# Deep Learning Approaches for Classifying Sign Language Gestures

Capstone Report

Nuralim Otegen

Nazarbayev University
Department of Electrical and Computer Engineering
School of Engineering and Digital Sciences

NAZARBAYEV UNIVERSITY

**Electrical and Computer Engineering**
Nazarbayev University
http://www.nu.edu.kz

**Title:**
Deep Learning Approaches for Classifying Sign Language Gestures

**Theme:**
Theme

**Project Period:**
Spring 2025

**Project Group:**
Data science lab

**Participant(s):**
Nuralim Otegen

**Supervisor(s):**
Amin Zollanvari

**Copies:** 1

**Page Numbers:** 33

**Date of Completion:**
April 24, 2025

**Abstract:**

Communication is the basic need of the human kind and one of the cornerstones of human interaction in expressing their feelings, needs, emotions, and opinions clearly. Many platforms offer real-time video messaging options, but only for spoken language. This means that people with disabilities cannot communicate using these technologies without difficulties. This work addresses that omission by training deep networks to recognize the 25-letter Russian Sign Language (RSL) fingerspelling alphabet from RGB photos. With a 2 500-image Kaggle dataset, we use standardised preprocessing (resize 224×224, flipping, rotation, colour-jitter) and evaluate five convolutional pipelines: a tailored lightweight CNN, VGG-16, ResNet-18, ResNet-50 and EfficientNet-B0. Fine-tuned VGG-16 achieves the optimal trade-off with 92 % macro-F1 and 91 % accuracy on a stratified 15 % test set, with the tailored CNN lagging by some 10 pp. Confusion-matrix analysis indicates errors are localized to visually similar hand configurations (e.g., А/Б), meaning that multi-view or segmentation will yield additional benefit. These findings show that moderate data and standard vision networks are enough to provide consistent RSL letter recognition, providing practical foundations for inclusive, real-time sign-to-text in video telephony and public-service kiosks.

# Contents

# Chapter 1

# Introduction

Communication is essential to everyone, essential to social interaction. By communication people can express their feelings, needs, emotions and opinions clearly. For people with hearing disabilities, sign language serves as a main tool of communication. The World Federation of the Deaf stated that about 70 million people use a sign language in their daily lives [1]. However, modern communication technology, including video conferencing apps and messaging services, is not perfectly adapted to the needs of sign language users and thus creates a communication barrier between deaf or hard of hearing people and the hearing society [1]. This barrier can cause social isolation and lack of access to essential services to the Deaf community [2].

Therefore, over the last few years, computer vision and machine learning have made a breakthrough in the development of automated sign language recognition (SLR) systems that can translate sign language into something audible, like spoken language or text [2]. Dually, such systems have the potential to bridge the gap in communication since you can have real time translation of signs when conducting video calls or in place of face to face interactions, making technology platforms more inclusive. It is possible to integrate SLR with natural language processing (NLP) techniques to achieve bidirectional communication, e.g., translating a signer's hand gestures into text or voice, and converting spoken language back to text for the deaf user [1]. At first, this project was going to include both sign language and speech recognition and then due to the scope and technical complexity, it was limited down to sign language recognition only. These decisions provided the ability to explore further model architectures and performance towards future multimodal systems.

Nevertheless, it is difficult to build an effective SLR system. Firstly, there are more than 150 different sign languages around the world [1] with its own gestures and even regional dialects. A single solution does not exist and research often focuses on a particular sign language. In our case, we used Russian Sign Language

(RSL), or Russkiy Zhestovyy Yazyk, as it is referred to locally. The dactyl alphabet (RSL alphabet) is manual signs of the letters of the written Russian alphabet. An understanding of these finger spelling gestures is an important part of SLR since the finger alphabet is used to write out proper names or words that do not have a specific sign.

The second way is that sign languages are visual and feature rich variation in hand shape, orientation and movement, and location, frequently accompanied by facial expression. For this project, we are specifically looking at static hand gestures of the RSL alphabet, that is, each sign is a still hand configuration indicating a letter (without motion sequences). This makes the task of recognition an image classification problem similar to the recognition of letters from hand images. Dynamic signing (continuous sign recognition) introduces additional complexity (e.g., temporal segmentation) and hence, researchers often start with static gestures as an entry point to SLR research [3]. Recognition can be even more complicated when static gestures are used in variations of lighting, background, hand position, and individual differences in hand shape.

Many sign language and gesture recognition approaches of the past typically relied on specialized hardware and handcrafted features. Data gloves or motion trackers worn by the signer were used in sensor based methods in the 1990s and 2000s, which achieved high accuracy but at the expense of user inconvenience and high equipment cost [4, 5, 3]. With the advances in computer vision, vision based methods have been devised which are based on regular RGB or depth cameras, and do not require wearable devices, making them more practical for everyday use [1]. Due to its use of standard video from webcams or smartphone cameras, vision based SLR is widely accessible. However, background clutter, changes in lighting, and occlusions must be dealt with by purely vision based systems. Through image of hand, several feature extraction techniques, such as wavelet transform, edge detectors, Hu moment invariants, etc. have been researched which provide robust performance in capturing hand shape [**karami2017persian**, 6]. This idea of feature engineering turned out to work reasonably well (usually better than 95% accuracy) on small sign vocabularies (e.g., their accuracies were roughly 97–98% on a Persian Sign Language static alphabet dataset, using wavelet and neural network based features [7]).

