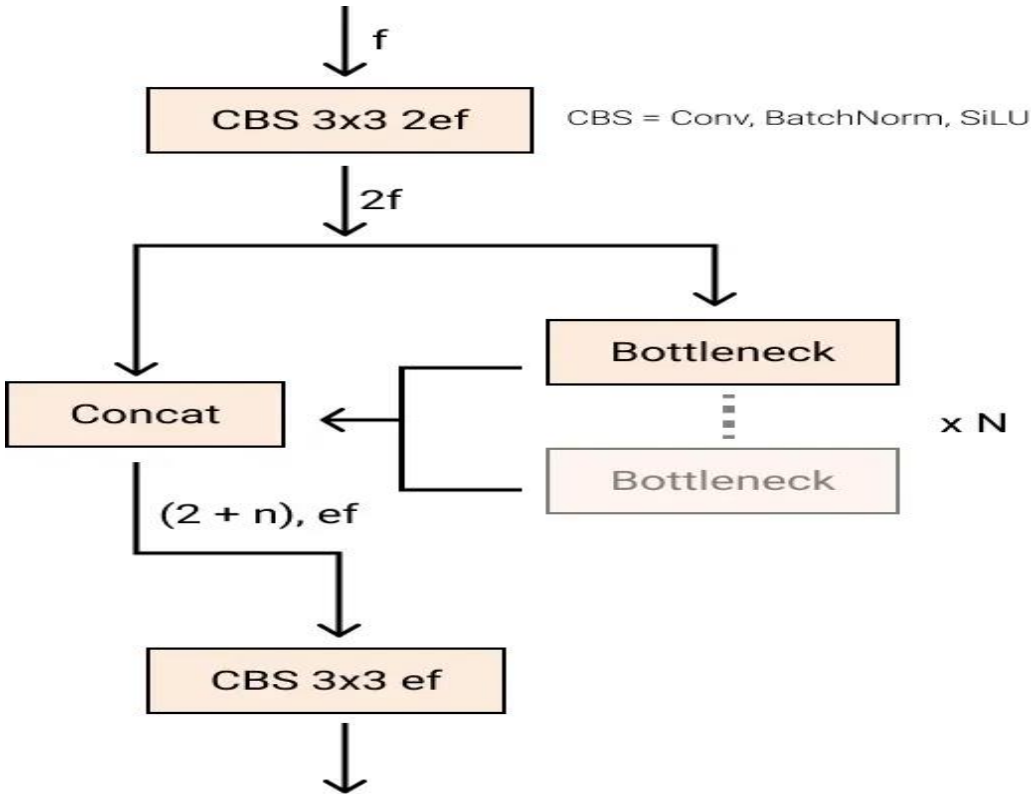


Figure 3.11

The `Bottleneck` is the same as in YOLOv5 but the first conv's kernel size was changed from `1x1` to `3x3`. From this information, we can see that YOLOv8 is starting to revert to the ResNet block defined in 2015.

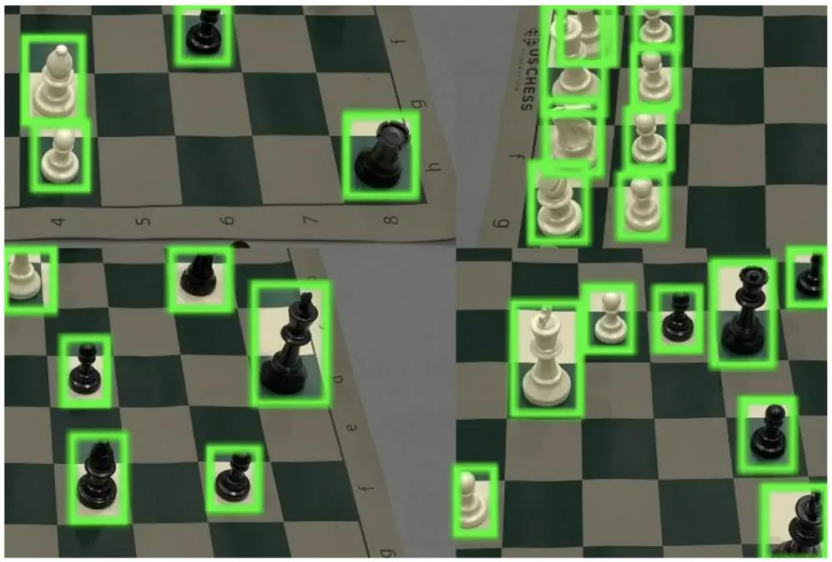
In the neck, features are concatenated directly without forcing the same channel dimensions. This reduces the parameters count and the overall size of the tensors.

Closing the Mosaic Augmentation:

Deep learning research tends to focus on model architecture, but the training routine in YOLOv5 and YOLOv8 is an essential part of their success.

YOLOv8 augments images during training online. At each epoch, the model sees a slightly different variation of the images it has been provided.

One of those augmentations is called mosaic augmentation. This involves stitching four images together, forcing the model to learn objects in new locations, in partial occlusion, and against different surrounding pixels.



V. Yolo Model Results

Class	P	R	mAP50
Ain	0.896	0.915	0.983
Al	0.658	0.967	0.917
Alef	0.848	1	0.991
Beh	0.972	1	0.995
Dad	0.977	1	0.995
Dal	0.973	1	0.995
Feh	0.933	1	0.995
Ghain	0.668	1	0.886
Hah	0.977	1	0.995
Heh	1	0.819	0.995
Jeem	1	0.824	0.995
Kaf	0.852	1	0.995
Khah	0.931	1	0.995
Laa	1	0.714	0.995
Lam	0.939	1	0.995
Meem	0.819	0.84	0.896
Noon	1	0.792	0.995
Qaf	0.791	1	0.995
Reh	0.976	1	0.995
sad	0.976	1	0.995
seen	0.788	1	0.995
sheen	0.768	1	0.995
tah	0.653	1	0.995
teh	1	1	0.995
Teh_Marbuta	1	0	0.995
Thal	0.62	0	0.995
Theh	0.8	0	0.995
Waw	1	0	0.995
Yah	0.78	0	0.995
Zah	0.9	0	0.995
Zain	0.933	1	0.995

Column Definitions:

- Class: This is the category or type of object the model was trying to detect or classify.
- Instances: Number of instances or examples of each class present in the dataset or during the evaluation.
- P (Precision): Precision indicates the accuracy of the positive predictions. It is the ratio of correctly predicted positive observations to the total predicted positives. High precision relates to a low rate of false positives.
- R (Recall): Recall (also known as sensitivity) is the ratio of correctly predicted positive observations to all actual positives. High recall relates to a low rate of false negatives.
- mAP50 (mean Average Precision at IoU = 0.50): This metric evaluates the mean average precision of the model at an Intersection over Union (IoU) threshold of 0.50. It is a common metric in object detection that measures the model's accuracy in terms of detecting objects with a bounding box that overlaps at least 50% with the ground-truth bounding box.

