Sign Away: Automating Translation of Sign Language Using AI Models like SVM, CNN, RNN, YOLO, and Transformer

1. Introduction/Problem Statement

In today's interconnected world, the ability to communicate effectively is paramount. Yet, for millions of deaf or hard-of-hearing individuals, a significant barrier stands in the way: the lack of understanding and accessibility to sign language. According to the World Health Organization (WHO), by 2050, over 700 million people - approximately 10% of the global population - will grapple with disabling hearing loss (World Health Organization, 2023). In response to this pressing challenge, we strive to bridge the communication chasm between sign language users and those unfamiliar with its intricacies. Mastering sign language presents a formidable challenge due to its complex and nuanced nature. Therefore, instead of imposing the onus on oral communicators to learn sign language, we propose leveraging on current technology; creating a model designed to accurately recognize and translate American Sign Language into text. By harnessing the power of machine learning, we hope to foster inclusivity and understanding, facilitating seamless communication and interaction across diverse communities.

Motivation

The imperative for effective communication within the deaf and hard-of-hearing community extends far beyond mere accessibility; it is foundational to their inclusive education, meaningful employment opportunities, and seamless integration into social spheres.

Studies have highlighted the profound impacts of effective communication on individuals' quality of life and societal integration. A research conducted in 2020 indicates that individuals with hearing impairments face higher rates of social isolation, unemployment, and mental health challenges compared to their hearing counterparts (Shukla et al., 2020). By unlocking the potential to translate intricate hand signs into readily understandable words, this project serves as a vital conduit for bridging linguistic divides. It not only empowers individuals within the deaf and hard-of-hearing community but also enriches the broader population by fostering empathy, awareness, and a deeper understanding of diverse communication modalities. Furthermore, inaccessibility to sign language interpretation services in crucial settings such as healthcare exacerbates these disparities, limiting opportunities for meaningful participation and engagement (Barnett et al, 2011).

In an era defined by rapid technological advancements, leveraging these innovations to facilitate sign language translation represents a monumental leap forward in inclusivity and accessibility. By embracing this transformative approach, we not only enhance the quality of

life for millions but also cultivate a fertile ground for innovation and collaboration. Ultimately, investing in initiatives that champion inclusive communication is not merely an ethical obligation; it is an indispensable strategic endeavour that propels us towards a more equitable and prosperous future, where every voice is heard and valued.

2. Literature review

1. K-nearest neighbour (KNN)

Alphabet Sign Language Recognition Using K-Nearest Neighbour Optimization (Utaminingrum et al, 2019)

Summary

This review explores the effectiveness of integrating the Simple Multi Attribute Rating Technique (SMART) with K-Nearest Neighbor (K-NN) classification for alphabet sign language recognition. Through the implementation of SMART for weighting and K-NN for classification, the research aims to optimize the recognition process. The methodology involves hand detection, sign grouping and weighting using SMART, feature extraction, and image classification through K-NN, achieving an average accuracy between 94% and 96% under varying light conditions.

Methodology

Dataset Preparation: The research utilized hand gesture images, resized to 300x400 pixels, and applied skin detection techniques for preprocessing.

SMART Weighting and K-NN Classification: The study combined SMART weighting to optimize the K-NN classification process, dividing alphabet signs into three groups based on hand shape and using Euclidean distance for classification.

Feature Extraction: Statistical data from the distance between the center of the image and the edge of the hand was used for feature extraction.

Evaluation: The method's effectiveness was assessed under different lighting conditions, demonstrating that lighting significantly impacts the accuracy of sign language recognition.

Results

The combined approach of SMART weighting and K-NN classification achieved high accuracy rates in alphabet sign language recognition, with average accuracies of 94%, 95%, and 96% under dark, normal, and bright lighting conditions, respectively.

Conclusion

The study concluded that the integration of SMART weighting with K-NN classification significantly improves the accuracy of alphabet sign language recognition. The research highlights the importance of lighting conditions in the recognition process and suggests that the proposed method can effectively enhance communication for the disabled through improved hand gesture recognition.

2. Support Vector Machine (SVM)

American Sign Language recognition using Support Vector Machine and Convolutional Neural Network (Jain et al., 2021)

Summary

This review discusses use of Support Vector Machines (SVM) in identifying sign language, with the use of MNIST Kaggle's dataset, consisting of 25 classes representing different letters in American Sign Language (ASL). The SVM utilized 4 different kernels for finding the most accurate solution, proposing the recognition of ASL.

Methodology

Dataset Preparation

The dataset consisting of American Sign Language images from MNIST Kaggle was prepared. This dataset includes 25 classes, excluding letters "j" and "z" which are dynamic signs, corresponding to different letters in ASL, each represented by 28x28 pixel grayscale images.

SVM Model Selection

The study explores SVM for ASL sign recognition, employing four different kernels:

1) Linear: Suitable for linearly separable data, this kernel was tested to see how well ASL signs could be classified without transforming data into a higher dimensional space.
2) Polynomial ('poly'): This kernel transforms the data into a higher-dimensional space where a polynomial boundary separates the classes, allowing for the classification of data that is not linearly separable in the original space.
3) Radial Basis Function ('rbf'): A popular choice for non-linear data, the RBF kernel can handle cases where the relationship between class labels and attributes is non-linear.
4) Sigmoid: Mimicking the behavior of neural networks, this kernel was tested to evaluate its effectiveness in sign language recognition.

SVM Training

The SVM models were trained on the dataset with the objective of classifying the ASL signs into their respective categories. Training involves adjusting the model parameters to minimise errors in classification. For SVM, key parameters include C, Gamma, Cache Size and Probability.

Results

After training, the models' performance was evaluated based on their accuracy in classifying the test data.Through this metric, the study aimed to identify which kernel and parameter settings provided the best results for ASL sign recognition. Linear kernel had an accuracy of 80.53%, Poly kernel had an accuracy of 81.49%, Rbf kernel had an accuracy of 64.09%, Sigmoid kernel had an accuracy of 13.78%.

Conclusion

The study found that the SVM with a 'poly' kernel achieved the highest accuracy of 81.49% among the SVM models, indicating that this kernel's ability to handle non-linear data transformations was most effective for the ASL recognition task within the SVM framework.

3. Convolutional Neural Network (CNN)

A Review on Sign Language Recognition Using CNN (Ugale et al., 2023)

Summary

This review discusses use of Convolutional Neural Networks (CNNs) in identifying sign language, offering research findings on translating sign language gestures into text or speech to improve communication for hearing impaired individuals. Various CNN approaches and models have shown notable progress in accuracy and efficiency, contributing to improvements in sign language recognition systems.

Methodology

- Data acquisition and preprocessing: Researchers utilized custom datasets, comprising images and videos of sign language gestures, to train and test CNN models. These datasets included a wide range of signs from different sign languages such as American Sign Language (ASL), BritishSign Language (BSL), and Indian Sign Language (ISL).
- Studies employed various CNN configurations, incorporating layers such as convolutional, pooling, dropout, and fully connected layers using activation functions like ReLU (Rectified Linear Unit). There were also other studies that integrated advanced techniques like Hierarchical Attention Networks and Long Short-Term Memory (LSTM) units to improve recognition accuracy.
- Training and optimization: Models were trained using techniques like supervised learning, transfer learning, and stochastic gradient descent with varying learning rates, batch sizes, and epochs to optimize performance.

Results

- Recognition accuracy varied across studies, with signer-dependent models achieving between 69% to 98% accuracy, and an average of 88.8%. Signer-independent models demonstrated high accuracy, with one study achieving up to 99.93% training accuracy and 98.64% testing and validation accuracy.
- Integration of LSTM units with CNNs resulted in high accuracy levels of up to 99% on training sets and 93% with real-world testing.
- Challenges such as boundary identification in continuous sign language recognition and differentiation among similar signs were addressed through techniques like transfer learning and the use of more diverse samples in datasets.

Conclusion

CNN-based approaches for sign language recognition have shown promising results, with continuous advancements in model architecture, training methodologies, and dataset quality contributing to improved accuracy and efficiency. While there are still challenges that exist such as finding the optimal combination of hyperparameters and the risk of overfitting, and particularly in recognizing nuances and variations in sign language gestures, ongoing research and technological developments are likely to overcome these hurdles, further enhancing communication possibilities for individuals with hearing or speech impairments.

4. Detection Transformer (DETR)

A Review on Sign Language Recognition from Digital Videos Using Feature Pyramid Network with Detection Transformer (Liu et al., 2023)

Summary

In the article Sign Language Recognition from Digital Videos Using Feature Pyramid Network with Detection Transformer by Liu, Y. et al, 2023, they proposed a Vision Transformer-based sign language recognition method called Detection Transformer (DETR). The DETR method combines the use of a deep learning model called ResNet152 and a Feature Pyramid Network (FPN). The model uses 12 video fragments depicting nine words: “Love”, “Good”, “You”, “Meet”, “Yes”, “No”, “Please”, “Name”, and “My”. With the 12 video fragments, 7192 images were gathered with 5000 images used for training and the remaining 2192 images used for testing.

Methodology

The Detection Transformer was used to solve the problems of RNN models where it is unable to perform parallel computation. Transformer has the encoder-decoder framework which uses Attention Mechanism to parallelize computation during learning and inference.

Convolutional Neural Network was adopted as the backbone of the proposed model in order to extract features from images. A typical problem experienced by CNN is the vanishing gradient problem where the changes in weights are so small, resulting in minimal improvement to the model. To address this, ResNet152, a variant of the Residual-Network architecture was used as the backbone. Output of the ResNet152 after each stage will be passed to the Feature Pyramid Network to enhance feature maps for better object detection.

Results

The DETR model was evaluated based on its Average Precision (AP) value and compared to other models using ResNet18, ResNet34, ResNet50, ResNet101, and You Only Look Once (YOLO) models - YOLOv3, YOLOv4, YOLOv5, YOLOX. From the results, their proposed DETR model with ResNet152 and FPN had the highest AP with the value at 96.46% compared to the second highest at 93.61% and the lowest at 83.63%.

The proposed model in the article was able to achieve the highest prediction accuracy for all classes compared to other models.

Conclusion

They concluded that their proposed method achieved a superior performance in sign language recognition and also highlighted some limitations of their model - the datasets they used are limited to only nine words and that the model complexity is higher than other models such as the recurrent neural network (RNN). Those limitations can be compensated by expanding the dataset and the use of a more powerful GPU.

It also showcased the potential of Transformer and suggested ways to increase the accuracy through the addition of more convolutional layers and increasing feature maps.

5. Recurrent Neural Networks (RNN)

American Sign Language Recognition and Training Method with Recurrent Neural Network (Lee et al., 2021)

Summary

They proposed an ASL learning application with a real-time sign recognition system that could recognize 26 alphabets in American Sign Language with a high accuracy rate.The feature extraction-based classification was conducted using a long-short term memory recurrent neural network (LSTM-RNN) combined with the k-nearest neighbor (kNN) approach.

Methodology

In regards to the motion capture, a leap motion controller was used for real-time data acquisition. Sphere radius, angles between fingers and distance between finger positions were extracted as input for the classification model. The real-time sign recognition system consists of three stages: data acquisition, feature extraction, and classification. Initially, data is acquired directly from the leap motion API. Subsequently, certain data undergoes additional processing to extract features. Once the data is structured, it can be fed into the pre-trained classification model for real-time recognition. Gestures are then categorized into one of the 26 classes. 30 features are extracted from 200 frames to account for variations in hand gesture motion and the characteristics of American Sign Language (ASL), which include both static and dynamic signs. After collecting every set of 200 frames, they are forwarded to the RNN classifier for categorization. The classification outcomes are transmitted for storage in a CSV file.

The proposed model would consist of three layers after the input layer; LSTM, Lambda and Dense.The LSTM layer is selected due to its capability for handling data in a long-time frame.The Lambda layer in the middle would be a K-means clustering layer. The algorithm would assign N data points into 1 of the K clusters. K-mean clustering is opted for the second layer since it is an efficient clustering method for handling multi-class classification.The Dense layer uses logistic regression as the output function, in which the log odd ratios calculated would be probabilities of each class in multiclass classification, transforming the group predictions into class probabilities for output

Results

In regards to evaluation, a categorical cross-entropy is measured on the test set to evaluate the accuracy of the model in the predictions. Here, the output vector represents a probability

distribution, and cross-entropy serves as a measure of the gap between the network's prediction for the distribution outcome and the "true answer" for that distribution.The 26-class confusion matrix is further produced into another matrix containing true positive (TP), true negative (TN), false positive (FP) and false negative (FN), which are used for generating accuracy (ACC), sensitivity (Se) and specificity (Sp) for each class.

As for the results, the accuracy of the model is estimated to be 91.8%, averaged across 5 trials. Per-class accuracy and specificity for the model were calculated to be over 98%. When compared to other known methods in ASL classifications (LSTM, SVM and RNN), the proposed models outperforms them in the numerical analysis

Conclusion

The effectiveness of RNN in sign language recognition is highlighted in this study. RNN would be especially helpful for taking in the dynamic signs of ASL, which require the handling of sequences of input.

As for the limitations of the proposed model, tuning hyperparameters such as learning rates, hidden layer sizes, and sequence lengths can be challenging, and suboptimal choices may lead to poor performance. Overfitting can lead to poor generalization performance on unseen data, limiting the applicability of the model in real-world scenarios. A limitation unique to this study is the position and angle of the leap motion controller which affects the accuracy of the model.

Despite limitations, RNNs are valuable for sign language recognition due to their adeptness at modelling sequential data. They capture temporal dependencies crucial for understanding gestures. Advanced architectures like LSTM and GRU mitigate challenges like vanishing gradients. Another viable option is combining with other architecture such as CNNs.

3. Dataset

The initial dataset chosen during the proposal was not suitable for our project as each class contains the same image, meaning there is no variation in each class (images are duplicated). This resulted in our models overfitting and did not generalize well with unseen data.

1. Roboflow American Sign Language

Link - https://public.roboflow.com/object-detection/american-sign-language-letters/

The American Sign Language Letters dataset is an object detection dataset of each ASL letter with a bounding box. This dataset was curated and released as a public dataset by David Lee, a data scientist focused on accessibility (American Sign Language Letters Object Detection Dataset, 2022). It contains images and bounding box labels for all 26 English alphabets with 1512 training sets and 144 test sets in total.