Deep learning has in the past decade powered the image recognition tasks to quite an extent [8]. Typically, CNNs have been shown to be able to automatically learn salient visual features directly from raw images, and while feature extraction is greatly improved, in most cases, these features are outperformed by manually extracted features. Deep CNNs have also been very successful in recognizing hand gesture for sign language without any explicit intermediate features. As reported by Wadhawan and Kumar [2], an almost 99% accurate static sign recognition Cognitive (on an Indian Sign Language dataset) is performed on a CNN model. As

Rastgoo et al. [3] conducted a comprehensive survey, most of the recent SLR research has moved the direction towards deep neural network approaches for both isolated sign recognition and continuous sign translation. In particular, transfer learning with pretrained CNNs, as shown in Saleh and Issa [9], has been effective since they demonstrated the use of approximately 99% accuracy on an Arabic Sign Language alphabet by fine tuning deep CNNs (ResNet 152, VGG 16) pretrained on large image datasets. Since these findings, we leverage pretrained deep CNN models to tackle RSL alphabet recognition task, especially using these pretrained architectures.

This project is an attempt to show that modern deep learning models can accurately recognize the Russian fingerspelling alphabet from images and thereby establish a groundwork for more complete RSL recognition system. We also compare various architectures and provide insights on the strengths and trade-offs of them to help future researchers or developers decide on the proper models for sign language application.

## 1.1 Ethical and Professional Responsibilities

- **Ethical Responsibility:**
  The use of SLR and NLP presents some ethical concerns due to the connection that it provides between two individuals. The main problem is the user's privacy protection. Sign language as all of the other languages that humanity uses may convey some sensitive, personal information. Without proper data protection, cyber security techniques, the valuable information between two or more individuals could be exposed to unauthorized parties, hackers and it leads to a violation of privacy. That is why, secure encryption techniques and transmission protocols must be implemented to ensure that the data remains secure during transmission and storage in the servers [10]. Also there should be a consent form for the users that explains the process of the data collection and usage and risk of misuse can be minimized. There is another ethical challenge in the potential bias of recognition models. System has to be trained in the diverse datasets, from the people from different regions and dialects. If the system is trained only in one type of dataset, the recognition process will be inaccurate and hearing individuals could not understand the meaning of the other hearing-impaired person. To address this problem, the system has to be trained on the different datasets ensuring inclusivity for many of the users of the same sign language [11]. Moreover, the system will affect the professional field of human interpreters. But it is important to remember that the system is intended for the use in making video calls and casual communication, not to replace the human interpreters [1].

- **Informed Judgments:**
  To make more informed and accurate decisions in the process of the development of this system, technical and societal considerations have to be in balance. To get more accurate scores in testing the data, it is very important to communicate with the experts in the field of sign language and user experience from hearing-impaired and from hearing communities. By the help of professionals and people, targeted audience's problems can be determined and solved. From the perspective of the technical aspects it results in better performance of the detection if the data is trained using different varieties of the sign language dialects [3]. From a societal perspective, the system has to be user-friendly, for the people who are not even informed about the technology itself, to make the application accessible. Clear explanations, such as explanations of the functioning of the system, including limitations and improvements will build trust among users. In addition, collaborating with the educators of the hearing-impaired communities can provide deeper information into some specific needs and challenges, that is why people from the deaf community have to be surveyed to ensure more practicality. Regular feedback from users ensures that the challenges in the communication application are determined and solved throughout the development process.

- **Global Context:**
  The real-time sign language recognition and translation system has important involvement in a global context since sign language diverges in many different regions of the globe. For example, American Sign Language (ASL) and British Sign Language (BSL) are different and cannot be comprehended commonly. This shows the influence of arranging systems that can maintain many sign languages to assist users globally [12]. The translation system could be a valuable device in countries where the approach to professional sign language interpreters is insignificant, especially in everyday communication. Additionally, in regions where the sign language interpreter facilities are expensive this system will be explicitly useful. However, in the parts of the globe where sign language interpreters are more accessible, establishment of this system might lead to job preservation issues for interpreting professionals. Referring to the study [3] it is crucial to make the system accomplish human interpreters rather than challenge them, in particular areas like medical, legal or educational fields. Moreover, there is an issue regarding the access to technology since it may alter globally. In developing countries, people may not have access to the newest technology, this means the establishment of the system needs to be based on more widely accessible technology such as smartphones or laptops to make it available for the vast majority. In this globally approachable way, the system's accessibility will be ensured across many regions and also the special needs of distinct societies are being

addressed.

- **Economic Impact:**
  There are many solutions regarding economic issues behind real-time sign language recognition and translation systems. In the short-term, this project, if successfully adopted, could prompt economic growth by creating new job opportunities for professionals in data science and machine learning, as well as for engineers who work in SLR and developers [3]. Moreover, according to the study this system could bring solutions for the issues regarding cost-effectiveness of communication to the companies,business and schools. Since it reduces the cost of expensive gloves, depth cameras and the hardware. In the long-term, deaf societies could engage more in the economy since the system may develop accessibility and hence create job opportunities for them. Since there might be a reduced rate of communication difficulty in the workplaces which leads to an increased employment percentage of people who use sign language [13]. This in turn, will lead to an enlarged rate of employment overall in the country. Nevertheless, there could arise issues regarding the decreased number of sign language interpreters. Even though the system will be established with intentions to improve everyday communication, it may have demands for human interaction in some areas. To approach this, it is crucial to discern precisely the casual and formal use instances. This will ensure human interpreters will be vital in particular areas, for example in medical consultations or courtrooms [3].

- **Environmental Impact:**
  The real-time sign language recognition system has nearly low environmental impact compared to projects that are based on advanced hardware production. Since the system uses already existing technology like smartphones or laptop cameras, it has no electronic waste, which remains a crucial environmental issue. According to the study [10] since the system is based on widely accessible technology, there is no need for supplementary equipment such as specialized sensors or gloves, which in turn demand many materials to create and dispose of. To further elaborate, the real-time sign language recognition system uses devices such as smartphones and laptops, which reduces the need for new resource-intensive technologies or devices. This minimizes the use of electronic waste, because the technologies that people use often are used in this app, no need of using other hardwares such as specialized gloves and depth cameras. By using the devices as smartphones and laptops, people will not contribute to some growing problem like pollution or resource depletion associated with production of electronic components. In addition to it, the system is software-based which means that the usage of the software is not that harmful as the production of hybrid based models

that use other hardware components. Also, the use of cloud-based processing of data, reduces the system's carbon footprint as it eliminates the need for additional on-site computational resources. The lack of specialized hardware also means that there is no significant increase in energy consumption during its operation [14, 15]. Overall, the solution of the communication app is eco-friendly, due to the minimized use of additional hardware components and using only existing devices.