Analysis of Results:

- **High Performers:**

- Ain, Beh, Dad, Dal, Feh, Hah: These classes show very high precision and recall, indicating that the model performs excellently in detecting these classes with very few false positives or false negatives. Their mAP50 values are close to 1, showing almost perfect accuracy in bounding box predictions.

- **Variable Precision and Recall:**

- Al: Despite a high recall of 0.967, its precision is only 0.658, indicating some false positives are being predicted as 'Al'.
- Ghain: Shows lower precision (0.668) but perfect recall (1), suggesting the model captures all 'Ghain' instances but with some false positives.
- Teh_Marbuta, Thal, Theh, Waw: These classes have precision, recall, or both set to 0 despite having a high mAP50. This suggests possible issues in data labeling or errors in the calculation/reporting. Specifically, an mAP50 of 0.995 with a recall of 0 at the same time is inconsistent under normal circumstances.

Conclusion:

The model shows strong performance across many classes with high precision and recall values, indicating effective learning and prediction capabilities, especially for classes with a sufficient number of instances. However, the anomalies in some classes with zero precision or recall but high mAP50 indicate potential issues in data processing, labeling, or metric calculation that might need a deeper investigation.

In machine learning, especially in object detection tasks, having balanced and sufficiently large datasets for each class, reliable labeling, and consistent metric calculations are crucial for developing a robust model. The discrepancies observed suggest revisiting these aspects for certain classes.

Metrics:

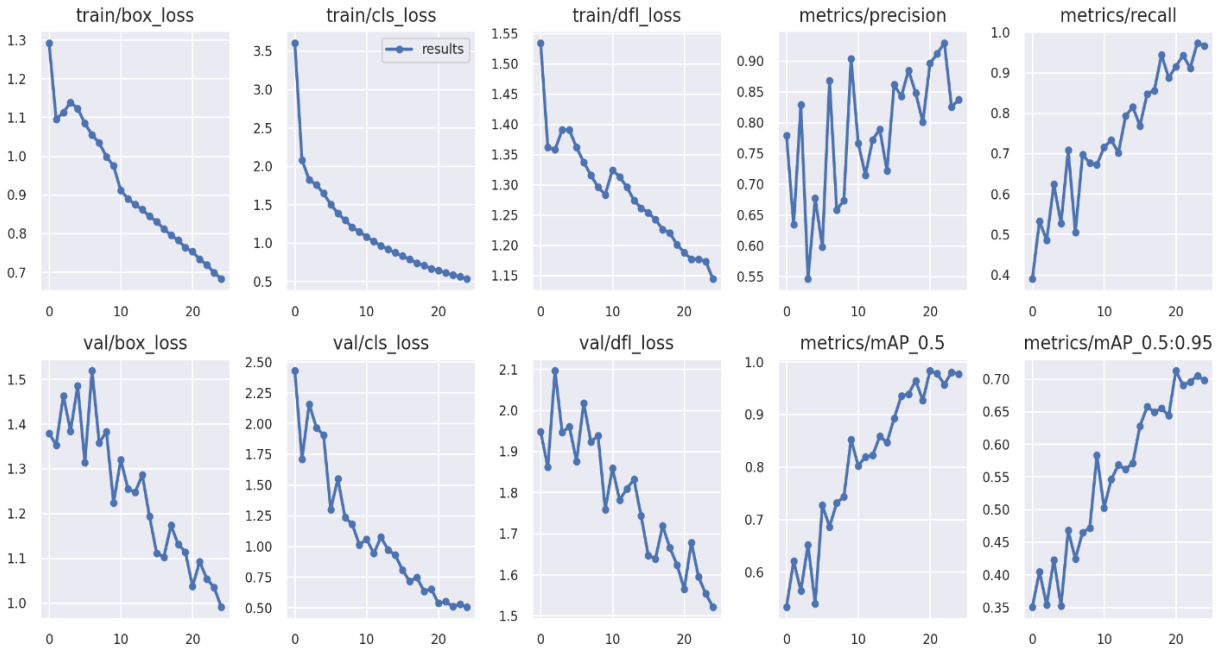


Figure 3.12

- train/box_loss and val/box_loss: These plots show the loss related to bounding box predictions during training and validation phases, respectively. The loss is decreasing over time, which indicates that the model is getting better at predicting the correct location of the bounding boxes around the objects.

- train/cls_loss and val/cls_loss: These represent the classification loss during training and validation. A decrease in classification loss suggests that the model is becoming more accurate in assigning the correct class labels to the objects within the bounding boxes.
- train/dfl_loss and val/dfl_loss: DFL could stand for a specific type of loss, potentially related to a distribution-focal loss which might be part of the model's architecture. Again, a decreasing trend indicates improvement in whichever aspect this loss is measuring.

- metrics/precision: This is a measure of the number of correct positive predictions divided by the total number of positive predictions. The fluctuation is normal, but the overall trend is upward, suggesting an improvement in precision over time.
- metrics/recall: Recall measures the number of correct positive predictions divided by the total number of actual positives. The steady increase in this plot suggests that the model is getting better at detecting all the relevant objects.
- metrics/mAP_0.5: mAP stands for mean Average Precision, and the '0.5' refers to the Intersection over Union (IoU) threshold. This metric is commonly used in object detection to evaluate model performance. The trend here is positive, showing that the model is improving in terms of precision at a 0.5 IoU threshold.
- metrics/mAP_0.5:0.95: This is the mean Average Precision calculated over different IoU thresholds from 0.5 to 0.95. This is a more stringent metric, as it requires the model to be accurate at various levels of overlap between the predicted and ground truth bounding boxes. The upward trend here is a very good sign, indicating that the model is performing better over time across a range of IoU thresholds.

Overall, based on these plots, the model seems to be learning and improving its performance as training proceeds. There's some fluctuation in validation losses and metrics, which is normal due to variations in the data batches the model sees during validation. The key is the downward trend in losses and the upward trend in precision and recall, both of which are evident here.