This dataset was chosen because the images contain a variety of backgrounds which can help our model to generalise well to different scenarios and places. Another reason was that this dataset contains the bounding box labels for each image and thus would save us time in manually labelling each image we have before training our models. Hence, the YOLO model uses this dataset for training.

However, we do note that there is a lack of variety for different skin tones and the majority of the hand signs are at around the same fixed distance which may lead our models to not perform as well when detecting smaller hand signs further away in the image. This is one of the primary motivations to include another dataset.

2. Synthetic ASL Alphabet

Link - https://www.kaggle.com/datasets/lexset/synthetic-asl-alphabet?select=Test_Alphabet

The second motivation was because we felt that the roboflow dataset did not have sufficient data for our deep learning models, hence, we combined that dataset with the Synthetic ASL Alphabet dataset(Arikeri, 2021) to include wider diversity as well as get a larger dataset.

The Synthetic ASL Alphabet dataset contains 27 thousands images, with 900 data in each class for training and 100 in class for testing. It has 27 classes which includes alphabets A-Z and Blank. We did not use the blank class for our model training.

Most of the images were synthetically created using Seahaven. This dataset contains hand signs from different angle as well as under different lighting conditions.

For SVM, CNN, RNN and Vision Transformer (ViT), we combined both datasets for training to improve the performance.

4. Methodology

Our solution is a live-video sign language interpreter where the hand sign will be bound by a box with the class label on the top left of the box. You Only Look Once (YOLO) model will be used as the object detection model to predict the bounding box coordinates. For class label prediction, we will use 5 classifiers. They are SVM, CNN, RNN, YOLO and ViT.

4.1. Data Pre-Processing

Figure 1

Resizing images

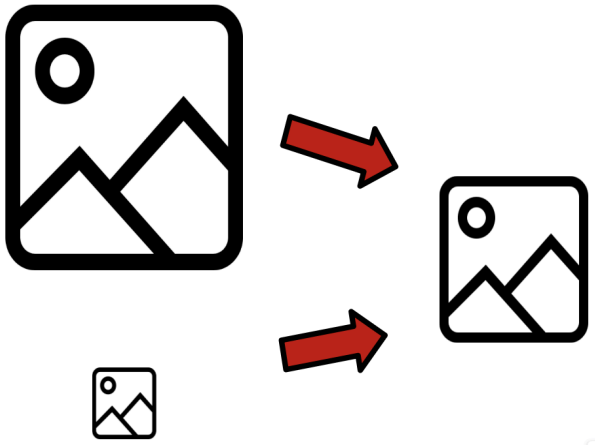
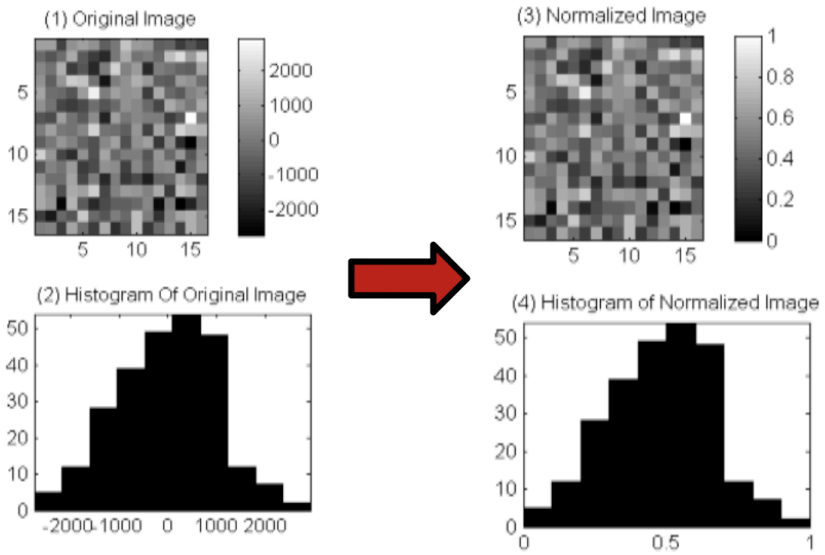


Figure 2

Normalizing images



Note. Normalize images to change the pixel range to 0 - 1. From *Image normalization, image range and image scaling for different stack of images* [Diagram], by Steve, 2012, Stack Overflow (https://stackoverflow.com/questions/11178925/image-normalization-image-range-and-image-scaling-for-different-stack-of-images).

The first step of any machine learning task is to perform data pre-processing. For our project, this is relatively short as we are dealing with image data. For images, 2 pre-processing steps are crucial. They are image resizing (Figure 1) and normalization (Figure 2). As we use a combined dataset from different sources, the size of the images would be different. To ensure consistency, resizing is done. Normalization is important as it helps normalize the pixel range from (0 - 256) to (0 - 1). Normalization can contribute to improved optimization efficiency, faster convergence, and potentially better performance of the model on unseen data.

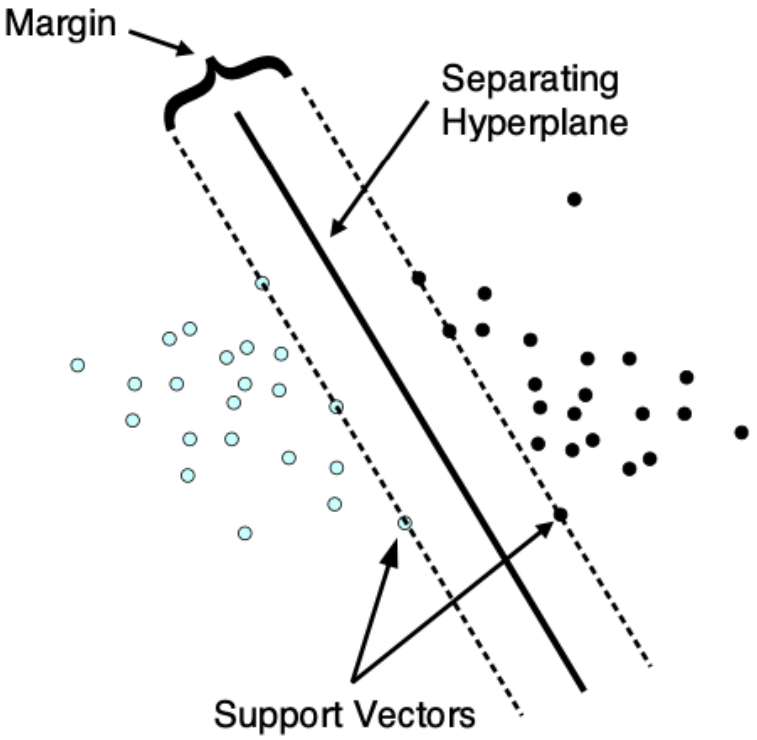
Train test split is usually required, however, our dataset has a separate folder for training and testing. Hence, we did not do a split.

4.2 Support Vector Machine (SVM)

SVM, first introduced by Vladimir N. Vapnik and his colleagues, is one of the many capable supervised machine learning algorithms used for classification tasks (Meyer & Wien, 2001). The practical use case of SVMs for classification tasks ranges from handwritten character classification to biometrics and imagery (Ma & Guo, 2014).

Figure 3

Support Vector Machine Classification



Note. Support Vector Machine Classification (linear separable case). From Support Vector Machines [Diagram], by Meyer, D., & Wien, F. T., 2001, (https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4850faab0aab5c4b40fcdd5a77d9e7626a163db5).

How Support Vector Machine Works?

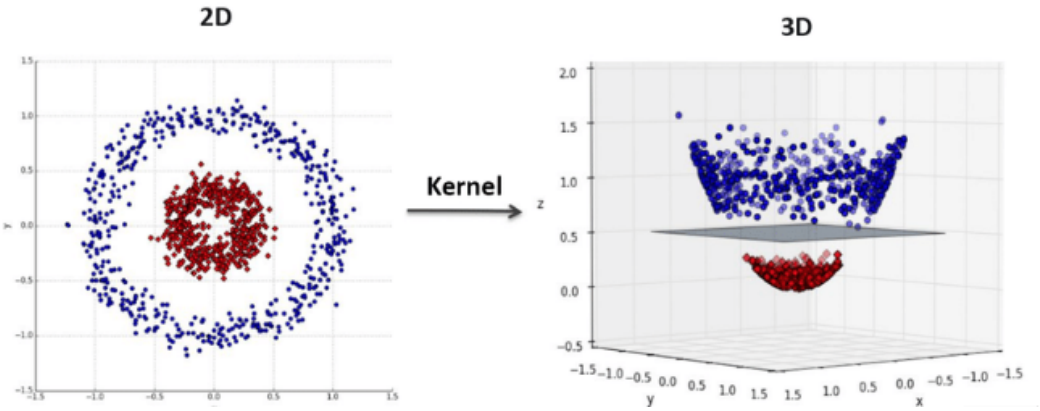
With reference to Figure 3 above, how the SVM classifier works is that it tries to find a hyperplane that can separate the 2 classes linearly. Its objective is to find the optimal hyperplane where the hyperplane is located as far away as possible from the closest data point from each class (Fletcher, 2009). SVM first locates two data points which are close to each other but belong to different classes. These two points are known as the support vectors. The initial hyperplane is then positioned in between these 2 data points.

Adjustments (moving or rotation) to the hyperplane is based on the distance between the hyperplane and the support vectors as well as the misclassification error of all data points.

Addressing Limitations of Classification in a 2-Dimensional Space (Kernel Trick)

Figure 4

Support Vector Machine Kernel Trick



Note. Support Vector Machine Kernel Trick. From *The transformer blueprint: A holistic guide to the transformer neural network architecture* [Diagram], by Nyandwi, J, 2023, deeprevision (https://medium.com/@Suraj_Yadav/what-is-kernel-trick-in-svm-interview-questions-related-to-kernel-trick-97674401c48d).

For tasks that have more than two classes and the data is not linearly separable in a two-dimensional space, we can transform the data into higher dimensional space for SVM to find the optimal hyperplane (Figure 4). However, computation to find the optimal hyperplane would then be computationally expensive because there are more support vectors and it would be equivalent to the number of classes in the current task (Wilimitis, 2018). To overcome this, the kernel trick can be used to implicitly map these data into higher dimensional space without actually transforming each data point.

Types of SVM Kernels

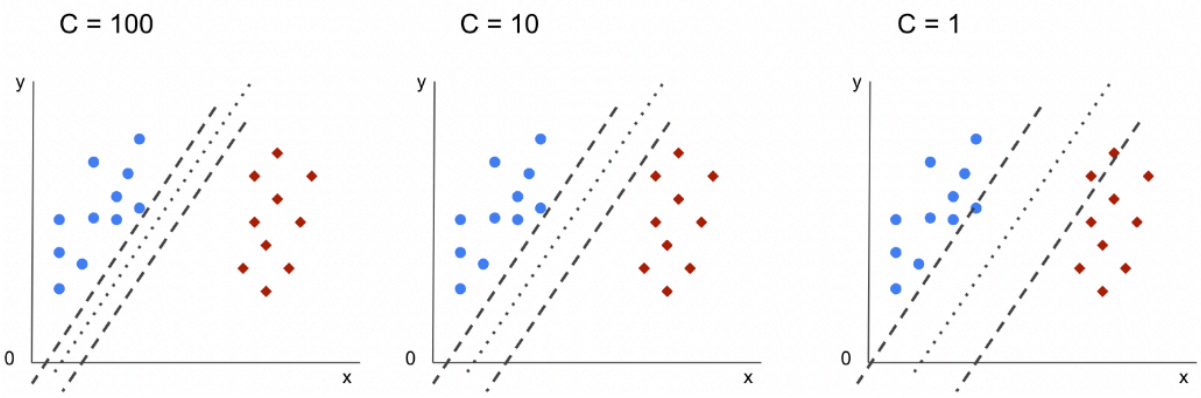
Linear kernel, polynomial kernel and Gaussian kernel (also known as the radial basis function, RBF, kernel) are some commonly used kernels to implement the SVM kernel trick (Yadav, 2023). The **linear kernel** is used when the data is roughly linearly separable, whereas the **polynomial kernel** is used when the data contains dependencies between the features. The **gaussian (RBF) kernel** is used when the data contains complicated areas of overlap and the data cannot be separated by a linear or polynomial decision boundary (Yadav, 2023).

SVM Hyperparameters

Beyond choosing suitable kernels for SVM, it is also important to tune the hyperparameter of the SVM to find the best model configuration. Regularisation parameter (C), the gamma value and the degree of the polynomial kernel are some of the crucial hyperparameters that will influence the performance of SVM models (Sampaio, 2023).

Figure 5

SVM Regularisation Parameter (C)

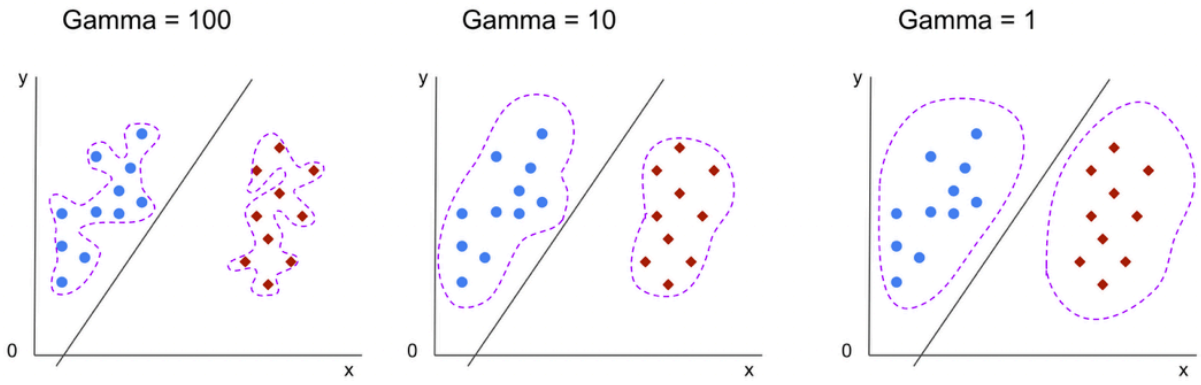


Note. Support Vector Machine Regularisation Parameter (C). From *Understanding SVM Hyperparameters* [Diagram], by Sampaio, C, 2023, StackAbuse (https://stackabuse.com/understanding-svm-hyperparameters/).