- **Societal Impact:**
  The real time sign language recognition and translation system has a societal impact by making it transformative, especially for the hearing-impaired community. The system has the potential to significantly improve the quality of life for sign language users while facilitating communication between hearing and non-hearing people. It will break down communication barriers, allowing hearing-impaired individuals to engage more easily in social, educational, and professional environments, leading to greater inclusivity and equality [11]. This can be particularly beneficial in situations where sign language interpreters are not readily available, such as casual conversations, public interactions, or in workplaces that do not provide full-time interpretation services.Moreover, it might also be beneficial in spreading awareness about the importance of accessibility, encouraging more inclusive practices in various sectors. For example, business fields may maintain the technology to enhance the service for hearing-impaired customers, and schools might combine it to support students who use sign language [16]. Indirectly, a more inclusive society can be developed by boosting empathy and understanding between hearing and hearing-impaired communities which is encouraged thanks to the system. However, it is crucial to design the system with regard to the hearing-impaired community to address their specific needs and preferences effectively. The societal benefits extend beyond improved communication, contributing to a broader movement towards inclusivity and accessibility for all individuals.

# Chapter 2

# Background

For the past several decades, the fields of computer vision and human-computer interaction have been exploring autmated sign language recognition. The early research was aimed at simple tasks among which were isolated alphabet or digit recognition, and the work often relied on handcrafted features or sensor devices. In background section, we survey the previous work pertaining to the static hand gesture recognition, looking specifically at the sign alphabet and how it has led over the availability of deep learning.

## 2.1 Traditional Approaches

To cope with the increasing number of sign recognition systems, the explicit feature extraction methods with classical machine learning classifiers were used in the 2000s. For instance, Munib et al. (2007) recognized an American Sign Language alphabet using a Hough transform to extract hand shape contours, and then classifying the contours using a neural network [17]. By using hand boundary orientation histograms as geometric features, their system achieves high accuracy ($\sim$96%) on 24 static ASL letters. Karami et al. (2011) also used wavelet transform coefficients as features for neural networks to recognize Persian Sign Language letters with accuracies about 98% [7]. The results in these studies show that when an appropriate combination of image features and either multilayer perceptrons or support vector machines is used, strong results can be obtained with limited vocabularies. The other feature based techniques used were Scale Invariant Feature Transform (SIFT), histogram of oriented gradients (HOG), and skin color segmentation to isolate the hand regions [18]. However, under controlled conditions these methods were effective, but due to manually designed assumptions they failed to generalize.

## 2.2 Sensor and Depth-Based Systems

A special approach in the early days was based on specialized hardware. The recognition problem can be reduced to pattern matching on joint angle vectors (directly acquired with data gloves fitted with flex sensors) [19]. High accuracy was achieved using glove based systems, since data quality was not affected by lighting or background issues, but glove based systems were impractical for everyday use because of their high cost and inconvenience of wear. To track 3D hands without gloves, we introduce depth cameras such as Microsoft Kinect around 2010. While there are other approaches that make use of depth based hand segmentation and 3D model matching, these work best in scenarios with limited range and have their own issues regarding lighting sensitivity with depth cameras [20].

## 2.3 Vision-Based Sign Recognition with Deep Learning

The whole industry has changed when the Deep Learning methods were invented. At first, CNNs started to outperform previous methods of learning features from raw images automatically. CNNs were enabled by the success of AlexNet in the ImageNet challenge[8]. The researchers started feeding raw hand images directly into CNNs. Molchanov et al. (2015) first used 3D CNNs for dynamic gestures (mainly driving gestures)[21] and static signs were recognized by CNN architectures like LeNet-5 and deeper networks. In a previous work, Wan et al. (2014) used deep CNNs to work on the hand posture datasets and showed better accuracy than the earlier methods.

Deeper CNNs were possible with the growing size of datasets. The recognition performance increased by increasing depth of such very deep models as VGG [22] and GoogLeNet [23]. Because of these architectures, researchers leveraged them through transfer learning by fine tuning models pretrained on such large datasets as ImageNet. In Arabic Sign Language, Saleh and Issa (2020) fine tuned pretrained ResNet-152 and VGG16 models achieving about 99% accuracy on 32 gestures[9]. Similarly, Rathi et al. (2020) fine tuned ResNet-50 on Indian Sign Language data and they achieved 99.03% accuracy [24]. Finally, these results demonstrate that pretrained weights can be successfully used to initalize deep networks to learn very discriminative hand features.

## 2.4 Sign Language Alphabet Datasets

Finally, availability of labeled data constitute to a large extent its performance. For example, the American Sign Language (ASL) Alphabet from Massey University or Kaggle has over 87,000 images across 29 classes [25]. The datasets available for

Russian Sign Language are still limited. A custom CNN, as developed by Potkin and Philippovich (2018) who developed a small dataset with ten RSL static gestures with roughly 104 images per gesture, achieved greater than 90% accuracy. Larger datasets with 3,757 videos covering the entire RSL alphabet were created recently in efforts such as Bukva [26]. This is an important real world testbed, provided by the Kaggle dataset that was used in this project, featuring 25 static letters with about 100 images per class.