Chapter Four

4.1 Mobile App

4.1.1 Purpose:

This app is an interface where deaf and non-deaf people can easily understand sign language through this application.

4.1.2 Usage & Current Functionality:

To use the application is simple. They must point the camera to the sign language speaker, and the application will output the text in real time via text or record.

As a mobile application, users will be able to translate sign language anytime anywhere.

4.1.3 UI/UX Design:

I. Splash screen:

This page contains the logo of the application, which expresses the idea of the application and through it enter to Login page and Register page.

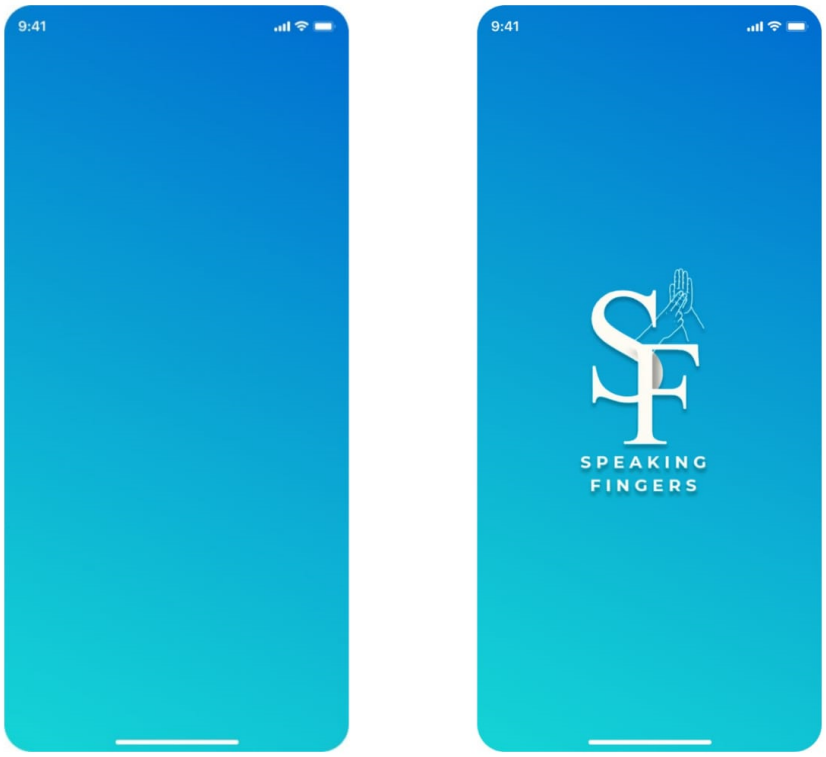


Figure 4.1.1 splash page

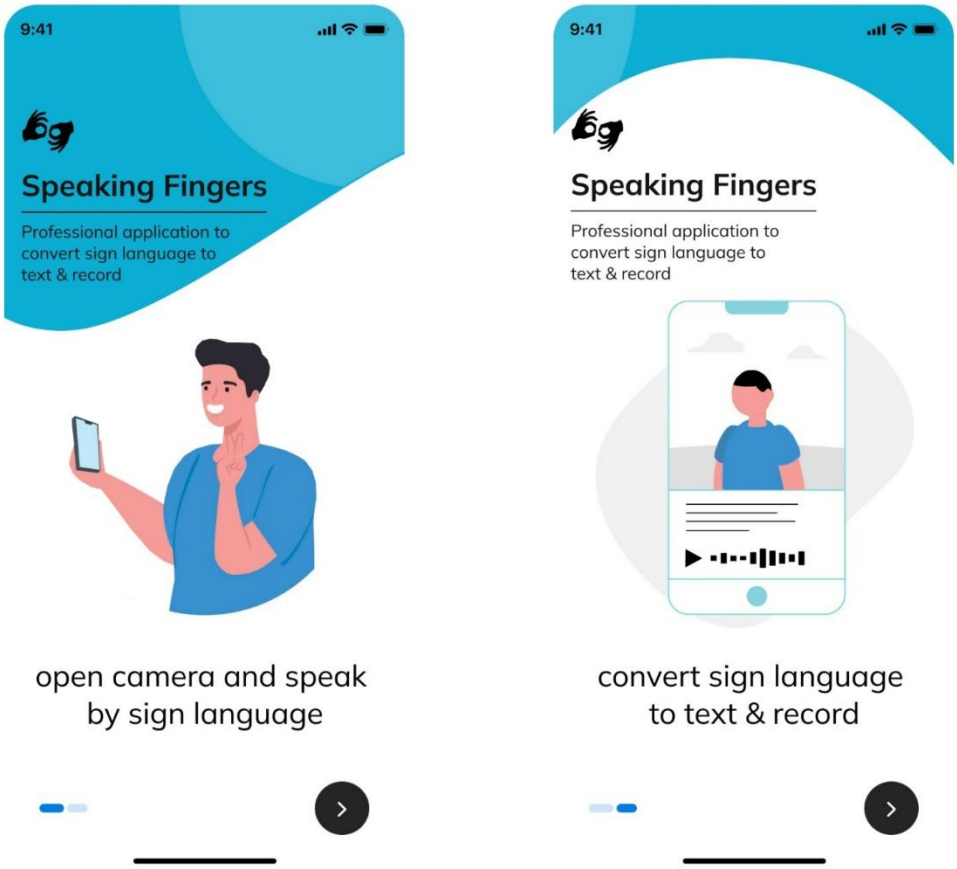


Figure 4.1.2 onboarding page

ll. Login and Register:

Users can simply use their email and password to register an account. In the backend, a new user will be created using that information. When the user logs in with their credentials, the app will receive a unique use rid that can be used to retrieve relevant information from the backend.