The regularisation parameter (C) is applicable to any SVM kernels. It controls the number of misclassification errors in each training sample (Sampaio, 2023). The lower the value of C, the larger the margin size (Figure 5). With a larger margin, this means there will be more misclassifications because the model is more tolerant of outlier support vectors (Sampaio, 2023). On the other hand, the higher the value of C, the smaller the margin size (Figure 5). While a smaller margin can reduce misclassification errors, it may result in overfitting of the model to training data, and bad generalization on unseen data (Sampaio, 2023).

Figure 6

SVM Gamma Parameter

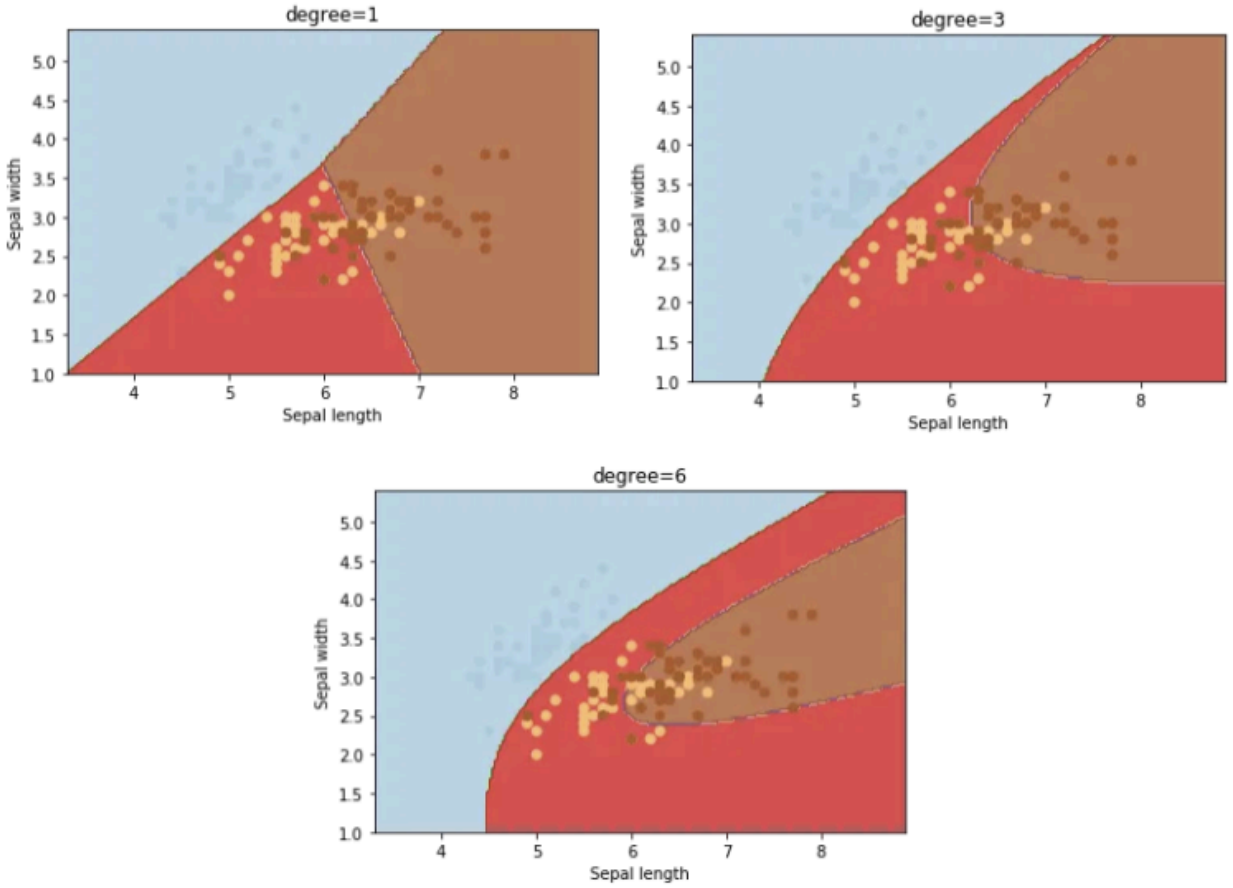


Note. Support Vector Machine Gamma Parameter. From *Understanding SVM Hyperparameters* [Diagram], by Sampaio, C, 2023, StackAbuse (https://stackabuse.com/understanding-svm-hyperparameters/).

The gamma parameter controls the distance a single data sample exerts influence on the decision boundary (Sampaio, 2023). Similar to the regularisation parameter, the higher the gamma value, the closer the points that are considered for the decision boundary (Figure 6). A higher gamma value will make the decision boundary smoother and less prone to overfitting on training data (Sampaio, 2023). On the contrary, the lower the gamma value, the farther the points are considered for choosing the decision boundary (Figure 6). As such, this will cause the model to capture noise in the training data and result in overfitting (Sampaio, 2023).

Figure 7

Degree Parameter for SVM Polynomial Kernel



Note. Degree Parameter for Support Vector Machine Polynomial Kernel. From *In Depth: Parameter tuning for SVC* [Diagram], by Fraj, M, B, 2018, Medium (https://medium.com/all-things-ai/in-depth-parameter-tuning-for-svc-758215394769).

The degree parameter is used in the polynomial SVM kernel. It is the degree of the polynomial used to determine the optimal hyperplane to classify the classes in the training data (Fraj, 2018). As observed from Figure 7, a higher level of degree can capture more complex relationships between data points but it will lead to overfitting. Whereas a lower degree polynomial will give a smoother decision boundary and better generalisation on unseen data.

Implementation of Model to Our Project

Figure 8

Data Pre-processing Before Model Training

```
n_samples = len(training_images)
# Flatten images
training_images_flattened = training_images.reshape(n_samples, -1)
training_images_normalized = training_images_flattened / 255.0
```

Regarding the implementation of the SVM classifier in our project, we first pre-process our image data. We resize the image to a uniform size of 50x50 pixels and then convert to numpy arrays. We then normalised the numpy arrays by dividing them by 255 to scale the pixel values to the range of 0 and 1. Lastly, we flattened the image data into a 1-dimensional numpy array before passing the input into the model for training (Figure 8).

After the necessary data pre-processing of the image data, we use the SVM model from Scikit-learn's library for our SVM model training.

We undergo a series of hyperparameter tuning to improve our SVM model performance. The motivation of this process of hyperparameter tuning is to understand how parameters like kernel functions, regularisation parameters, gamma values and the level of degree affect the performance of our SVM model.

We used the GridSearchCV function from the Scikit-learn library for our hyperparameter process. This function is useful to automatically explore the parameter space defined by us and find the best hyperparameter configurations.

Furthermore, we also implemented k-fold cross-validation in our model training to improve the performance of the model. However, due to computational limitations, we chose a low k-fold value of 2-fold cross-validation.

Figure 9

Hyperparameter Tuning Process using Linear SVM Kernel

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.01, 0.1, 1, 10, 100, 1000],
    'kernel': ['linear'],
}

# Grid search setup
grid = GridSearchCV(SVC(random_state=42), param_grid, cv=2, refit=True, verbose=2)
grid.fit(X_train, y_train)
```

Figure 9 illustrates an example of the hyperparameter tuning process using linear SVM kernel.

Linear Kernel

Figure 10

Parameter Values Used for Tuning the Linear SVM Kernel

```
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100, 1000],
    'kernel': ['linear'],
}
```

Figure 10 shows the parameter values used for tuning the linear SVM kernel. The primary focus of the hyperparameter tuning for linear kernel SVM is on the regularization parameter C. This parameter helps us to explore whether a simple linear decision boundary is suitable for the classifications of 26 different ASL signs.

Polynomial Kernel

Figure 11

Parameter Values Used for Tuning the Polynomial SVM Kernel