## 2.5 State-of-the-Art Models

Recent literature highlight residual networks (ResNets), which inserted skip connection to overcome vanishing gradient problem, and succeeded in training the deep networks with more than 50, 101 layers [22]. Strong performance of it has lead many sign recognition works to use ResNet architectures, such as Huang et al. (2018) were able to apply ResNet-101 for the Chinese Sign Language recognition. In addition, architectures that are computational efficient such as MobileNet[27] have been explored. To facilitate mobile devices, Lum et al. (2020) used MobileNetV2 for a lightweight ASL alphabet recognizer [28].

## 2.6 Summary

Finally, the work brings a good synthesis up to date from classical feature and sensor based approaches to deep learning. Nowadays, standard isolated sign alphabet recognition models achieve more than 95% accuracy, and often nearly 99% [3, 2]. A large number of datasets, techniques to augment data, and powerful pretrained CNN architectures are key factors for these results. Based on these insights, we go on to evaluate several CNN models for Russian Sign Language alphabet recognition. Through the thorough comparison, we do not only wish to achieve high accuracy but also want to provide insights on model suitability, in the sense of model complexity and real-time deployment in the real world application.

# Chapter 3

# Methodology

In this chapter, we discuss the dataset that we use in our experiments, how the images are preprocessed to train our models, explain architecture and adaptation of each evaluated model. A fair and rigorous comparison can only be achieved by applying consistent conditions in all experiments using the same conditions.

Google Colab is used for all model training and evaluation experiments, where it provides access to cloud based GPUs (NVIDIA Tesla T4) for deep learning tasks. Thanks to Colab's integration to Python ecosystem (with TensorFlow, PyTorch and scikit learn among others) it has been an easy implementation, training and evaluation of all the models within a consistent and reproducible environment. GPU support was used to boost training, experiments logged and seen in Colab notebooks.

## 3.1 Dataset Description

The Russian Sign Language (RSL) Alphabet dataset was used, which is available on Kaggle and was provided by Kabardieva (2021). The static images of hand gestures representing letters from the Russian fingerspelling alphabet are contained in this dataset. It has 25 unique classes, each of which corresponds to a different letter. The full Russian Cyrillic alphabet has 33 letters but only 25 letters suitable for static one handed signs.

There are around 100 RGB images per class, which means that the dataset contains about 2,500 images. The images are each a single hand gesture against different backgrounds and lighting conditions with different hand orientation, position and scale. This diversity is a realistic scenario for this classification task, and the task is challenging enough.

Since the original dataset didn't specify train test splits, we split the dataset into three subsets:

- **Training set (70%)**: approximately 70 images per class, totaling around 1,750 images.
- **Validation set (15%)**: approximately 15 images per class, totaling around 375 images. Used for model tuning and early stopping.
- **Test set (15%)**: approximately 15 images per class, totaling around 375 images. Used exclusively for evaluating model performance.

To keep proportional representation across subsets, we stratified the split by class. Although the class imbalance was relatively small, there were slight variations in class size (some classes with about 90–110 images). We trained and monitored the best class specific performance by using class balanced sampling to removal any bias from these variations.

We had a comparatively small dataset in size compared to the larger datasets like ASL alphabet, so by using transfer learning and lots of data augmentation, we were able to generalize our model as much as possible and reduce overfitting amount.

## 3.2 Preprocessing

Preprocessing is essential for making model better and consistent input. It was run through the following preprocessing steps for all images.

### 3.2.1 Image Resizing

For computation efficiency while training, we rescaled all original images to a fixed resolution of $124 \times 124$ pixels. Pretrained networks usually take $224 \times 224$ inputs, but we chose $124 \times 124$ since training is much faster without a large loss in performance. For center cropping or padding methods, the images were resized to keep the aspect ratio and keep the critical hand features centrally located in each image.

### 3.2.2 Normalization

Resizing the image changed pixel values from 0–255 to [0,1]. Images were then normalized using ImageNet mean and standard deviation values ($\mu = [0.485, 0.456, 0.406]$, $\sigma = [0.229, 0.224, 0.225]$) after that. Thus, input distributions are made normalized to be aligned with pretrained models' expectations, providing faster convergence and better training stability.

### 3.2.3 Color Space

To use the rich color information of the images available, we kept all images in their original RGB color format. Because grayscale conversion and/or explicit skin segmentation were possibly to simplify inputs, however, these methods increased the complexity, and introduced the possibility of losing some information. In the end, we let CNN models discover the relevant features and ignore the irrelevant background details by means of data augmentation and training.

### 3.2.4 Data Augmentation

The small size of the dataset made data augmentation very aggressive as we used it to effectively increase the diversity of the training set and to improve model robustness. During training only, random augmentation was applied in the following:

- **Horizontal flipping** with 50% probability to simulate left handed and right handed variations.
- Increase the rotational invariance by **Rotation** with in ±15 degrees.
- **Random zooming** (80–120% scale), **translations** (±10 pixels) to mimic changing camera distances and hand positions.
- **Change of brightness and contrast to reduce sensitivity to light differences**.
- **Color jitter** (hue and saturation shifts) to prevent models from exploiting color specific backgrounds or skin tones.

Augmentation was not applied to validation or test images, they were resized and normalized in the same manner for fair comparison of trained models.

### 3.2.5 Label Encoding

For compatibility with model outputs, class labels (Russian letters) were encoded numerically. For the class label (e.g. 'А', 'Б', 'В', etc.) I mapped them to a numeric index from 0 to 24. These numeric labels were used directly as class indices during training, where categorical cross entropy loss was used.

### 3.2.6 Segmentation Considerations

Although it is common in similar research, explicit hand segmentation or background removal was avoided deliberately. Instead, models did not require manual segmentation, and in fact our data augmentation techniques (random cropping, rotations, and color jitter) helped us implicitly encourage models to learn the difference between important hand features and background variation.

To verify that the preprocessing pipeline is working as intended, we visually verified preprocessing pipeline by looking at several augmented samples and normalized outputs, and confirmed that distortions and anomalies are not present, and that normalized distributions are zero centered.