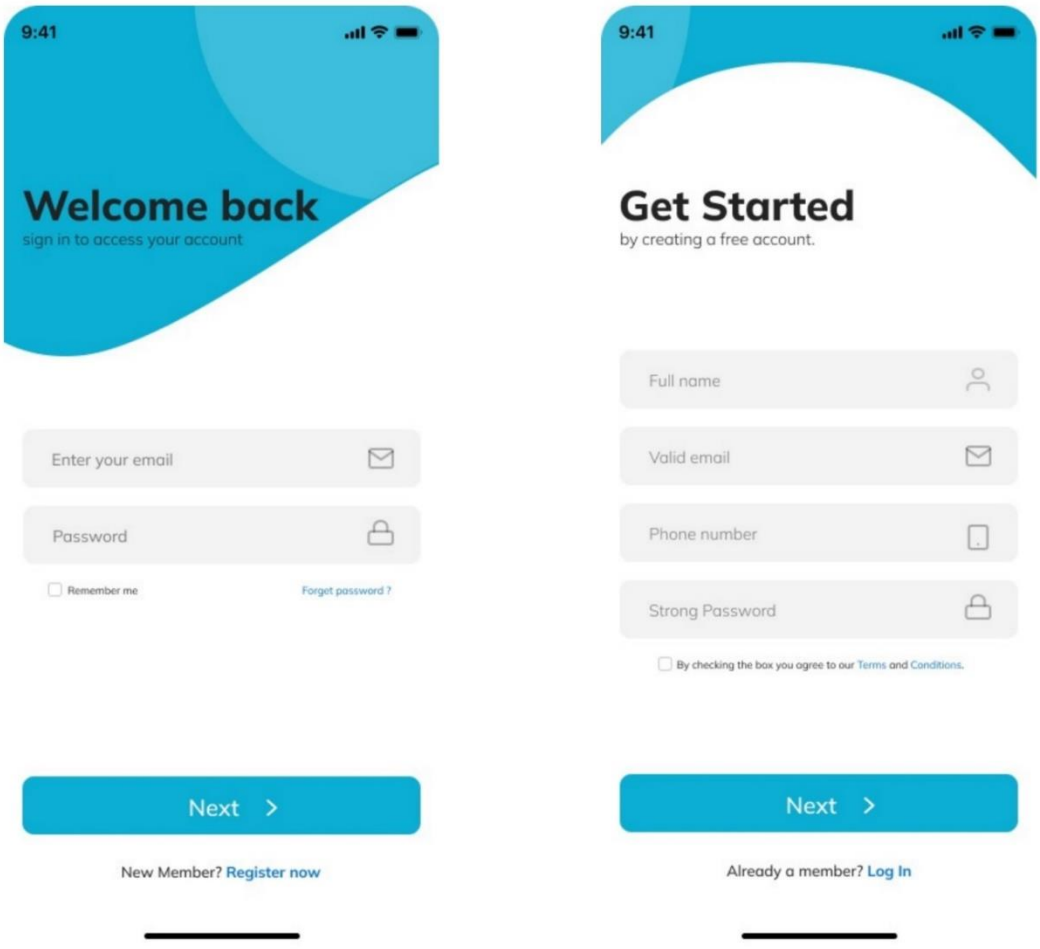


Figure 4.1.3 Login & Sign up Page.

III. Home page:

The flutter app has three options for the user on the home page (Login, Register, and Continue as Guest). If the user would like to save and view their history, they would need to authenticate with the application.

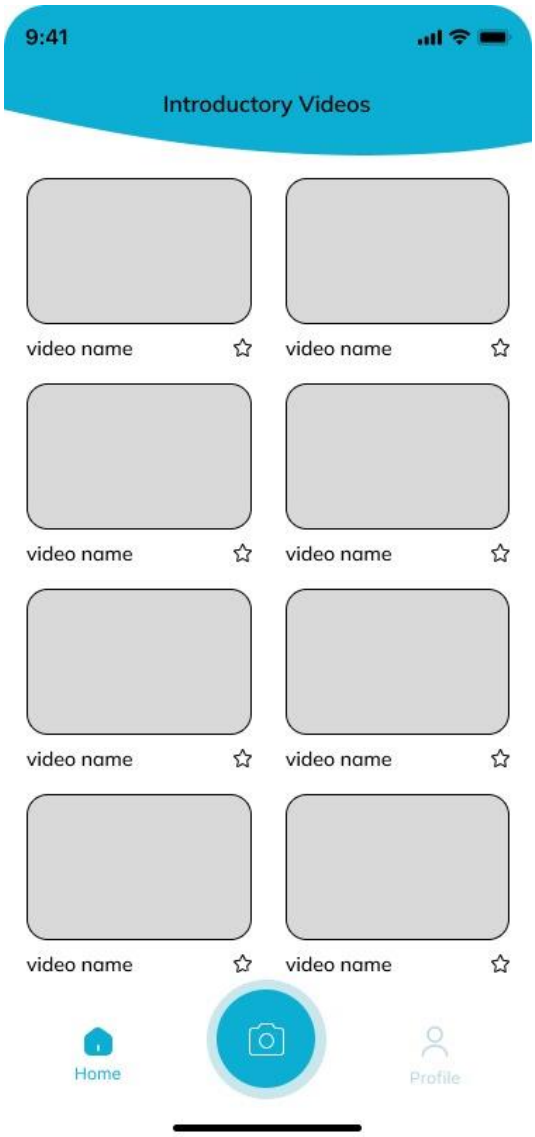


Figure 4.1.4 Home Page

IV. Settings Page:

Here contain profile and favorite video list and allow users are able to edit profile