```
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['poly'],
    'degree': [4, 5],
    'gamma': [0.1, 1, 10]
}
```

Figure 11 shows the parameter values used for tuning the polynomial SVM kernel.The primary focus of the hyperparameter tuning for linear kernel SVM is on the regularization parameter C, level of degree, and the gamma values. These parameters help us to explore

whether complex non-linear decision boundaries are suitable for the classifications of 26 different ASL signs.

Radial Basis Function (RBF) Kernel

Figure 12

Parameter Values Used for Tuning the RBF SVM Kernel

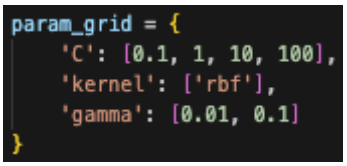
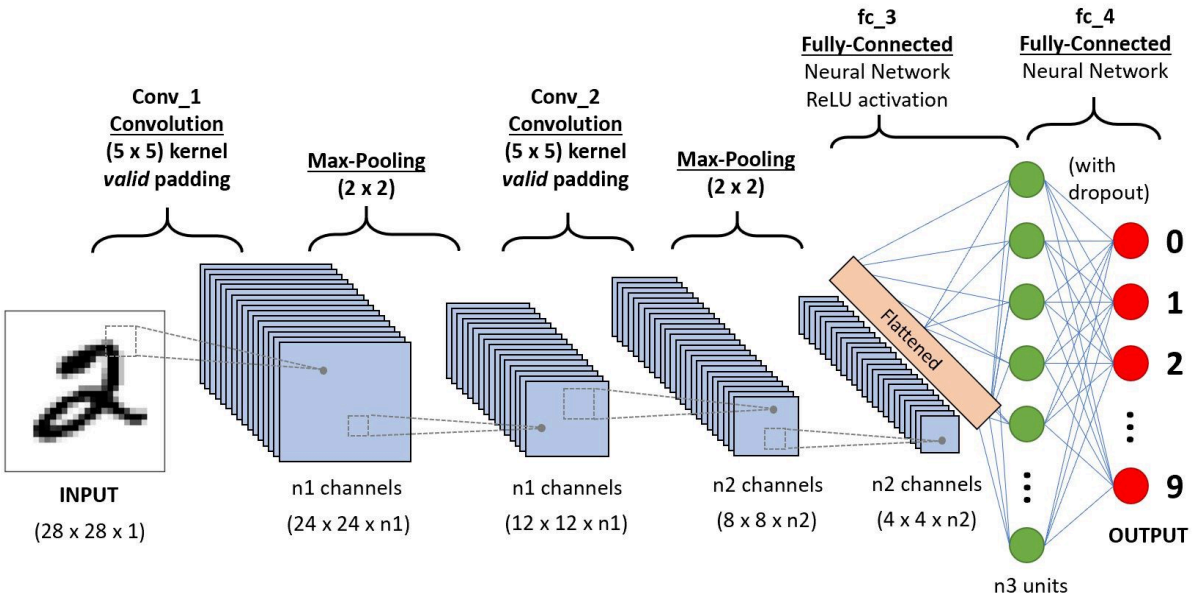


Figure 12 shows the parameter values used for tuning the RBF SVM kernel. The primary focus of the hyperparameter tuning for linear kernel SVM is on the regularization parameter C and the gamma values. These parameters help us to explore whether complex non-linear decision boundaries are suitable for the classifications of 26 different ASL signs in infinite dimensions.

4.3 Convolutional Neural Network (CNN)

Figure 13

CNN Architecture



Note. An example of CNN architecture to classify numbers. From *Convolution Neural Network (CNN) - Fundamental of Deep Learning* [Diagram], by T, Nikhil, 2021, Idiot Developer (http://idiotdeveloper.com/convolution-neural-network-cnn-fundamental-of-deep-learning/)

CNN is a commonly used model in image classification tasks. It utilises a Sequential layered architecture, whereby the output of a layer is the input for the next layer, stacking the layers in a sequential order. The Convolutional and MaxPooling layers are important layers to help the model train and learn the complex features in an image in order to do classification. Traditional neural networks require the input to be in 1-dimension (1D), but flattening the image to 1D would lead to a loss in spatial information such as the spatial relationship between each pixel. This is where a convolutional layer is required as it allows the input to be in 2-dimensions. Each convolution layer consists of filters for images, finding out different features from the images and calculating match feature points when testing.

The MaxPooling layers take large images and shrinks them down while preserving the core information within them, keeping the maximum value from each window, while reducing trainable parameters and computation during training.

The output from either the Convolutional or MaxPooling layer will be flattened into 1D and passed into a fully connected layer which does the classification. The output layer at the end will output the probability distribution of the image for each class.

With CNN, we first loaded the dataset and pre-processed the images using ImageDataGenerator. ImageDataGenerator is used to produce batches of augmented image data. This not only allows us to control the input into the model, but also helps with expanding the diversity of input images we train our model on (Brownlee, 2019). Within the augmentation step, we normalised and resized them, set the batch size, and the class mode. The use of normalisation is crucial as it reduces the large numbers that our model would have to work with, having the data in the same range for pixel values, and thus having a positive effect on the learning rate of the model (Riva, 2024). The batch size determines the number of training examples used in each iteration, which was a value of 32 we found from trial-and-error, finding that too large or too small numbers would affect the accuracy output of the model. Classmode was categorical due to the way our data was structured, and our use case, which was having images in a folder labelled with the categorical class they belong to. This classmode was used to access the images in the folders accordingly. Overall, we found the step of pre-processing to be extremely crucial in the model training step, as it helped to reduce the noise and latency of the data that the model trains on, thus improving the stability of the model's learning progress.

Additionally, we also explored the use of contrast and rotation of images. However, we found that contrasting the images brought the accuracy down, as did rotation. We attributed this to the fact that the lower contrast images would cause the model to lose out details that were important in the images such as the orientation of the hands. This was also likely for the rotation augmentation, as we noticed that some of the signs such as 'O' and 'C' use the same hand shape, as did 'H' and 'U', which were 2 fingers, differing in the rotation of the wrist. Hence, we chose not to do contrast and rotation for pre-processing in order not to introduce unnecessary noise that would not contribute to the model's learning.

The augmentation was done on all the images, so that the images are resized to the size of (50,50) the model expects as input. This included all our datasets - Train, Test and Validation. We explored the use of validation dataset, as it helps with tuning of the model's hyperparameters during training, and would help reduce overfitting by the model, as with other techniques we found to be useful to our model mentioned above in pre-processing, such as data augmentation (Lin, 2020).

Our CNN model utilised convolutional, max pooling, and dense layers. We have a flatten layer which flattens the input volume into a 1D array, which is necessary before passing it through fully connected layers. Finally, we have a fully connected layer, which takes the filtered images and translates them into categories with labels, outputting integer numbers upon classification.

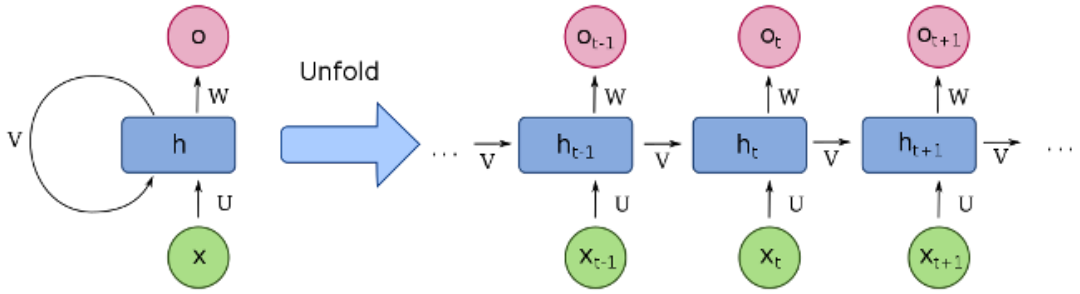
For CNN, there are various hyperparameters we can tune to improve the performance of the model. For the Convolutional layer, we varied the kernel size, number of filters and stride size. Activation functions for the layers are also hyperparameters that we can choose. From our research, Rectified Linear Unit (ReLU) was the most common activation function when it came to neural network models, swapping every negative number of the pooling layer with 0, which helps the model with the vanishing gradient problem, allowing for most robust training of the models. However, after reading further, we found that there was an alternative to ReLU, which is LeakyReLU, which factored for the “dying ReLU” problem ReLU faces, where neurons become inactive when training, outputting zero gradients. LeakyReLU solves this by allowing a small positive gradient, allowing some information to flow even for negative inputs, mitigating the vanishing gradient problem while ensuring that neurons can continue contributing to the learning process.

We ran multiple training while tuning the hyperparameters as well as adjusting the CNN architecture to identify the best performing model.

4.4 Recurrent Neural Network (RNN)

Figure 14

RNN Architecture



Note. Rolled RNN vs a Unrolled RNN. From *Deep Learning for Information Extraction* [Diagram] by Chi N., 2018, itemis (https://blogs.itemis.com/en/deep-learning-for-information-extraction)

While traditional neural networks assume that the inputs and outputs are independent of each other, RNN utilizes their “memory” and retains information from prior inputs to affect the subsequent inputs and resulting outputs. RNN is a commonly used model for processing sequential data such as time series data and natural languages. While it is a less conventional model for an image classification task such as ours, our research persuaded us to test the effectiveness of the model.

RNN leverages on internal memory, where the outputs are used as inputs again to understand the context for improved accuracy. Data processing nodes are structured as input, output and hidden layers. The input layer takes the sequence input, the hidden layer processes the data recursively, and the output layer generates the final output of the neural network. The hidden layer allows the network to capture sequential dependencies by remembering previous inputs while processing. Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) versions improve the RNN’s ability to handle long-term dependencies (IBM, n.d.). The variant of RNN used for this task, is the Long Short-Term Memory (LSTM) which uses a gated cell architecture to update and forget information selectively in the network memory (cell and hidden states).

Similar to the CNN, we pre-processed the images, using ImageDataGenerator, which includes resizing and normalising the input data. Each ASL hand sign image was resized to

64 by 64 pixels to maintain consistency in input dimensionality across the dataset. Furthermore, pixel values were normalized to a range of 0 to 1, facilitating faster and more stable training by reducing the variability in input data scale.

Unlike our CNN model, our RNNs required additional preprocessing as we had to align the image data format with the input requirements of the LSTM layers. A custom generator, “reshape_images_for_rnn”, was implemented to adapt these images for sequential processing by the RNN. Each image was reshaped from a 2D grid of pixels into a sequence of vectors. Each vector represents a row of pixels in the image, transforming the image into a series of timestep data that the RNN can process.

As mentioned earlier, we have decided to use an extension of RNN, LSTM. LSTMs are capable of storing and learning long-term dependencies and are designed so that the vanishing gradient problem is almost completely removed. These cells are designed to retain information and understand patterns over time. The core of our approach involves the two LSTM layers. The initial LSTM layer is defined with 256 units, which is the quantity of the amount of information the layer stores. The setup of the parameter “return_sequences=True” allows it to pass the full sequence of outputs to the next layer, capturing detailed temporal information across the entire "sequence" of the image. The second LTSM layer similarly features 256 units but does not return sequences. It only returns the output of the last timestep. This summarizes the information of the learned features, which is then used for the classification. The output layer is a Dense layer that uses a softmax function to classify the input into one of the 26 letters.

In the multiple training attempts of the RNN model, we experimented with different strategies to achieve better performance of the model. Initially we were working on a simple RNN architecture but soon found out that there were limitations with it as our performance was seen to stagnate. Hence, we shifted to LSTM. We explored the augmentation of the dataset where transformations such as rotations, zooms and different resizings were applied to the original images using the ‘ImageDataGenerator’. Another strategy was hyperparameter tuning where adjustments to the learning rate, number of LSTM units as well as the batch were explored in pursuit of a better performance. We had explored adding inserting Dropout layers in attempts to for the model to learn more robust features and prevent overfitting.