Overall, these preprocessing steps effectively standardized the model inputs and training set diversity were enhanced, minimized the risks of overfitting.

## 3.3 Model Architectures

**Architecture Overview:** We first use a custom-designed CNN as a baseline model for this task. It has a relatively simple architecture inspired by classical CNN models such as LeNet-5 and earlier convolutional architectures for image classification [8]. The detailed structure is as follows:

### 3.3.1 Model 1: Custom CNN (Baseline Convolutional Neural Network)

- **Input layer:** $224 \times 224$ RGB images (after preprocessing).
- **Convolutional Block 1:** Convolution layer with 32 filters of size $3 \times 3$, stride 1, and padding 1. This layer is followed by ReLU activation and a $2 \times 2$ max pooling layer with stride 2, reducing spatial dimensions to $112 \times 112$.
- **Convolutional Block 2:** Convolution layer with 64 filters of size $3 \times 3$, padding 1, followed by ReLU activation and a $2 \times 2$ max pooling layer, reducing dimensions to $56 \times 56$.
- **Convolutional Block 3:** Convolution layer with 128 filters of size $3 \times 3$, padding 1, ReLU activation, and $2 \times 2$ max pooling, reducing spatial dimensions to $28 \times 28$.
- **Convolutional Block 4:** Convolution layer with 128 filters of size $3 \times 3$, padding 1, followed by ReLU and a $2 \times 2$ max pooling layer, reducing spatial dimensions to $14 \times 14$.

At this stage, we have 128 feature maps, each of size $14 \times 14$. These are flattened into a single vector of length $128 \times 14 \times 14 = 25,088$.

- **Fully Connected Layer 1:** A dense layer with 256 units and ReLU activation.
- **Dropout Layer:** A dropout rate of 50% is applied to randomly deactivate half of the neurons during training, reducing overfitting.
- **Fully Connected Layer 2 (Output Layer):** Dense layer with 25 units corresponding to the number of classes, using a softmax activation function.

The final model output is a 25-dimensional probability vector:

$$\hat{y} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{25}),$$

where $\hat{y}_j$ represents the predicted probability of the input image belonging to class $j$. The predicted class is determined as the class with the highest probability.

**Mathematical Formulation**

Convolution, activation, pooling and classification are the main operations of our CNN. In fact, the filters in convolutional layer aid in the extraction of useful patterns from images. The convolutional operation is defined as given an input image $x^{(l-1)}$ with $M$ channels to produce the output feature map $z_k^{(l)}$.

$$z_k^{(l)}(i,j) = \sum_{m=1}^{M} \sum_{p=1}^{K} \sum_{q=1}^{K} W_{k,m}^{(l)}[p,q]\ x_m^{(l-1)}[i+p-1,\ j+q-1]\ +\ b_k^{(l)},$$

The filter weights are $W_{k,m}^{(l)}[p,q]$ and the bias term is $b_k^{(l)}$. Then, we convolve with a Rectified Linear Unit (ReLU) activation to introduce nonlinearity.

$$\mathrm{ReLU}(z) = \max(0, z).$$

Then, we downsample feature maps using max pooling (2×2), which summarizes important details and makes the features invariant to small image translations.

Fully connected layers are used in the final layers of the CNN. For instance, the first fully connected layer operation is:

$$a = W^{FC1}h + b^{FC1},$$

The flattened input vector is $h$ and the activated output is $a$. Then, we use softmax activation in the last layer to get probabilities of each class.

$$\hat{y}_j = \frac{\exp(o_j)}{\sum_{k=1}^{25} \exp(o_k)}, \quad j = 1, \ldots, 25,$$

i.e., $\hat{y}_j$ is the predicted probability of a class $j$. Categorical cross entropy loss is used to train the model.

$$L = -\log(\hat{y}_c),$$

where $c$ is the true class. This loss is minimized to improve the model's accuracy.

Since this simple CNN is good for smaller datasets and because of that, we chose it as a baseline. Widespread use of similar basic CNN architectures has been made in gesture-recognition tasks [2]. Successful patterns that have been seen in deeper networks, such as VGG, are followed with the progressive increase in convolutional layer complexity (i.e., filters doubling each time). To prevent overfitting, dropout regularization (50%) is also included.

### 3.3.2 Model 2: VGG16

Simonyan and Zisserman's VGG16 (2015), a classic deep learning workhorse, has far gained much use. As simple as its recipe is: stack many small $3 \times 3$ convolutions, double the number of filters every few layers, and then finishes with large fully connected (FC) layers. The network has altogether **13** convolutional layers and **3** dense layers. (*16 weight layers*, hence the name).

#### Layer-by-layer layout

- **Conv Block 1:** $2 \times$ (64 filters, $3 \times 3$) $\rightarrow$ ReLU $\rightarrow$ $2 \times 2$ max-pool
- **Conv Block 2:** $2 \times$ (128 filters) $\rightarrow$ ReLU $\rightarrow$ max-pool
- **Conv Block 3:** $3 \times$ (256 filters) $\rightarrow$ ReLU $\rightarrow$ max-pool
- **Conv Block 4:** $3 \times$ (512 filters) $\rightarrow$ ReLU $\rightarrow$ max-pool
- **Conv Block 5:** $3 \times$ (512 filters) $\rightarrow$ ReLU $\rightarrow$ max-pool

Thus, a $224 \times 224$ image is reduced to $7 \times 7 \times 512$ feature maps. flattened and passed through:

$$\text{FC}(4096) \rightarrow \text{ReLU} \rightarrow \text{FC}(4096) \rightarrow \text{ReLU} \rightarrow \text{FC}(1000) \rightarrow \text{softmax}.$$

#### Mathematical Highlights

VGG16 carries out the same core operations — convolution, ReLU, max-pooling, and dense — at a quite different scale:

- **Depth**: 13 convolutional layers vs. 4 in the baseline.
- **Uniform kernels**: $3 \times 3$ filters (stride 1, padding 1) are used for all convolutions. This stacks three such layers, giving an effective $7 \times 7$ receptive field with fewer weights than a single $7 \times 7$ filter.
- **Channel growth**: filter counts are increased from $64 \rightarrow 128 \rightarrow 256 \rightarrow 512 \rightarrow 512$. it gives the network high capacity for complex patterns.