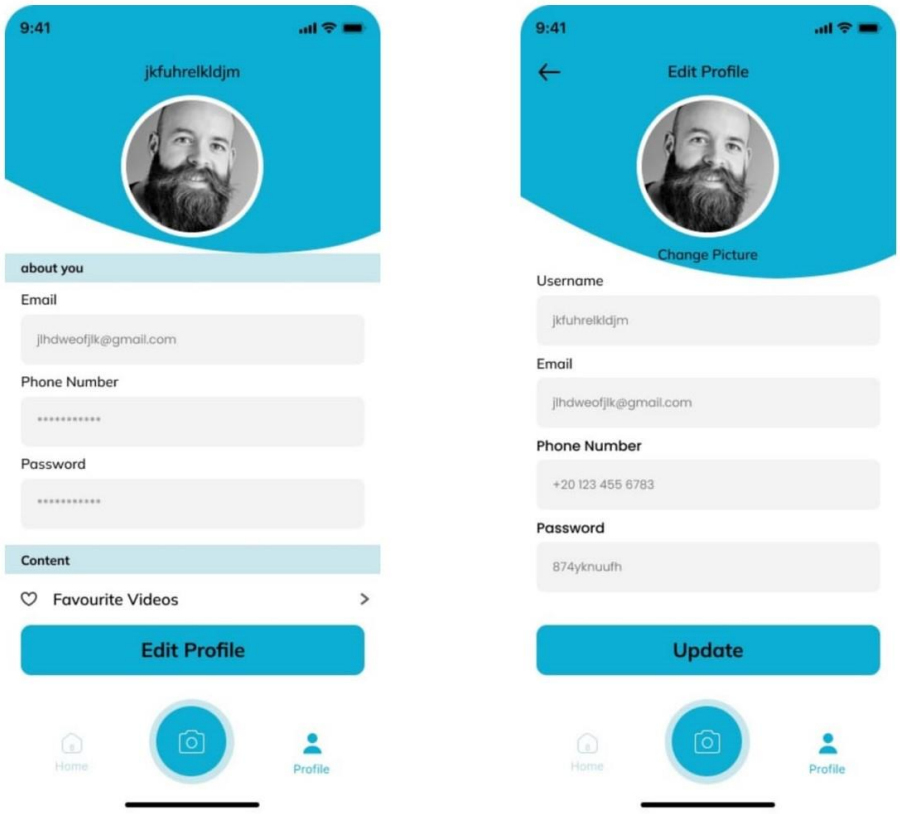


Figure 4.1.5 create and edit profile Page.

V. videos page:

This page allows to user watches educational videos for sign language. If user prefer video, can add it to his list of favorite videos.

Figure 4.1.6 Video Page

VI. Camera Page

This page allows to user pick image from Gallery to detect sign language and translate it and allow to record and convert sign language displayed by a person into text that the user can read in real time and voice that user can hear in real time . The square box will change color to provide feedback based on the confidence score and display helpful messages like "Keep Steady for accurate results".

Figure 4.1.7 Camera Page

4.1.4 Technology

I. What is Flutter?

Flutter is an open-source UI framework developed by Google that allows developers to build native-quality mobile, web, and desktop applications from a single codebase. Flutter is a valuable modern tool used to create stunning cross-platform applications that render native code on each device and OS. Flutter is compatible with Android, iOS, Linux, macOS, Windows, etc.

There are two key components of Flutter:

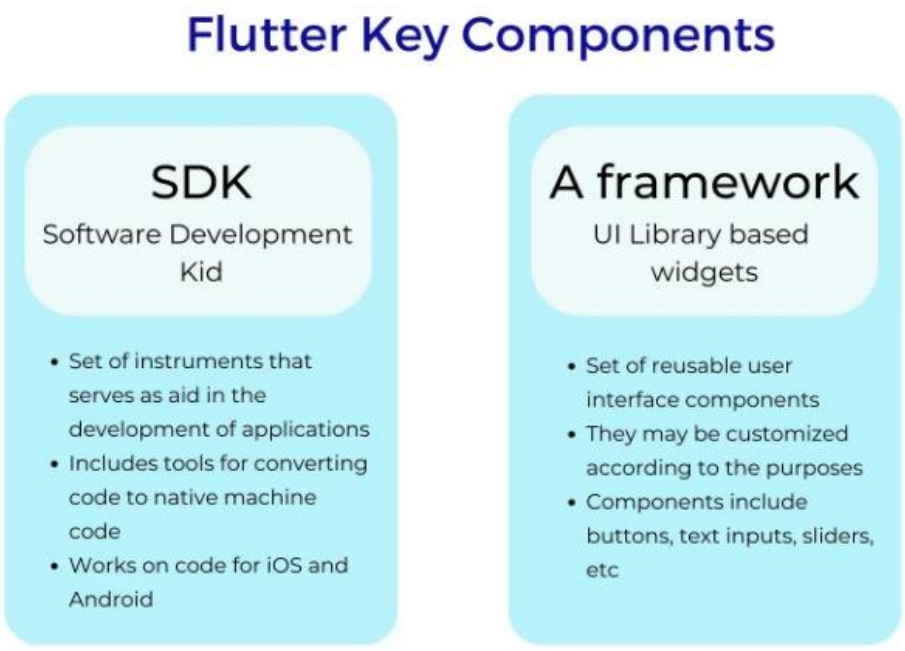


Figure 4.1.8

II. Flutter Architecture overview:

The Flutter tool incorporates three architectural layers:

- Framework: The most visible part of the Flutter technology. It is based on the Dart programming language.

- Engine: Written in C/C++, this layer provides graphics, accessibility support, text layout and other essential APIs.

- Embedder: A platform specific embedder is used to help the Flutter App to run on any OS.

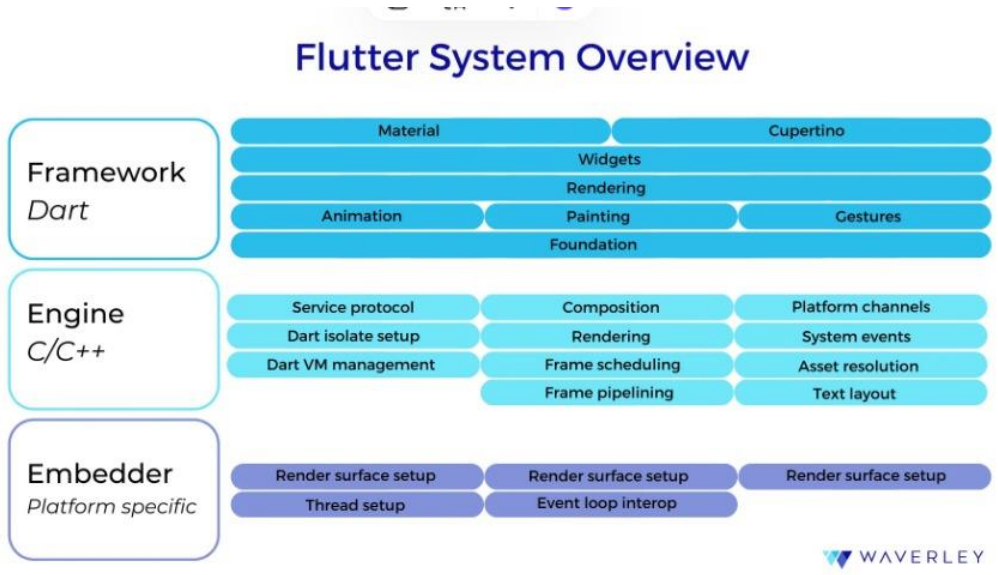


Figure 4.1.9

III. How Cross-platform Development Works with Flutter:

Flutter is a multiplatform technology that maximizes code exchange and ensures interoperability with different platforms. The developer creates an original code that becomes exclusive to each platform instead of creating native solutions.