The training was augmented by a ModelCheckpoint callback, which monitored the validation accuracy and saved the model weights whenever an improvement was detected. This approach helped us identify the best model.

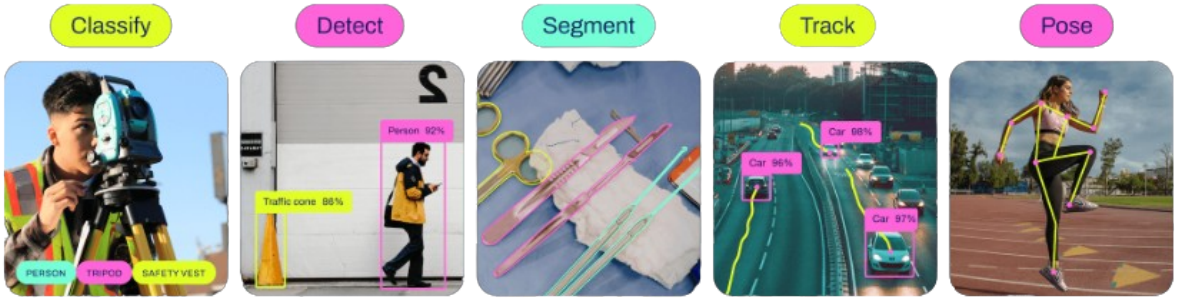
4.5 You Only Look Once (YOLO)

YOLO was developed by Joseph Redmon and Ali Farhadi at the University of Washington in 2015. Other models use classifiers to perform object detection but YOLO approached it as a regression problem to separately identify an object bounding box and its class label (Redmon et al., 2015).

YOLO is still in active development and comes with different models: Classify, Detect, Segment, Track, and Pose.

Figure 15

YOLO Models

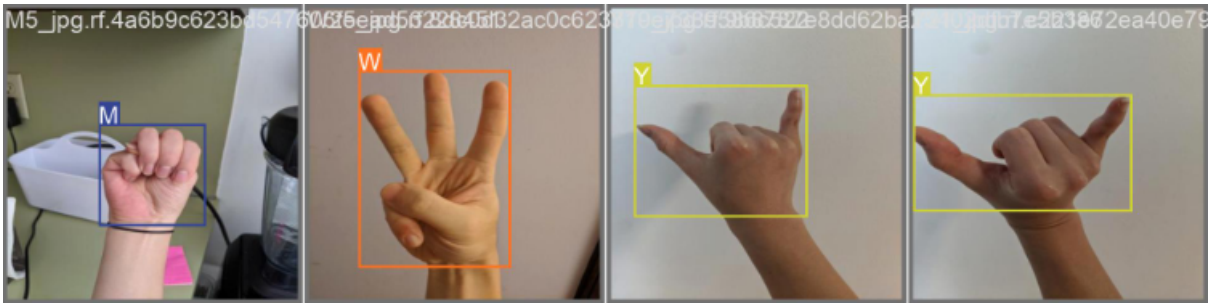


Note. Image of YOLO models by Ultralytics. From *GitHub Ultralytics*. (https://github.com/ultralytics/ultralytics?tab=readme-ov-file)

For our sign language detection purpose, we will be using the Detect model. The detect model identifies the location of an object, overlays a bounding box over the object, and gives it a class label. As shown in Figure 16, it is trained to detect our hands and label it an alphabet class that it identifies that the hand is signing.

Figure 16

YOLO Output



Note. Image taken from our YOLOv8 model output.

YOLO model has 5 key steps (Torres, 2024):

1. Image Division: It works by first dividing the input images into a grid of cells, typically 13 x 13 or 26 x 26 in size.
2. Feature Extraction: A deep CNN model is used here to extract high-level features from the image such as edges, shapes, and texture.
3. Bounding Box Prediction: For each cell, the model predicts multiple bounding boxes that are likely to contain the object - hand sign.
4. Class Prediction: With the bounding boxes, the model then predicts the probability of the object belonging to a specific alphabet class.
5. Non-Maxima Suppression (NMS): A hand sign could be predicted to be from different alphabet classes. Therefore, NMS uses the probability to filter for only 1 class that it is the most confident of before outputting the image with the bounding box and the associated alphabet class.

YOLO models detection as a regression problem. It divides the image into a S x S grid and for each grid cell predicts B bounding boxes, confidence rate for those boxes, and a class probability C. These predictions are encoded as the following tensor:

$$S \times S \times (B * 5 + C)$$

YOLO Model Tensor Equation (Redmon et al., 2015)

For our YOLO model, we trained it on the Roboflow American Sign Language dataset by David Lee. The dataset has already been labelled to recognize ASL which saves us time in manually labelling each of the images as the hand positions are different in each of the images. The YOLO model we used is specifically the YOLOv8 version which was the latest version at the time. It provided improvements in performance, flexibility, and efficiency compared to previous versions.

We trained the model for a total of 50 epochs with the images resized to 800x800 pixels. Given the library's capability, it was able to provide us with the best model from the 50 epochs which we then used as our final model for the live detection of ASL using webcam.

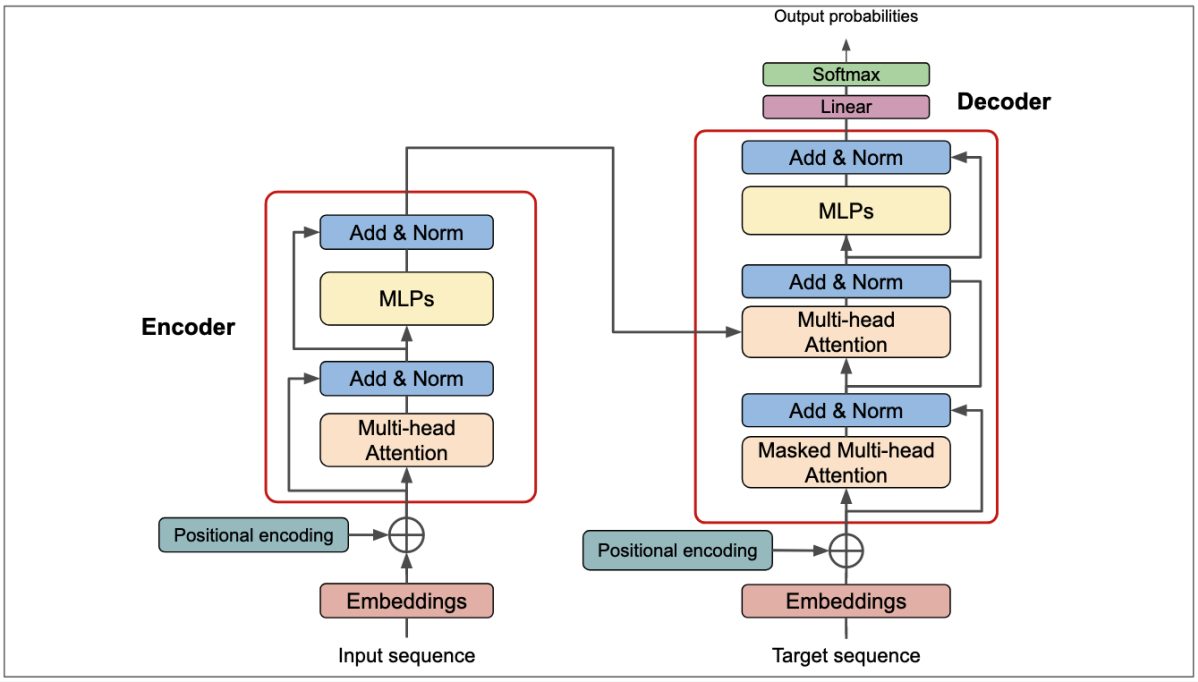
4.6 Vision Transformer (ViT)

ViT was introduced in 2021 by a group of Google Researchers in a paper called “An Image is Worth 16x16 Words: Transformers For Image Recognition at Scale” (Dosovitskiy et al., 2020).

It uses a Transformer architecture with an image as the input. Below is an image of the Transformer architecture. For ViT, it only uses the encoder part, which is the left side in the Figure below.

Figure 17

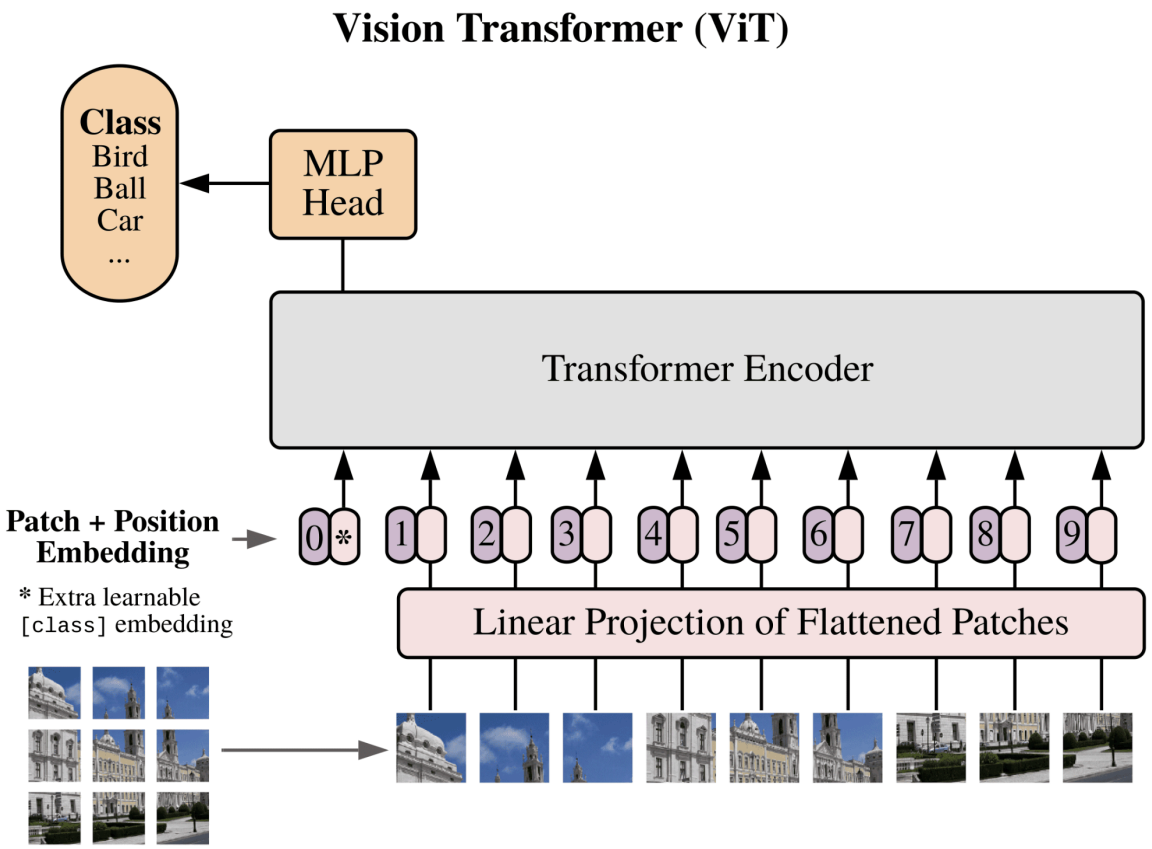
Transformer Architecture



Note. Standard architecture for Transformers. From *The transformer blueprint: A holistic guide to the transformer neural network architecture* [Diagram], by Nyandwi, J, 2023, Deep Revision (https://deeprevision.github.io/posts/001-transformer/).

Figure 18

Vision Transformer Process



Note. Vision transformer architecture and process flow. From *Vision transformer* [Diagram], by Papers with Code, (n.d), Paperswithcode (https://paperswithcode.com/method/vision-transformer). CC-BY-SA.

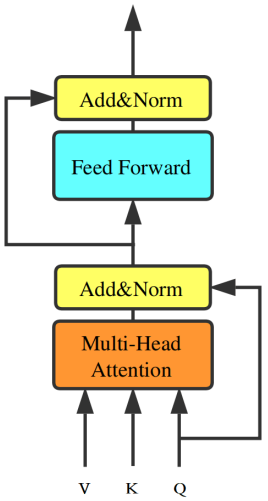
Firstly, an image input is divided into equal-sized patches (a hyperparameter for the model) and flattened into one dimension as seen above. Flattening the patches will scramble the patches, which results in the loss of spatial information of the image. To retain the spatial information, position embedding is added to each image patch to let the model know which part of the image patch comes first and the next patch after.

Besides passing the positional embedded image patches, an additional class embedding (token) will be passed into the Transformer encoder. This token serves as a placeholder to store the summary of the input image. This summary consists of the features, the relationship between the features and the global context of the features, representing the input image passed into the model. The Multi-layer Perceptron (MLP) head does the classification using only the information from the class token. Using the token helps avoid

bias towards specific image patches, allowing for more accurate prediction as it considers the entire image (Team, 2022).

Figure 19

Transformer Encoder Layers



Note. Neural network layers inside Transformer encoder block. From *Pre-trained bidirectional temporal representation for crowd flows prediction in regular region* [Diagram], by Duan, W, et al., 2019, IEEE Xplore (https://ieeexplore.ieee.org/abstract/document/8854786). CC-BY-SA 4.0.

The next step is passing the embedded patches into the Transformer Encoder. The encoder can contain one or more blocks, where a block consists of a combination of neural network layers within them. Typically in ViT, a block consists of a Multi-head Attention layer, Normalisation layer and an MLP layer (also known as FeedForward Neural Network).

The multi-head Attention block is where the model learns about the features in each image patch in parallel, allowing the model to learn both local and global information about the image. Multi-head attention addresses one of the limitations of CNN by reducing the loss of crucial spatial information that occurs when the image goes through convolutional layers. Each attention head is a self-attention mechanism which allows the model to focus on important features in the patch (Kirouane, 2023).

From Figure 19 above, two arrows skip the Multi-head attention block and the MLP layer. These two arrows represent residual connections. Residual connections allow data to skip

some layers and flow to the later layers. Residual connections have proven to make training of deep neural networks easier. (Wong, 2022).

Models with residual connections require an Add and Normalization layer to concatenate the residual connection data with the previous layer(s) output. Normalization helps to keep the data in the same range to improve the stability and efficiency of training.

Lastly, the MLP head will output the probability of the class distribution.

Similar to other models, data preprocessing was done. For ViT, the images were resized to 256x256 pixels and normalized by dividing the pixels by 255.

The initial idea was to use a ViT built from scratch using the Keras library. With reference to the standard Transformer encoder architecture, we successfully built a Vision Transformer from scratch. Before passing the image into the encoder, each image is split into patches and embedded with positional embeddings.

Hyperparameter tuning and architecture adjustments were conducted to further improve the performance of the model based on the validation accuracy obtained after training.

5. Results and Discussions

5.1. Support Vector Machine (SVM)

After a comprehensive hyperparameter tuning process across the different SVM kernel types, the following is the training accuracy for the different kernel types after tuning them respectively.

Kernel Type (with hyperparam. tuning)	Train Accuracy
Linear	71.90%
Polynomial	42.30%
Radial Basis Function (RBF)	52.70%

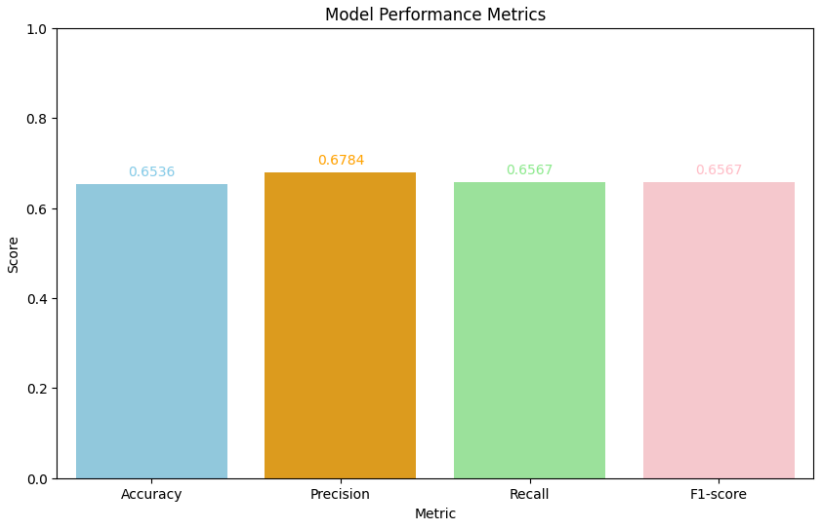
Table 1: Training Results from SVM Model across Different Kernels

Based on Table 1, it is evident that the linear kernel SVM model outperformed the poly kernel SVM model and the RBF kernel SVM model. The linear kernel SVM model gives the highest train accuracy of 71.90%. This suggests that a linear decision boundary is more suitable to classify the 26 different ASL signs (classes) for our dataset.

The results from the hyperparameter tuning process challenges our initial belief that classifying 26 different ASL sign classes is very complex and RBF kernel will outperform linear kernel and polynomial kernel. However, despite the inherent complexity of the ASL signs, the linear decision boundary can actually capture the nuances between the different sign classes.

Figure 20

Model Performance Metrics of the Linear SVM Kernel on Test Dataset

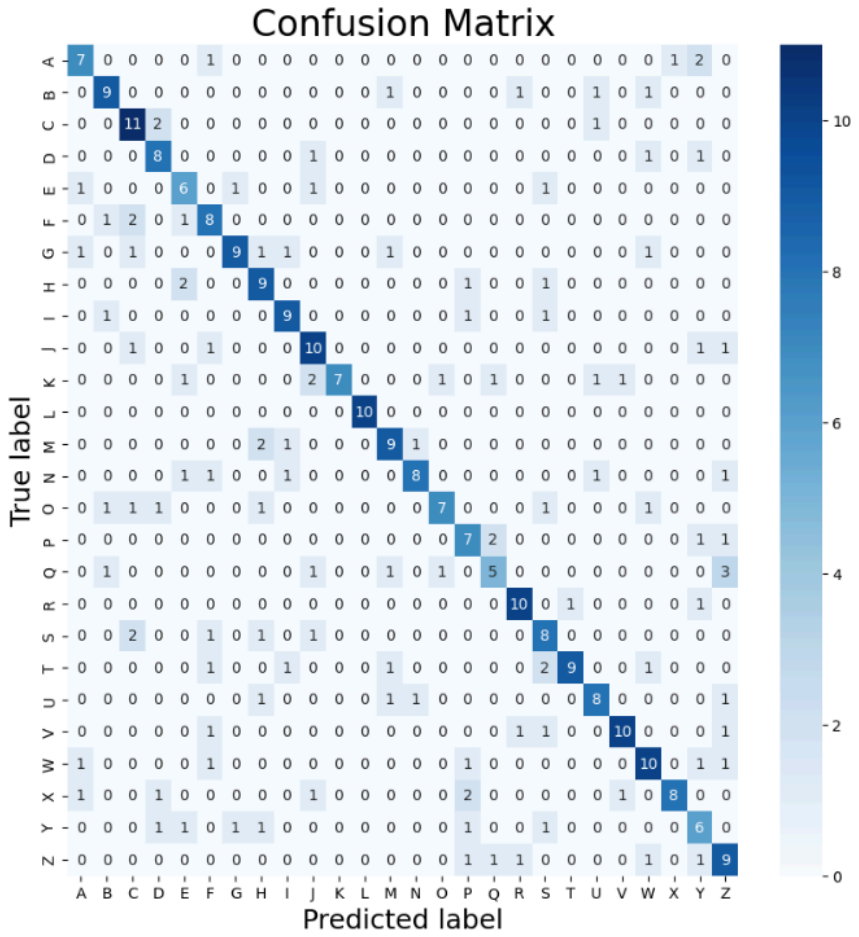


Key performance metrics, such as accuracy, precision, recall and f1-score were used to evaluate the performance of our best model, **Linear SVM Kernel.**

Based on Figure 20, the model gives an accuracy of 65.36%. This indicates a good proportion of total predictions were correct. The model gives a precision of 0.6784. This highlights that the model was correct about two-thirds of the time when it predicted an ASL sign class. The model gives a recall of 0.6567. This means that the model identified over 65% of all actual positives. Lastly, the model gives a F1-score of 0.6567.

Figure 21

Confusion Matrix for Linear SVM Kernel Classification on Test Dataset



We further evaluated the model's classification performance using the confusion matrix. Based on Figure 21, the model gives an overall good prediction, where a clear darker shade of diagonal line from the top left to the bottom right can be observed from the confusion matrix. This is a good sign because these values represent the true positives.

5.2. Convolutional Neural Network (CNN)

Before training the model, we compiled our model with Adam optimizer and used the categorical_crossentropy loss function, which is a common choice for multi-class classification problems, hence suiting our use case of multi-classes for the ASL alphabets.

Figure 22

Keras Callback Functions used for CNN Model