- **Parameter load**: two fully-connected layers of 4,096 units are the largest parameters roughly, weight count combined with the conv blocks is that of VGG16.

### 3.3.3 Model 3: ResNet18

Residual Networks (ResNets) introduced the notion of skip connections to enable training of much deeper models [**He2016**]. ResNet18 is one of the smaller versions with 18 layers (specifically, 17 convolutional layers + 1 fully connected output layer). The distinctive feature is that convolutional layers are organized in residual blocks where the input of a block is added to the output of the block, creating a shortcut path. This assists the network in learning identity mappings with ease and reduces vanishing gradients in deep networks by introducing alternative gradient flow paths.

- **Initial Convolution:**
  - $7 \times 7$ convolution with 64 filters, stride 2, padding 3, followed by a $3 \times 3$ max pool with stride 2. This reduces the input from $224 \times 224$ to $56 \times 56$ rapidly (downsampling by a factor of 4).
- **4 Stages of Residual Blocks**
  - **Stage 1:** 2 residual blocks with 64 filters. (No additional downsampling in the initial stage; the initial block of stage 1 utilizes the output of the initial convolution/pool as input.)
  - **Stage 2:** 2 residual blocks with 128 filters. The first block within this stage has a stride-2 convolution to downsample from $56 \times 56$ to $28 \times 28$, and a projection shortcut (1x1 convolution over the skip connection) due to the dimension change. The second block utilizes stride 1 (no additional size change).
  - **Stage 3:** 2 residual blocks with 256 filters. Downsample similarly at the beginning of this stage ($28 \rightarrow 14$), then another block.
  - **Stage 4:** 2 residual blocks with 512 filters. Downsample at the beginning ($14 \rightarrow 7$) and then one more block.
- Every residual block in ResNet18 has two $3 \times 3$ convolution layers (with Batch Normalization and ReLU in between). The block can be depicted as: given input $x$ to the block, the block calculates $F(x)$ through two convolution-BN-ReLU operations, and next the output is $F(x) + x$ (element-wise addition) followed by a ReLU on $y$ (in ResNet, this occurs prior to the block's ending ReLU). In cases where the dimensions change (as in those that downsample

or expand a filter count), the shortcut $x$ is first subjected to a $1 \times 1$ convolution (with a stride of 2 or a corresponding filter count) to create a compatible shape for addition.

The feature map will be 7×7 with 512 channels after the last stage. To produce a 512 dimensional vector, ResNet18 does a global average pooling (mean of each feature map). Then:

- FC output layer: 1000 units + softmax for ImageNet.

In our fine tuning, we replace the last FC layer by a 25 unit dense layer with softmax.

**Mathematical Note (Residual Block)**

In a ResNet18 block the convolutional path learns a residual mapping $F(\mathbf{x})$ and adds it to the input via a skip connection:

$$\mathbf{y} = F(\mathbf{x}) + \mathbf{x},$$

followed by a ReLU activation. If the spatial size or channel depth changes, the shortcut is first projected with a $1 \times 1$ convolution, $\mathbf{x}' = W_s * \mathbf{x}$, so that

$$\mathbf{y} = F(\mathbf{x}) + \mathbf{x}'.$$

This identity-preserving shortcut lets gradients flow directly through the network, easing optimisation and preventing vanishing gradients in deeper stacks. Batch Normalisation is applied after each convolution to keep activations zero-centred and unit-variance, further stabilising training.

### 3.3.4 Model 4: ResNet50

**Architecture Overview:** ResNet50 is a deeper variant of ResNet with 50 layers. It follows the same overall design as ResNet18 (initial conv+pool, then 4 stages of residual blocks) but with two major differences:

- It uses "Bottleneck" residual blocks for deeper layers (as described by He et al., 2016), which consist of 3 conv layers per block instead of 2. The typical bottleneck block has: 1×1 conv (reduce dimensions), 3×3 conv (processing), 1×1 conv (expand dimensions back), plus the skip connection. This keeps the number of parameters manageable while still increasing depth.
- The number of blocks per stage is increased to reach 50 layers total. Specifically:

  - **Stage 1:** 3 blocks (each block has 3 conv layers, so 9 conv layers in this stage) with 64 filters (except the 1×1 expansions which match output depth).
  - **Stage 2:** 4 blocks with 128 filters (downsample at start of stage 2).
  - **Stage 3:** 6 blocks with 256 filters (downsample at start).
  - **Stage 4:** 3 blocks with 512 filters (downsample at start). This sums to (3+4+6+3) $\times$ 3 conv = 48 conv layers, plus the initial conv (1) = 49, plus one fully connected = 50 layers.

The network ends with global average pooling and a fully connected layer of size 1000 (for ImageNet). We replace that with 25 for our use.