Flutter helps companies save their development budgets, as there's no need to hire native app developers, you just need a professional Flutter engineer creating the functionality, and the tool will do the rest. It is compatible with mobile platforms (iOS and Android), Desktop platforms (Windows, Linux, MacOS), as well as the web.

As a result, the shared code approach is used to quickly develop software of any complexity and makes Flutter one of the best-known tools for fast and cost-efficient MVP development.

IV. Why Use Flutter?

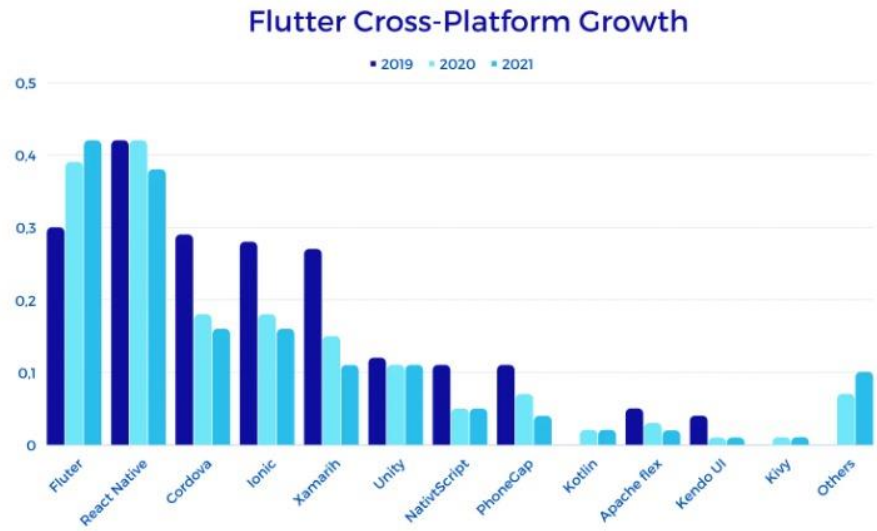


Figure 4.1.10

Ever since Flutter came out, it has drawn a large and engaged community of developers. One of the benefits of Flutter is that it is simple. This means widgets can have changes or are subject to customization with ease. Moreover, the Dart programming language and Flutter are easy to learn and start creating with, so that there never will be a shortage of specialists, any software engineer can easily and quickly switch if needed. One of Flutter's most appealing elements is UI widgets that adhere to important web application design standards.

➢ About Widgets:

Categories	Description
Accessibility	Widgets for making apps more accessible
Animation and motions	Widgets for adding animation to other widgets
Assets image and icon	Widgets for managing display images and icons
Async	Widgets for providing async functionality
Basics	Essential widgets for Flutter app development
Cupertino	Widgets designed for iOS apps
Input	Widgets for providing input functionality
Interaction Models	Widgets for managing touch events and guiding users
Layout	Supportive widgets for placing other widgets on the screen
Material Components	Widgets following material design guidelines
Painting and Effects	Widgets for applying visual changes
Scrolling	Widgets for enabling scrolling
Styling	Widgets for managing theme, size, and responsiveness
Text	Widgets for displaying and styling text

Widgets are components that appear on a screen of the Flutter app. The choice and arrangement of the widgets used to create an application is important. It has a significant impact on how the screen will look like.

4.1.5 Data Structure and Algorithm used in application:

Quicksort is a sorting algorithm based on the Divide and Conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.

The partition algorithm works by starting from the leftmost element and keeping track of the index of smaller (or equal) elements as i. While traversing, if a smaller element is found, the current element is swapped with arr[i]. Otherwise, the current element is ignored.

4.1.6 Design Pattern used in application:

1-Singleton Pattern

Singleton Pattern is probably the most widely used design pattern. It is a simple pattern, easy to understand and to use. Sometimes it is used in excess and in scenarios where it is not required. In such cases, the disadvantages of using it outweigh the advantages it brings. For this reason, the singleton pattern is sometimes considered an antipattern or pattern singleton.

2- Dependency Injection (DI)

Dependency Injection is one of the most widely used design patterns. It comes under the Software Design. In this article, we will discuss dependency injection with examples, why we need it, what is the use of dependency injection, advantages and disadvantages of dependency injection.

4.1.7 Guide to Routes Management Using GetX in Application:

Almost every app needs to navigate from one screen to another in flutter technology. But flutter makes route management complex sometimes. So today I am going to give you a tutorial on how to manage routes in GetX Flutter.

The GetX makes route management very simple and easy to navigate between screens with transitions and sending the argument on the screen. Hence, GetX package is the most liked package in pub. dev as you can see in the screenshot.

Not only route management, GetX provides other services as well.

Such as:

- State Management
- Dependency Management
- Local Storage
- GetX Server
- API Integration with GetConnect

GetX Route Management Advantages:

1. Navigate between the screen without context.
2. Can give transition animation between the screen.
3. Can show snackbar, dialog, and bottomsheet without context.
4. You can add middleware easily to protect the routes.
5. Transfer the data through [arguments] very easily.

4.1.8 Realtime Database:

The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. When you build cross-platform apps Flutter & Firebase, all of your clients can share one Realtime Database instance and automatically receive updates with the newest data.

4.1.9 Application Programming Interface (API):

An application programming interface (API) is an interface that defines interactions between multiple software applications or mixed hardware-software intermediaries. Flutter provides tools for API calling such as HTTP packages.

This package contains a set of high-level functions and classes that make it easy to consume HTTP resources. It's multi-platform (mobile, desktop, and browser) and supports multiple implementations.

More packages to build our application:

➢ Images:

- Cached_network_image: The cached network images stores and retrieves files.
- Image_picker: Flutter plugin for iOS and Android for picking images from the image library and taking new pictures with the camera.
- File_picker: A package that allows you to use a native file explorer to pick single or multiple absolute file paths, with extension filtering support.
- Flutter_svg: An SVG rendering and widget library for Flutter, which allows painting and displaying Scalable Vector Graphics 1.1 files.
- Camera: Flutter plugin for controlling the camera. Supports previewing the camera feed, capturing images and video, and streaming image buffers to Dart.

➢ fonts:

- Google_fonts (Mulish): A Flutter package to use fonts from fonts.google.com. Supports HTTP fetching, caching, and asset bundling.
- shared_preferences: Flutter plugin for reading and writing simple key-value pairs. Wraps NSUserDefaults on iOS and Shared Preferences on Android.
- Local Data Cache

4.2 Website

The sign language serves as a vital means of communication for individuals with hearing impairments. However, many deaf users face difficulty in understanding and communicating in environments lacking qualified translators or volunteers proficient in sign language. With this in mind, our application aims to provide an effective solution to this issue by automatically and instantly converting sign language into audio.