```
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5)
```

As seen in Figure 22, using the EarlyStopping Keras callback function, our CNN model was trained till the 34th epoch, where validation loss did not change much for 10 epochs, and it would restore the best weights. We also utilised the ReduceLROnPlateau callback function, which would adjust the learning rate by a factor of 0.1 if validation loss did not improve for 5 epochs, starting at 1.0000e-04, the learning rate was reduced to 1.0000e-06.

With the model and the tuned hyperparameters as well as features that we experimented with, removing and adding those we felt would help given our use case, we achieved an accuracy of 73.76%, a precision score of 0.7641, recall of 0.7376, and finally an f1-score of 0.7390, as seen in our classification report in Figure 23.

Figure 23

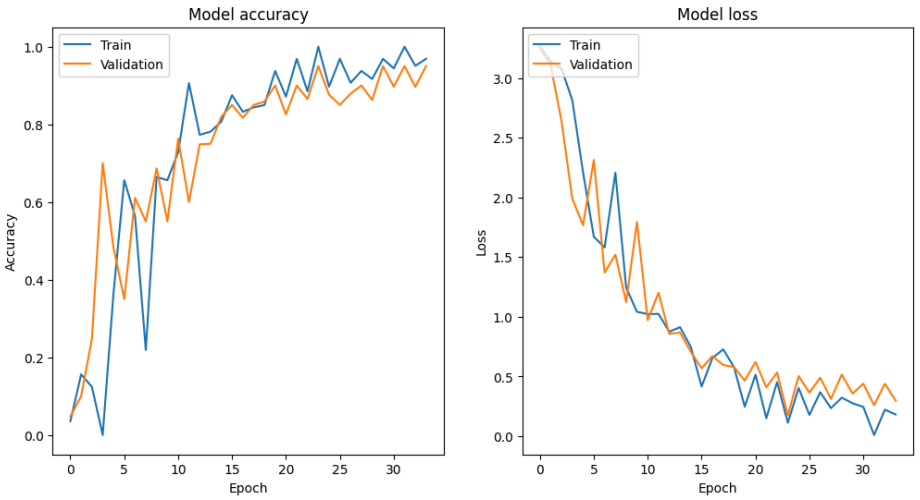
Classification Report of CNN Model

	precision	recall	f1-score	support
A	0.7200	0.8182	0.7660	22
B	1.0000	0.7692	0.8696	13
C	0.8333	0.7143	0.7692	14
D	0.7000	0.6364	0.6667	11
E	0.8889	0.8000	0.8421	10
F	0.7143	0.8333	0.7692	12
G	0.9000	0.6000	0.7200	15
H	0.6923	0.6923	0.6923	13
I	0.6429	0.7500	0.6923	12
J	0.6111	0.7857	0.6875	14
K	1.0000	0.5714	0.7273	14
L	0.9091	1.0000	0.9524	10
M	0.7500	0.6923	0.7200	13
N	0.5625	0.6923	0.6207	13
O	0.5000	0.6923	0.5806	13
P	0.7500	0.8182	0.7826	11
Q	0.5714	0.3333	0.4211	12
R	0.6667	0.8333	0.7407	12
S	0.7500	0.6923	0.7200	13
T	0.6667	0.6667	0.6667	15
U	0.5000	0.8333	0.6250	12
V	1.0000	0.7143	0.8333	14
W	1.0000	0.8667	0.9286	15
X	0.9000	0.6429	0.7500	14
Y	0.7500	0.7500	0.7500	12
Z	0.8235	1.0000	0.9032	14
accuracy			0.7376	343
macro avg	0.7616	0.7384	0.7383	343
weighted avg	0.7641	0.7376	0.7390	343

Overall, from Figure 24, we can see that the model accuracy and loss follows a roughly expected behaviour. Model accuracy increases as the model learns and shows to maintain at the same level before the callback function is triggered. Model loss also shows an elbow shape, as it slowly converges at a low point, which means the model is unlikely to have underfitted. Additionally, given that the training accuracies are generally found roughly equal or below the test set, as do the loss values, which appear higher for train than test, it is unlikely the model has overfitted, as it shows that the model is able to generalise well for test unseen data. The choppy nature of the graphs might be attributed to the callback functions which adjusts the model during training.

Figure 24

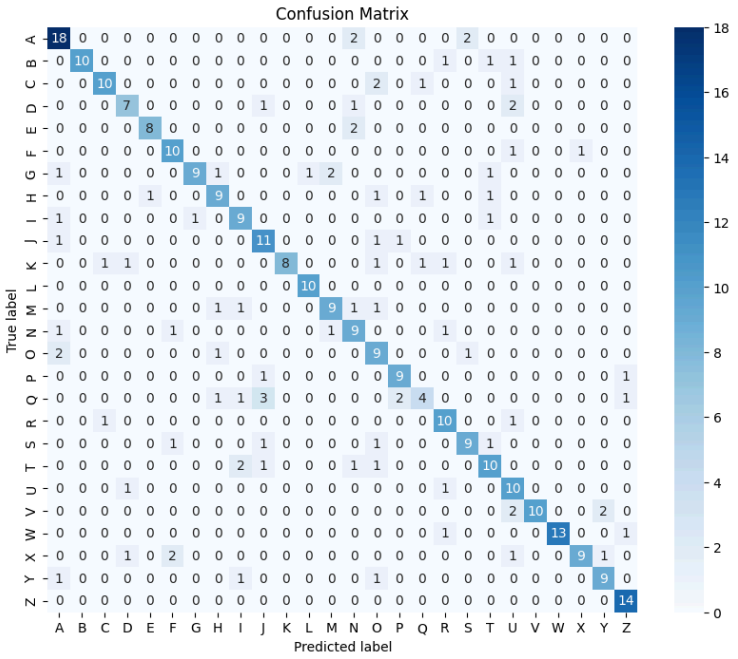
Model Accuracy and Loss of CNN Model



We can also see that in Figure 25, the confusion matrix shows a good representation of the Actual and predicted labels, where the majority of the labels are predicted correctly, substantiating the accuracy, precision, and recall values stated above.

Figure 25

Confusion Matrix for CNN Model



5.3. Recurrent Neural Network (RNN)

The model was trained for 10 epochs, with a batch size of 32 along with the usage of the Adam optimizer. Tensorflow's ModelCheckpoint function was used to save the best model configurations to be loaded for future training. The results obtained are as shown below.

Figure 26

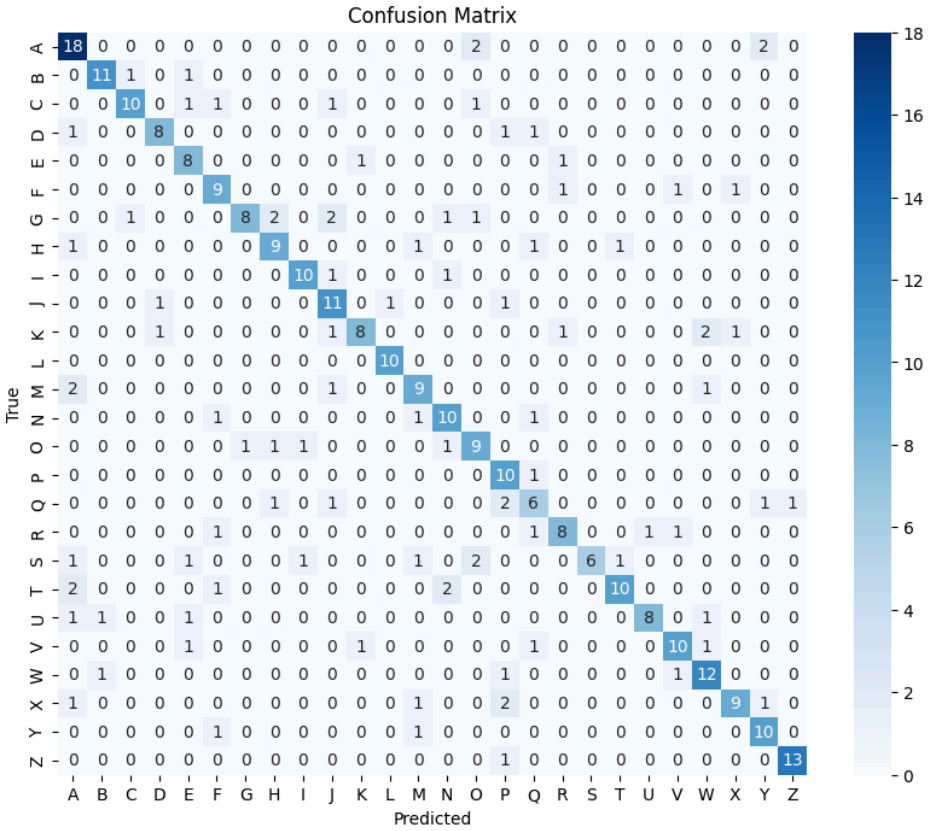
RNN classification report

	precision	recall	f1-score	support
A	0.6667	0.8182	0.7347	22
B	0.8462	0.8462	0.8462	13
C	0.8333	0.7143	0.7692	14
D	0.8000	0.7273	0.7619	11
E	0.6154	0.8000	0.6957	10
F	0.6429	0.7500	0.6923	12
G	0.8889	0.5333	0.6667	15