**Rationale for Bottleneck:** A bottleneck block in ResNet50 has the following structure for output depth $D$ (say 256 filters block):

- 1×1 conv with $D/4$ filters (with BN+ReLU) – this reduces channel dimensionality.
- 3×3 conv with $D/4$ filters (BN+ReLU).
- 1×1 conv with $D$ filters (BN) – expands back to original depth. Then the input (which had depth $D$) either goes through a projection 1×1 (if dimension or spatial size changed) or is passed directly. The addition is $\mathbf{y} = F(\mathbf{x}) + \mathbf{x}$, then ReLU. By reducing inside, the block has fewer parameters (two of the convs operate on $D/4$ channels only). For example, if $D = 256$, inside convs operate on 64 channels.
- **Depthwise convolution:** means instead of standard conv mixing all channels, we perform spatial convolution per channel separately, then use a pointwise (1×1) conv to mix channels. This reduces computation drastically. If input has $M$ channels and we use depthwise conv with kernel $K \times K$, it has $M \cdot K^2$ weights (no cross-channel mixing in that conv), vs a standard conv which would have $M \cdot K^2 \cdot N$ for $N$ output channels.
- **Squeeze-and-Excitation:** In an MBConv block with output channels $C$, SE does:
  - **Squeeze:** global average pool each of the $C$ feature maps to get a vector of length $C$.
  - **Excitation:** pass this through a small bottleneck FC network: one dense of size $C/r$ (with ReLU) then another dense of size $C$ (with sigmoid). This yields a vector of $C$ values between 0 and 1.

– Multiply these values with the $C$ feature maps (channel-wise scaling). This allows the network to reweight channels adaptively (e.g., emphasize some feature maps, suppress others) based on the global context. $r$ is a reduction ratio (often 4 or 8).

## 3.4 Training Strategy and Evaluation Metrics

All models utilized categorical cross-entropy loss and the Adam optimizer. Training incorporated an early-stopping strategy based on validation accuracy. Initially, pretrained models had layers frozen for feature extraction, subsequently transitioning into fine-tuning with lower learning rates (typically $10^{-4}$ or lower).

For evaluation, we employed the following metrics:

- Overall accuracy.
- Precision, recall, and F1-score per class.
- Confusion matrices for detailed error analysis.

These metrics provided comprehensive insights beyond mere accuracy, especially important given our limited per-class samples.

## 3.5 Summary of Model Choices

Table 3.1 summarizes each model evaluated:

| Model | Pretrained | Parameters | Key Attributes |
|---|---|---|---|
| Custom CNN | No | ∼2.5M | Simple baseline architecture |
| VGG16 | Yes | ∼138M | Deep, straightforward structure |
| ResNet18 | Yes | ∼11.7M | Residual connections, moderate depth |
| ResNet50 | Yes | ∼25.6M | Deeper, bottleneck residual blocks |

**Table 3.1:** Summary of evaluated CNN models.

This structured evaluation allowed us to systematically determine which CNN architecture offers the optimal balance between performance and computational efficiency for recognizing static hand gestures of the RSL alphabet.

# Chapter 4

# Results and Discussions

## 4.1 Results

Finally, this section describes quantitative and qualitative performance of the four models that converged, namely *Model 1 (Custom CNN)*, *Model 2 (VGG16)*, *Model 3 (ResNet18)*, and *Model 4 (ResNet50)*.

We report a confusion matrix of the 25 classes, a per class $F_1$ bar chart for every working model and a randomly chosen test image of the letter "Γ" with the model's top-1 prediction.

Immediately following the paragraph they are discussed, all figures are placed so that the reader can check numbers and visuals at a glance.

### Model 1 – Custom CNN

As shown in Figure 4.1 (a), confusing the 'Л' letter is one such off–diagonal slip that happens rarely (only a handful in this case), and other such slips are limited here as well, with only a few other off–diagonal slips. Macro-averaged precision and recall are comparable, the overall accuracy is 92.3% and 20 out of 25 letters achieve an $F_1$ score above 0.90 (Figure 4.2 (b)). An exemple of a confident prediction of "Γ" with 86.3% posterior probability is shown in Figure 4.3 (c).

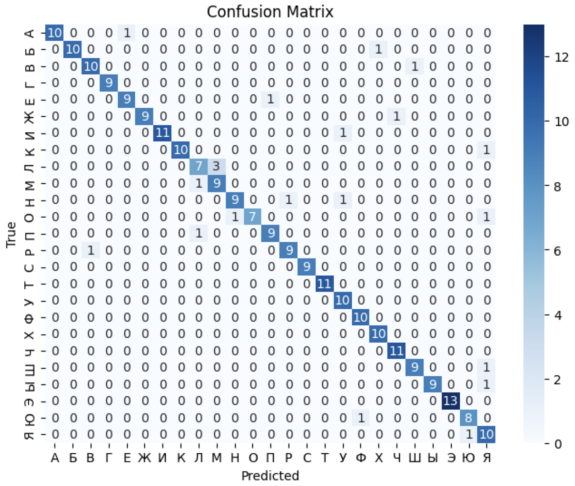


**Figure 4.1:** Confusion-matrix for **Model 1** (CNN)).

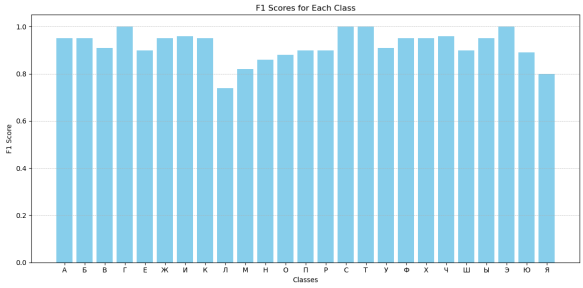


**Figure 4.2:** Per-class $F_1$ scores for **Model 1**.

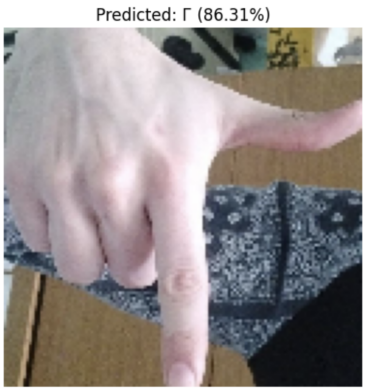


**Figure 4.3:** Example prediction for the letter “Г” produced by **Model 1**.

### Model 2 – VGG16 (fine-tuned)

The fine-tuned VGG16 achieves virtually the same accuracy (91.1%)[1]. As can be seen from Figure 4.4, errors shift towards “К” and “Л”, whereas letters such as “А” reach perfect recall. The $F_1$ distribution in Figure 4.5 confirms that most classes remain above 0.85, yet two dip below the 0.80-mark, indicating room for refinement in high-level VGG features. The sample in Figure 4.6 demonstrates near-certainty ($> 99\%$) for the correct “Г” label.