The concept of the application relies on utilizing artificial intelligence and speech analysis techniques to recognize hand gestures and convert them into audible words. The application features a simple and user-friendly interface, allowing users to record hand gestures using their smartphone camera or computer, and then converting these gestures into audible words instantly.

By providing an efficient means of converting sign language into audio, the application aims to enhance the ability of deaf individuals to understand and communicate in various social and professional settings, and provide them with an opportunity for full participation in society.

4.2.1 Project Technology:

Developing with React JS:

When it comes to developing modern web applications, choosing the right technology plays a crucial role in the project's success. ReactJS stands out as one of the best options available in the world of web application development. ReactJS boasts a range of features that make it an ideal choice for building modern web applications.

I. What Is React?

React is a framework that employs Webpack to automatically compile React, JSX, and ES6 code while handling CSS file prefixes. React is a JavaScript-based UI development library. Although React is a library rather than a language, it is widely used in web development.

The library first appeared in May 2013 and is now one of the most commonly used frontend libraries for web development.

React offers various extensions for entire application architectural support, such as Flux and React Native, beyond mere UI.

ll. React JS History

When compared to other technologies on the market. React is a new technology. Jordan Walke, a software engineer at Facebook, founded the library in 2011, giving it life. The likes of XHP, a straightforward HTML component framework for PHP, have an influence on React. React's newsfeed was its debut application in 2011. Later, Instagram picks it up and incorporates it into their platform.

III. Why React?

React's popularity today has eclipsed that of all other front-end development frameworks. Here is why:

- Easy creation of dynamic applications: React makes it easier to create dynamic web applications because it requires less coding and offers more functionality, as opposed to JavaScript, where coding often gets complex very quickly.
- Improved performance: React uses Virtual DOM, thereby creating web applications faster. Virtual DOM compares the components' previous states and updates only the items in the Real DOM that were changed, instead of updating all of the components again, as conventional web applications do.
- Reusable components: Components are the building blocks of any React application, and a single app usually consists of multiple components. These components have their logic and controls, and they can be reused throughout the application, which in turn dramatically reduces the application's development time.
- Unidirectional data flow: React follows a unidirectional data flow. This means that when designing a React app, developers often nest child components within parent components. Since the data flows in a single direction, it becomes easier to debug errors and know where a problem occurs in an application at the moment in question.

- Small learning curve: React is easy to learn, as it mostly combines basic HTML and JavaScript concepts with some beneficial additions. Still, as is the case with other tools and frameworks, you have to spend some time to get a proper understanding of React's library.

- It can be used for the development of both web and mobile apps: We already know that React is used for the development of web applications, but that's not all it can do. There is a framework called React Native, derived from React itself, that is hugely popular and is used for creating beautiful mobile applications. So, in reality, react can be used for making both web and mobile applications.
- Dedicated tools for easy debugging: Facebook has released a Chrome extension that can be used to debug React applications. This makes the process of debugging React web applications faster and easier.

The above reasons more than justify the popularity of the React library and why it is being adopted by a large number of organizations and businesses. Now let's familiarize ourselves with React's features.

IV. Features of React

React offers some outstanding features that make it the most widely adopted library for frontend app development. Here is the list of those salient features.

JSX

Figure 4.2.1

JSX is a JavaScript syntactic extension. It's a term used in React to describe how the user interface should seem. You can write HTML structures in the same file as JavaScript code by utilizing JSX.

```
const name = 'Simplilearn';
const greet = <h1>Hello, {name}</h1>;
```

Figure 4.2.2

The above code shows how JSX is implemented in React. It is neither a string nor HTML. Instead, it embeds HTML into JavaScript code.

V. Components in React:

Components are the building blocks that comprise a React application representing a part of the user interface.

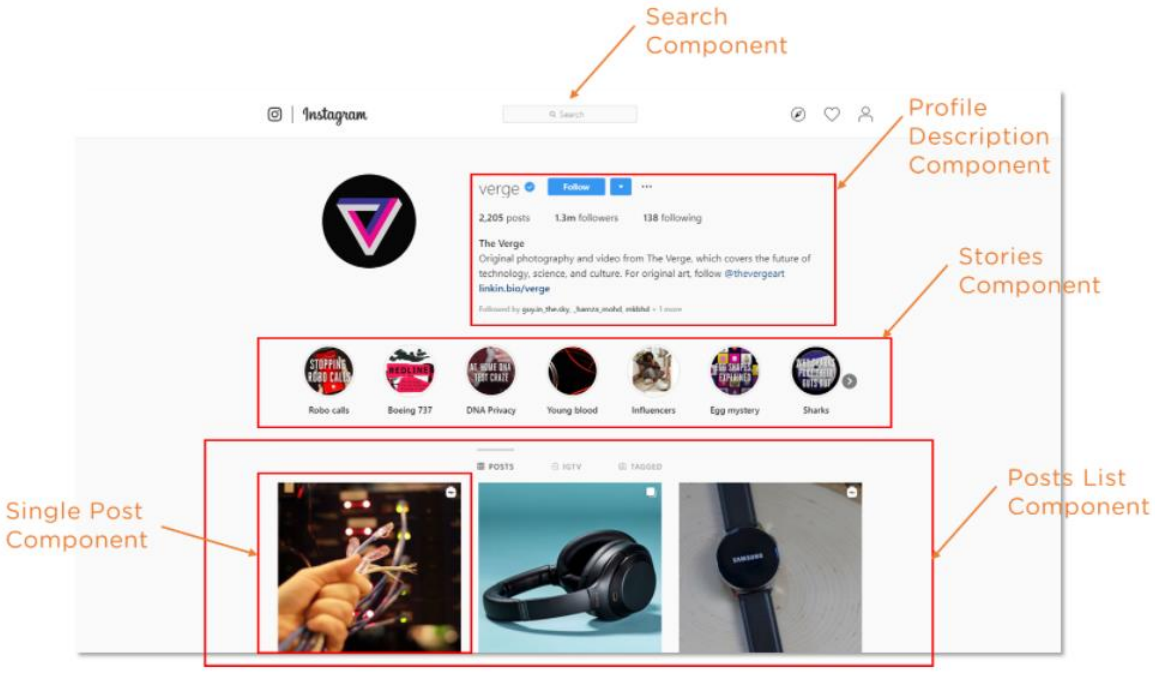


Figure 4.2.3

React separates the user interface into numerous components, making debugging more accessible, and each component has its own set of properties and functions.