H	0.6923	0.6923	0.6923	13
I	0.8333	0.8333	0.8333	12
J	0.6111	0.7857	0.6875	14
K	0.8000	0.5714	0.6667	14
L	0.9091	1.0000	0.9524	10
M	0.6429	0.6923	0.6667	13
N	0.6667	0.7692	0.7143	13
O	0.6000	0.6923	0.6429	13
P	0.5556	0.9091	0.6897	11
Q	0.5000	0.5000	0.5000	12
R	0.7273	0.6667	0.6957	12
S	1.0000	0.4615	0.6316	13
T	0.8333	0.6667	0.7407	15
U	0.8889	0.6667	0.7619	12
V	0.7692	0.7143	0.7407	14
W	0.7059	0.8000	0.7500	15
X	0.8182	0.6429	0.7200	14
Y	0.7143	0.8333	0.7692	12
Z	0.9286	0.9286	0.9286	14
accuracy			0.7289	343
macro avg	0.7496	0.7314	0.7289	343
weighted avg	0.7504	0.7289	0.7280	343

The classification report shows the overall accuracy of the model achieved a 72.89%, a precision of 0.7504, a recall of 0.7289. Resulting in an F1-Score of 0.7280.

Figure 27

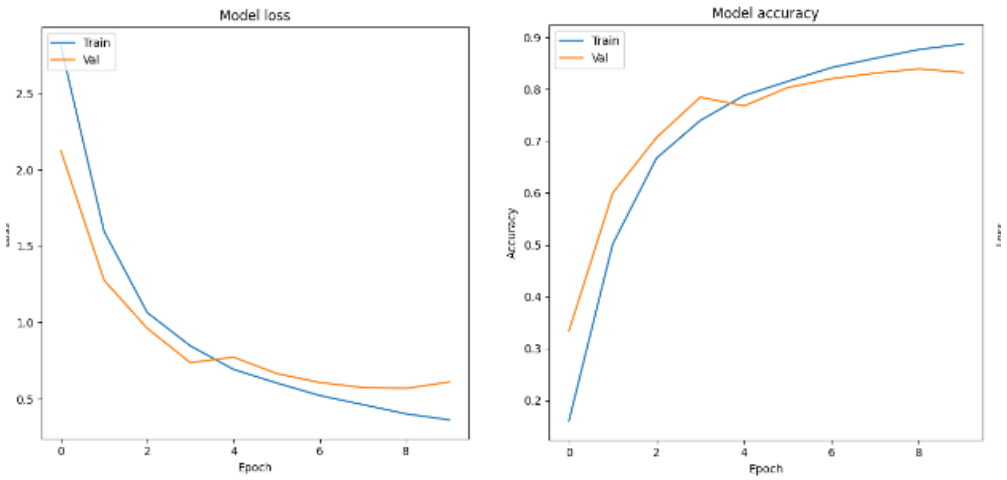
RNN confusion matrix



The confusion matrix helps to visualise the classification model. The matrix shows a good diagonal line, indicating that the model can accurately translate sign language into their respective alphabet classes.

Figure 28

RNN model loss and accuracy



The training phases were tracked as well, specifically the loss and accuracy. The behaviour was expected, showing the model's loss decreased and accuracy increased over epochs, demonstrating learning and improvement over time. Closer to the end, both the loss and accuracy start to reach a plateau.

5.4. You Only Look Once (YOLO)

We trained our YOLO model on YOLOv8 for a total of 50 epochs and used the best model for our hand sign detection. The best model is determined by the mean Average Precision at 50% (mAP50) and mean Average Precision at 50% to 95% (mAP50-95) with a weightage of 10% and 90% respectively (Ultralytics, n.d.). mAP50 measures the average precision of the model across different levels of recall, up to a threshold of 50% and mAP50-95 measures the average precision across different levels of recall within the range of 50% to 95%.

From our 50 epochs, looking at the confusion matrix in Figure 29, we can see that the model is largely accurate in predicting the correct labels with only the occasional mistakes where backgrounds are labelled as a recognized ASL. The best model used was epoch number 48 with a mAP50 value of 0.9659 and a mAP50-95 value of 0.8139 as shown in Figure 30. It had a precision value of 0.9372 and a recall value of 0.9068. The result is promising as the performance metrics are between 80% to 96% which means that it is likely to predict the hand signs correctly and will generalise well to predict for unseen data.

Figure 29

Confusion matrix for YOLO model on test dataset

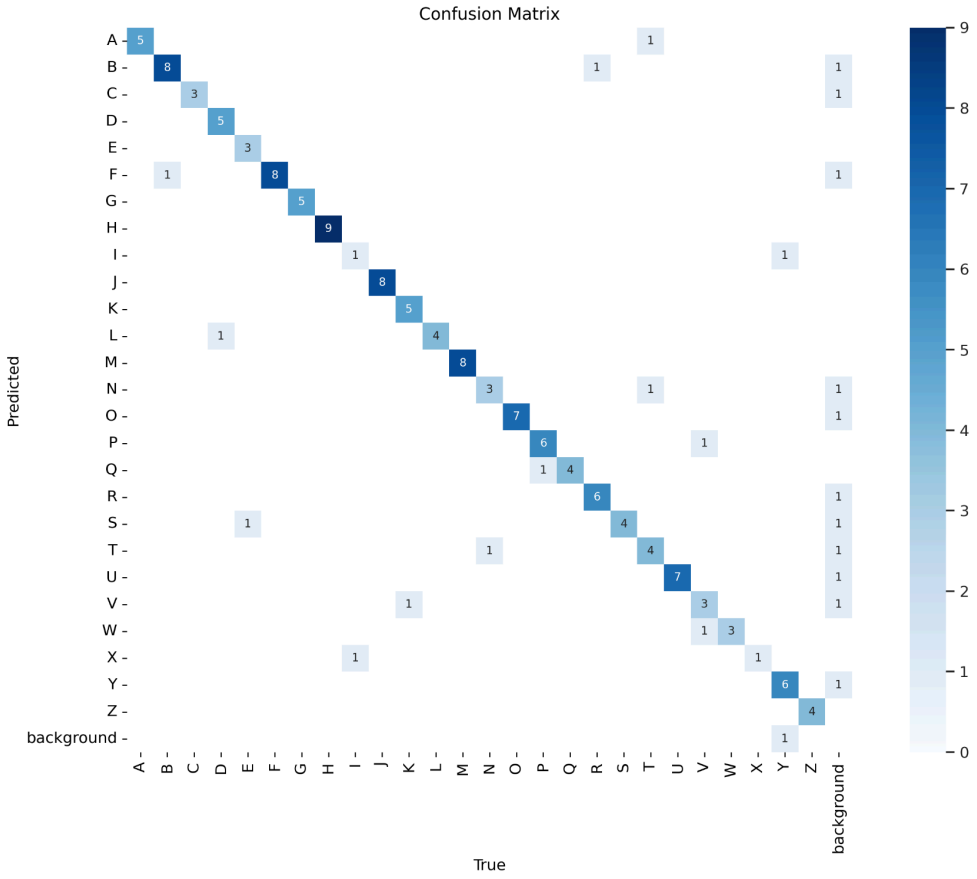
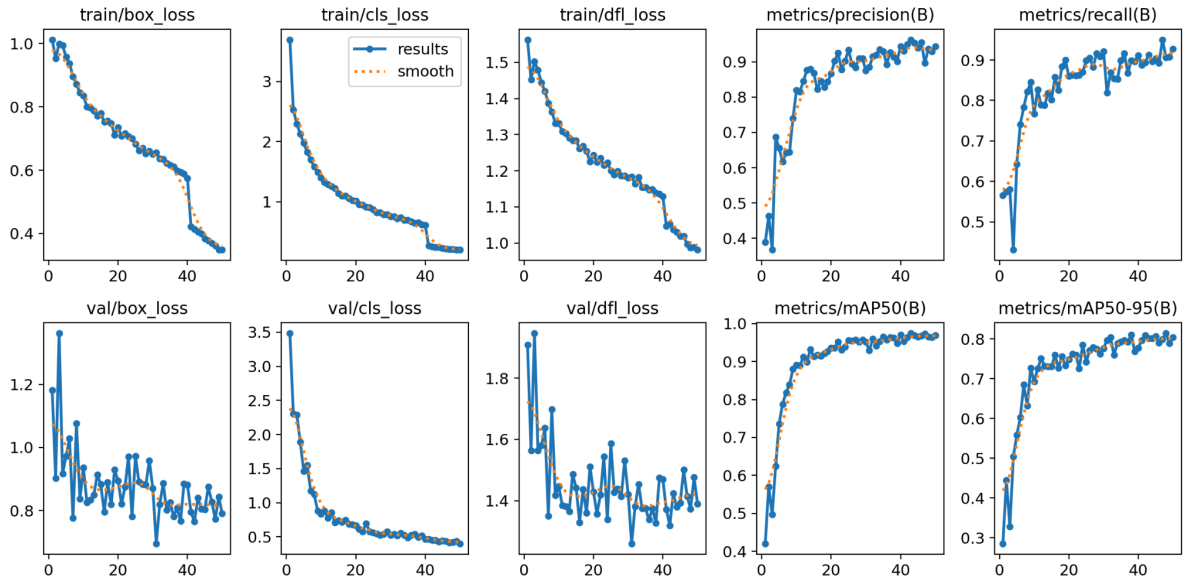


Figure 30

Performance charts on YOLO model for test and validation datasets across 50 epochs

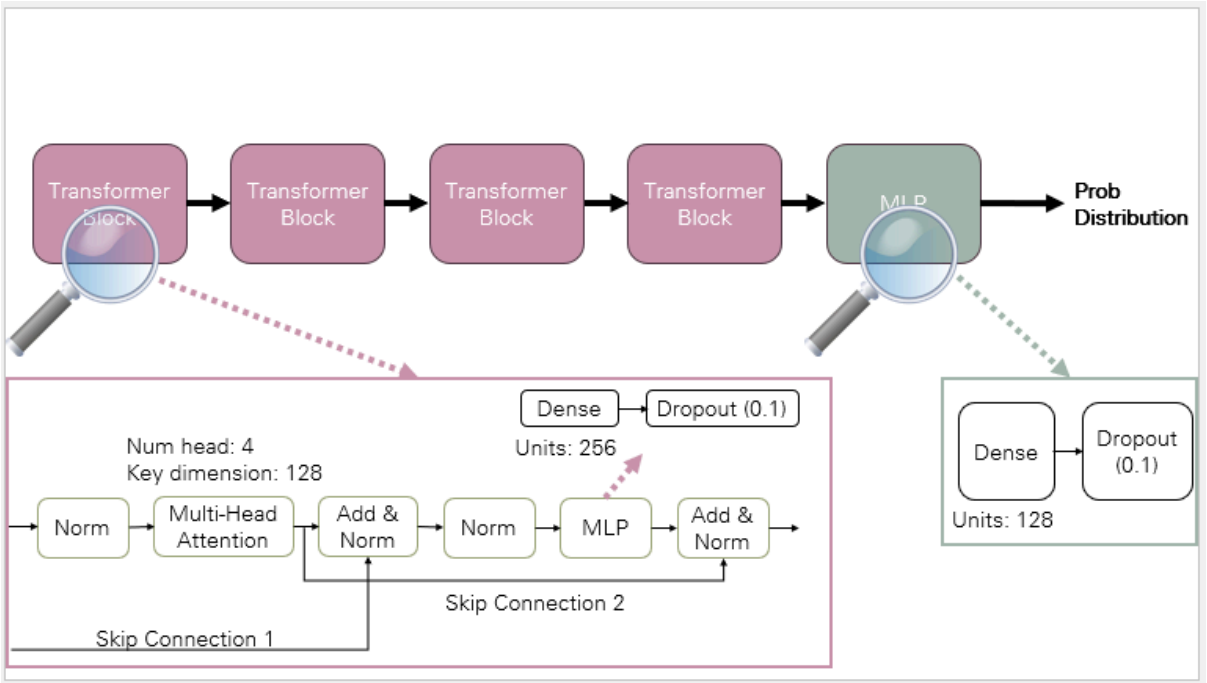


5.5. Vision Transformer (ViT)

This ViT has four transformer encoder layers, followed by an MLP block, before passing the data into an output layer that gives probability distribution. The image below describes the architecture of our ViT.

Figure 31

Custom ViT Architecture



Despite multiple hyperparameters and architecture adjustments, the model was unable to converge. With reference to the figure below, the training loss constantly remained around 3.265. We used early stopping to stop the training when the model did not improve after ten epochs.

Figure 32

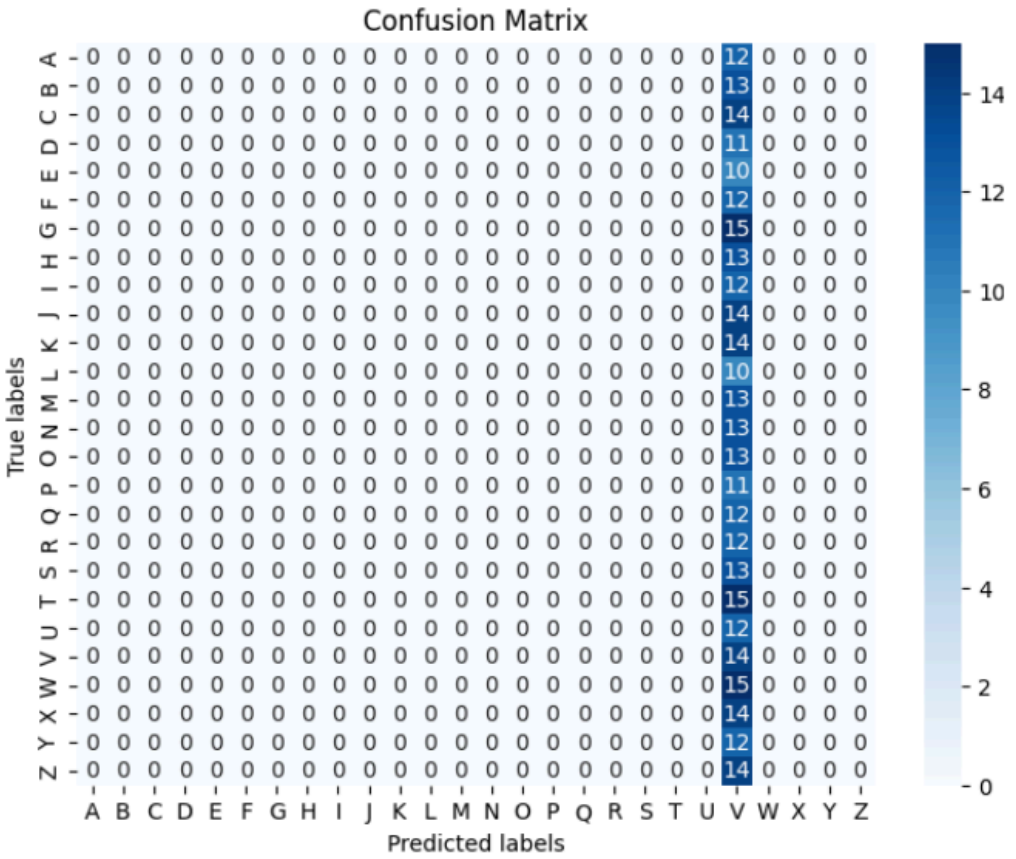
Training history for ViT model (built from scratch)