Here are some of the features of Components:

- Re-usability - A component used in one area of the application can be reused in another area. This helps speed up the development process.
- Nested Components - A component can contain several other components.
- Render method - In its minimal form, a component must define a render method that specifies how the component renders to the DOM.
- Passing properties - A component can also receive props. These are properties passed by its parent to specify values.

4.2.2 Industry Trends:

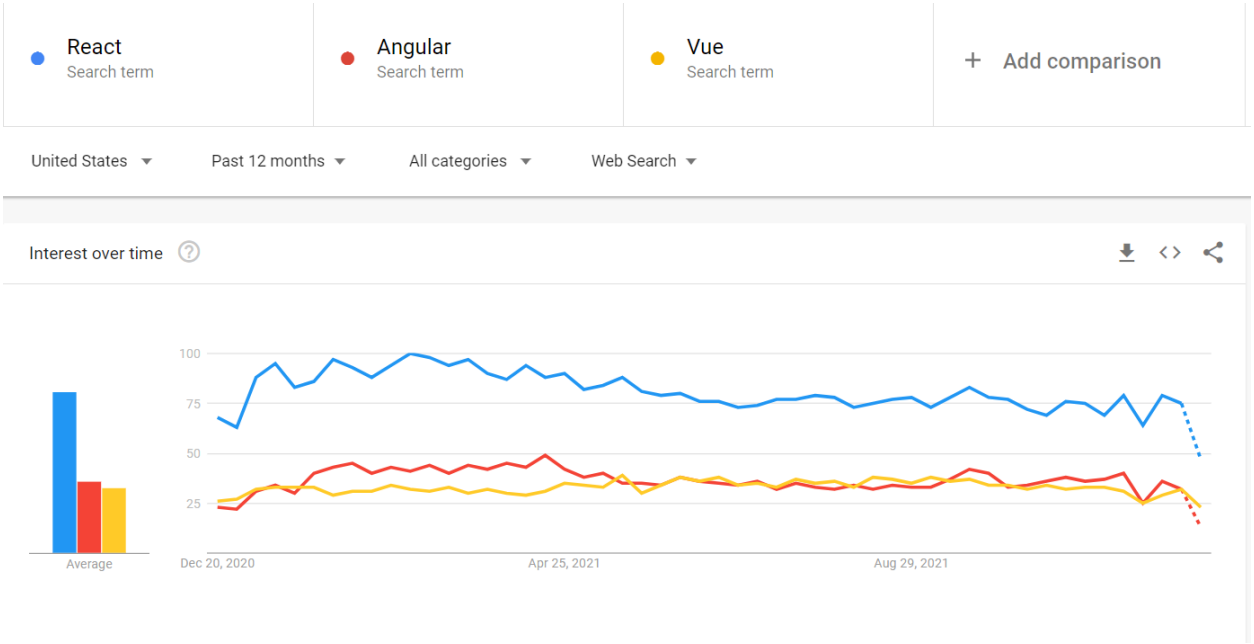


Figure 4.2.4

React is more popularly adopted by developers across the globe as compared to other frameworks and libraries.

- The average salary for an entry-level React developer in the USA is about 87,000USD per annum.
- The average salary for an entry-level React developer in India is about 6,50,000INR per annum.

3.2.3 Using APIs in the Project:

After discussing the project's technology and reviewing how ReactJS is used in developing the application, we will now talk about how we interact with Application Programming Interfaces (APIs) in our project. The application relies on APIs to retrieve necessary data and interact with the server to accomplish various functions.

3.2.4 Calling APIs with Axios:

In our project, we leverage Axios, a popular and efficient JavaScript library, to communicate with backend APIs. Axios simplifies the process of making HTTP requests from the client-side, offering a straightforward and easy-to-use API for handling asynchronous operations.

When integrating APIs into our ReactJS application, the following steps are typically involved:

- Installing Axios: First, we install Axios as a dependency in our project. We can do this using npm or yarn by running the command `npm install axios` or `yarn add axios` in the terminal.

- Importing Axios: Once Axios is installed, we import it into the relevant files where we need to make API calls. We usually import Axios at the beginning of the file where we plan to use it.

```
import axios from 'axios';
```

Figure 4.2.5

- **Making API Requests:** With Axios imported, we can start making API requests. We use Axios methods such as axios.get (), axios.post (), axios.put (), and axios.delete () to send HTTP requests to the backend server.

```
axios.get('https://api.example.com/data')
  .then(response => {
    // Handle successful response
    console.log(response.data);
  })
  .catch(error => {
    // Handle error
    console.error('Error fetching data:', error);
  });
```

Figure 4.2.6

By following these steps and leveraging Axios in our ReactJS application, we can seamlessly integrate with backend APIs, retrieve data, and interact with the server, enhancing the functionality and usability of our project.

3.2.5 App Features:

I. Home Page:

- The Home Page serves as the gateway to our project, offering an introduction to its purpose and guiding users to various project components. It acts as the central hub where users can access essential project features and navigate seamlessly to different sections.

II. Authentication System:

- Offer a secure and reliable login process using user credentials.
- Allow users to reset their password if forgotten.

III. Sign Language Learning Videos:

- Provide a wide range of specialized educational lessons and videos for learning sign language.
- Present lessons in an organized and structured manner based on difficulty levels and various topics.

IV. Add to Favorites:

- Enable users to create a list of their favorite videos for later reference.

V. Audio/Text Translation for Sign Language:

The application provides the user with the ability to open the camera to communicate using sign language, where they can convert their hand gestures into audible speech or written text. By utilizing image and sound recognition technologies, users can communicate easily and effectively with others, even if they are not familiar with sign language. This approach serves as an innovative means to achieve impactful communication and overcome language barriers, thus enhancing understanding and effective communication among all users of the application.

References:

1. https://www.researchgate.net/publication/224197735_ArSLAT_Ara bic_Si gn_Language_Alphabets_Translator#pf2
2. https://data.mendeley.com/datasets/y7pckrw6z2/1.
3. https://en.wikipedia.org/wiki/Sign_language
4. https://www.sciencedirect.com/science/article/pii/S1877050917320720
5. https://www.tensorflow.org/api_docs/python/tf/all_symbols.