```
779/779 [==============================] - 307s 376ms/step - loss: 3.4039 - Accuracy: 0.0366 - val_loss: 3.2609 - val_Accuracy: 0.0382
Epoch 2/50
779/779 [==============================] - 68s 86ms/step - loss: 3.2645 - Accuracy: 0.0359 - val_loss: 3.2611 - val_Accuracy: 0.0374
Epoch 3/50
779/779 [==============================] - 68s 86ms/step - loss: 3.2645 - Accuracy: 0.0362 - val_loss: 3.2616 - val_Accuracy: 0.0382
Epoch 4/50
779/779 [==============================] - 68s 87ms/step - loss: 3.2640 - Accuracy: 0.0381 - val_loss: 3.2610 - val_Accuracy: 0.0382
Epoch 5/50
779/779 [==============================] - 68s 87ms/step - loss: 3.2644 - Accuracy: 0.0378 - val_loss: 3.2627 - val_Accuracy: 0.0382
Epoch 6/50
779/779 [==============================] - 68s 86ms/step - loss: 3.2645 - Accuracy: 0.0389 - val_loss: 3.2620 - val_Accuracy: 0.0382
Epoch 7/50
779/779 [==============================] - 68s 86ms/step - loss: 3.2651 - Accuracy: 0.0374 - val_loss: 3.2626 - val_Accuracy: 0.0382
Epoch 8/50
779/779 [==============================] - 68s 86ms/step - loss: 3.2649 - Accuracy: 0.0389 - val_loss: 3.2614 - val_Accuracy: 0.0382
Epoch 9/50
779/779 [==============================] - 68s 86ms/step - loss: 3.2658 - Accuracy: 0.0386 - val_loss: 3.2620 - val_Accuracy: 0.0382
Epoch 10/50
779/779 [==============================] - 68s 87ms/step - loss: 3.2657 - Accuracy: 0.0386 - val_loss: 3.2626 - val_Accuracy: 0.0382
Epoch 11/50
777/779 [==============================>.] - ETA: 0s - loss: 3.2664 - Accuracy: 0.0360Restoring model weights from the end of the best epoch: 1.
779/779 [==============================] - 68s 87ms/step - loss: 3.2664 - Accuracy: 0.0360 - val_loss: 3.2631 - val_Accuracy: 0.0382
Epoch 11: early stopping
```

With this low performance, we evaluated this model on the test dataset and the model was predicting a single label for every image. This is evident from the confusion matrix below.

Figure 33

Confusion matrix for custom ViT prediction on test data



After multiple experimentations, the training loss and accuracy did not improve. To overcome this, we decided to use transfer learning, where we used a pre-trained model from HuggingFace and researched why the model faced trouble during training.

Our research found that ViT requires a lot of data and is computationally intensive to train. During the experiments to improve the performance of the ViT, we tried increasing the complexity of the model by increasing the number of transformer blocks but faced memory errors. The error is because ViT has a lot of parameters, and increasing the model complexity would mean it requires higher computational resources (Dosovitskiy et al., 2020).

We decided to use Google's pre-trained ViT model and fine-tune it to our problem domain, which is to recognize hand signs (*Google/Vit-base-patch16-224 · Hugging Face*, 2023). The pre-trained model was trained using the 1 million images in ImageNet. This model takes in an image with 224x224 pixels and splits into patches with 16x16 pixels. The figure below shows the confusion matrix, showing the number of true positives for each class using the test data. The pre-trained model performs better as there is a clear diagonal line with darker shades. The results show that the model can classify most of the images correctly.

Figure 34

Confusion matrix for pretrained ViT classification on test data

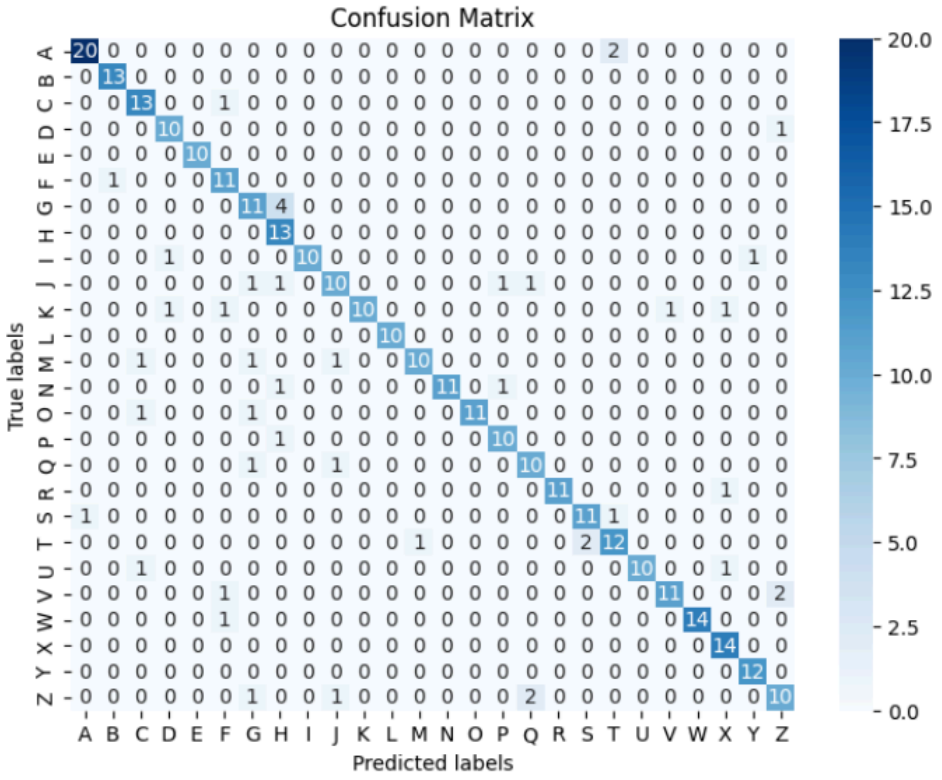
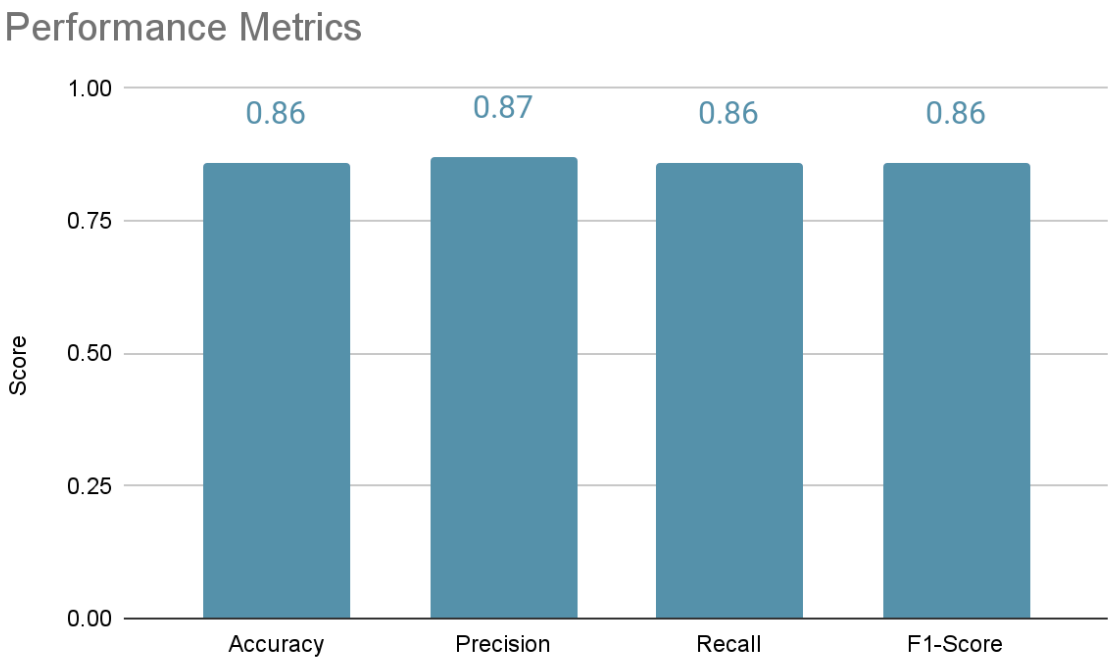


Figure 35

Performance metrics for pretrained ViT



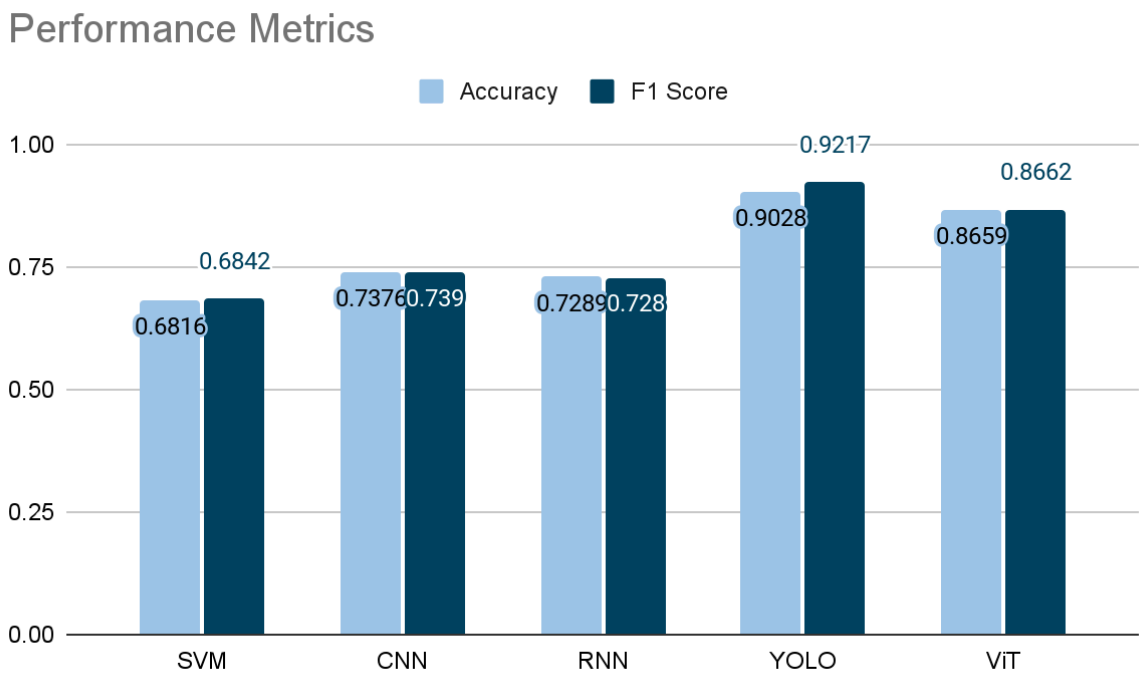
The pretrained ViT did significantly better than the custom ViT built from scratch with all of the performance measures, such as accuracy, precision, recall and F1-score, being over 0.8.

6. Conclusion and future work

6.1. Model Comparison

Figure 36

Performance metrics for all models



Among the 5 classification models, YOLO performed the best, with an accuracy of 0.9217 and F1 score of 0.9028.

6.2. Limitations

Despite obtaining good results on the test dataset, there is still room for improvement. When we tested our solution, we found that the different classifiers still made some mistakes when we used our hands. For example, for the hand sign M, the model may predict a different class sometimes. One reason for this might be due to our dataset being signer-dependent. Signer-dependent refers to when the dataset contains the hands of one or a few people. This characteristic will result in poor generalization as different people have different hand sizes or styles of hand signs due to their hand structure or muscles. Our results aligned with one of the literature reviews we had done earlier, where the signer-dependent models had lower accuracy than signer-independent models.

One possible solution for this limitation could be gathering a diverse hand sign dataset from the web. It can help ensure the models can learn robust features of the hand signs and not be biased towards specific signers.

The second limitation would be the use of static images for training. Hand signs like J and Z require motion, and our current models cannot predict J and Z accurately. In the future, we could use video as the training data to improve the performance of our models.

6.3. Future Work

Our solution showcases the potential of translating hand signs in real time but further work can be done in this area to foster inclusivity and understanding between deaf people and people who are able to hear. Instead of using letters, we could have the solution to translate words instead. In addition, our solution can be further improved by forming sentences as the user is doing the hand sign. This can help bridge communication between deaf people and people who do not know sign language.

7. References

American Sign Language Letters Object Detection Dataset. (2022, July 27).[Data set] Roboflow. https://public.roboflow.com/object-detection/american-sign-language-letters/

Arikeri, P. (2021, June 4). *American sign language (ASL) dataset*.[Data set] Kaggle. https://www.kaggle.com/datasets/prathumarikeri/american-sign-language-09az

Barnett, S., McKee, M., Smith, S. R., & Pearson, T. A. (2011). Deaf sign language users, health inequities, and public health: Opportunity for social justice. *Preventing Chronic Disease*, *8*(2), A45–A45.https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3073438/

Brownlee, J. (2019, July 5). *How to configure image data augmentation in Keras*. MachineLearningMastery.com. https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

Chi, N. (2018, Nov 19). Deep Learning for Information Extraction.[Diagram].itemis https://blogs.itemis.com/en/deep-learning-for-information-extraction

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020, October 22). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv.org. https://arxiv.org/abs/2010.11929

Duan, W., Jiang, L., Wang, N., & Rao, H. (2019). *Pre-trained bidirectional temporal representation for crowd flows prediction in regular region*. [Diagram]. IEEE Xplore *IEEE Access*, *7*, 143855-143865.

Fletcher, T. (2009). Support vector machines explained. *Tutorial paper*, *1118*, 1-19.

Fraj, M, B. (2018, January 5). *In Depth: Parameter tuning for SVC.* Medium. https://medium.com/all-things-ai/in-depth-parameter-tuning-for-svc-758215394769

Fraj, M, B. (2018, January 5). In Depth: Parameter tuning for SVC [Diagram]. Medium. https://medium.com/all-things-ai/in-depth-parameter-tuning-for-svc-758215394769

google/vit-base-patch16-224 · Hugging Face. (2023, December 21). https://huggingface.co/google/vit-base-patch16-224

IBM. (n.d.). *What are Recurrent Neural Networks?* IBM https://www.ibm.com/topics/recurrent-neural-networks

Jain, V., Jain, A., Chauhan, A., Kotla, S.S., & Gautam, A. (2021). American Sign Language recognition using Support Vector Machine and Convolutional Neural Network. *International Journal of Information Technology (Singapore. Online), 13*, 1193 - 1200. https://doi.org/10.1007/s41870-021-00617-x

Kirouane, A. (2023, April 4). *Demystifying Vision Transformers (VIT): A revolution in computer vision*. https://www.linkedin.com/pulse/demystifying-vision-transformers-vit-revolution-ayoub-kirouane/

Lee, C. K. M., Ng, K. K. H., Chen, C.-H., Lau, H. C. W., Chung, S. Y., & Tsoi, T. (2021, June). *American Sign Language Recognition and Training Method with Recurrent Neural Network*, 114403. https://doi.org/10.1016/j.eswa.2020.114403

Lin, D. C.-E. (2020, June 7). *8 simple techniques to prevent overfitting*. Medium. https://towardsdatascience.com/8-simple-techniques-to-prevent-overfitting-4d443da2ef7d

Liu, Y., Nand, P., Hossain, A., Nguyen, M., & Yan, W. (2023). Sign language recognition from digital videos using feature pyramid network with detection transformer. *Multimedia Tools and Applications*, *82*(14), 21673–21685. https://doi.org/10.1007/s11042-023-14646-0

Ma, Y., & Guo, G. (Eds.). (2014). *Support vector machines applications* (Vol. 649). New York: Springer.

Meyer, D., & Wien, F. T. (2001). Support Vector Machines. *R News*, *1*(3), 23-26.

Meyer, D., & Wien, F. T. (2001). Support Vector Machines [Diagram]. R News. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4850faab0aab5c4b40fcdd5a77d9e7626a163db5

Nyandwi, J. (2023, July 29). *The Transformer Blueprint: A holistic guide to the Transformer Neural Network architecture*. [Diagram]. Deep Revision https://deeprevision.github.io/posts/001-transformer/

Papers with code - Vision Transformer explained. (n.d.). [Diagram]. Papers with Code https://paperswithcode.com/method/vision-transformer

Redmon, J., Santosh, D. H. H., Ross, G., & Farhadi, A. (2015). You only look once: Unified, Real-Time Object Detection. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1506.02640

Riva, W. by: M. (2024, March 18). *Batch normalization in Convolutional Neural Networks*. Baeldung on Computer Science. https://www.baeldung.com/cs/batch-normalization-cnn

Sampaio, C. (2023, July 2). *Understanding SVM Hyperparameters.* StackAbuse. https://stackabuse.com/understanding-svm-hyperparameters/

Sampaio, C. (2023, July 2). Understanding SVM Hyperparameters [Diagram]. StackAbuse. https://stackabuse.com/understanding-svm-hyperparameters/

Shukla, A., Harper, M., Pedersen, E., Goman, A., Suen, J. J., Price, C., Applebaum, J., Hoyer, M., Lin, F. R., & Reed, N. S. (2020). Hearing Loss, Loneliness, and Social Isolation: A Systematic Review. *Otolaryngology--head and neck surgery : official journal of American Academy of Otolaryngology-Head and Neck Surgery*, *162*(5), 622–633. https://doi.org/10.1177/0194599820910377

Steve. (2012, June 24). Image normalization, image range and image scaling for different stack of images [Diagram]. StackOverflow. https://stackoverflow.com/questions/11178925/image-normalization-image-range-and-image-scaling-for-different-stack-of-images

Team, D. G. (2022, January 5). Vision Transformers for Computer Vision - Deep GAN Team - Medium. *Medium*. https://deepganteam.medium.com/vision-transformers-for-computer-vision-9f70418fe41a

Tomar, N. (2021). *Convolution Neural Network (CNN) - Fundamental of Deep Learning*. [Diagram]. Idiot Developer http://idiotdeveloper.com/convolution-neural-network-cnn-fundamental-of-deep-learning/

Torres, J. (2024, January 4). How Does YOLOv8 Work? A Peek Inside its Object Detection Brain - YOLOv8. *YOLOv8*. https://yolov8.org/how-does-yolov8-work/

Ugale, M., Shinde, O. R. A., Desle, K., & Yadav, S. (2023). A Review on Sign Language Recognition Using CNN. *Advances in Computer Science Research*, 251–259. https://doi.org/10.2991/978-94-6463-136-4_23

Ultralytics. (n.d.). ultralytics/ultralytics: NEW - YOLOv8 in PyTorch GitHub. https://github.com/ultralytics/ultralytics?tab=readme-ov-file

Ultralytics. (n.d.). ultralytics/ultralytics/yolo/utils/metrics.py at f2a7a29e531ad029255c8ec180ff65de24f42e8d · ultralytics/ultralytics. *GitHub*. https://github.com/ultralytics/ultralytics/blob/f2a7a29e531ad029255c8ec180ff65de24f42e8d/ultralytics/yolo/utils/metrics.py#L495

Utaminingrum, F. (2019). Alphabet sign language recognition using K-Nearest Neighbor Optimization. Journal of Computers, 63–70. https://doi.org/10.17706/jcp.14.1.63-70

Wilimitis, D. (2018, December 12). *The Kernel Trick in Support Vector Classification.* Medium. https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f

Wong, W. (2022, January 4). What is Residual Connection? - Towards Data Science. *Medium*. https://towardsdatascience.com/what-is-residual-connection-efb07cab0d55

World Health Organization. (2023, February 27). *Deafness and hearing loss*. WHO; World Health Organization: WHO. https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss

Yadav, S. (2023, April 29). *What is Kernel Trick in SVM? Interview questions related to Kernel Trick.* Medium. https://medium.com/@Suraj_Yadav/what-is-kernel-trick-in-svm-interview-questions-related-to-kernel-trick-97674401c48d

Yadav, S. (2023, April 29). What is Kernel Trick in SVM ? Interview questions related to Kernel Trick [Diagram]. Medium. https://medium.com/@Suraj_Yadav/what-is-kernel-trick-in-svm-interview-questions-related-to-kernel-trick-97674401c48d