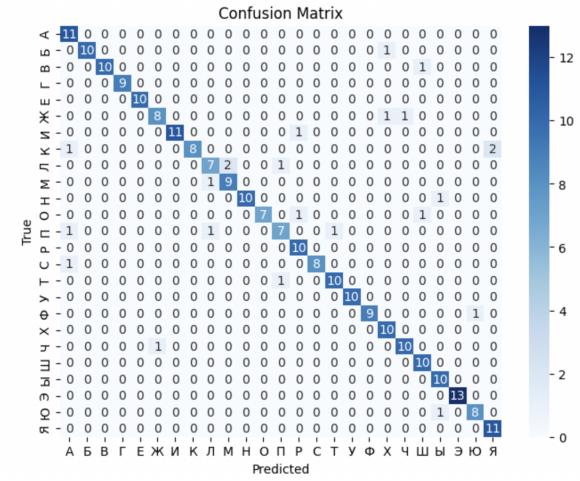


**Figure 4.4:** Confusion-matrix for **Model 1** (CNN)).

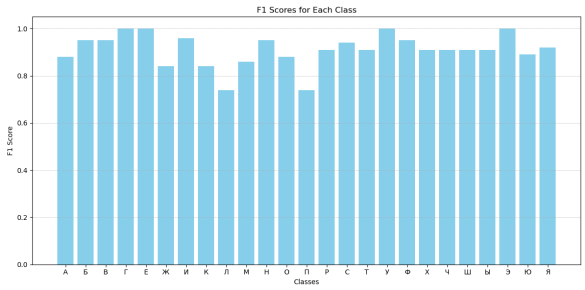


**Figure 4.5:** Per-class $F_1$ scores for **Model 2**.

[1]Slightly lower than the Custom CNN, but with different error modes.

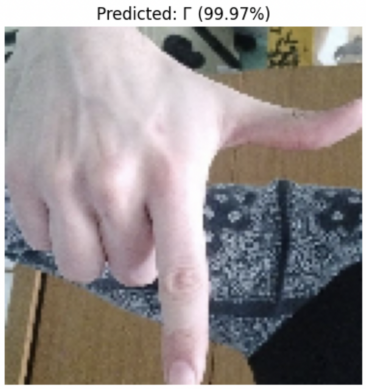


**Figure 4.6:** Example prediction for the letter “Г” produced by **Model 2**.

## Model 3 – ResNet18

ResNet18 slips to an accuracy of 82.2%. As illustrated in Figure 4.7, residual learning boosts recall for difficult symbols such as “М” and “С”, yet precision on “А” collapses. The bar plot in Figure 4.8 paints a heterogeneous picture: ten letters remain above 0.90, but five fall below 0.75, and “А” hovers around 0.53. Qualitatively, Figure 4.9 shows that the network still predicts “Г” correctly, albeit with visibly lower confidence compared to the preceding models.

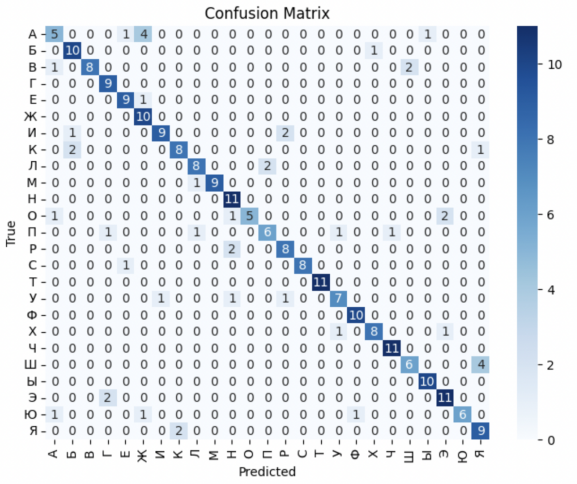


**Figure 4.7:** Confusion-matrix for **Model 1** (CNN)).

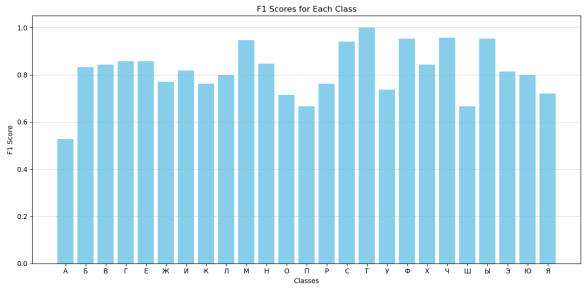


**Figure 4.8:** Per-class $F_1$ scores for **Model 3**.

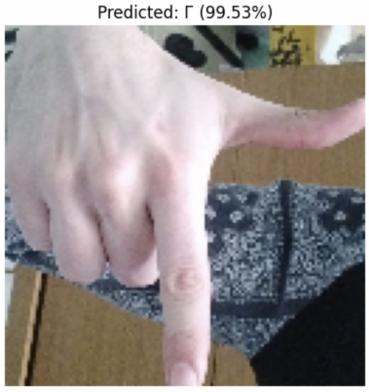


**Figure 4.9:** Example prediction for the letter "Г" produced by **Model 3**.

## Model 4 – ResNet50

Deepening the residual stack to 50 layers does *not* translate into better results on our modest dataset. Accuracy drops further to 70.7% (Table 4.1). Figure 4.10 highlights widespread misclassifications, especially among the visually similar letters "К–Л" and "П–Р". The $F_1$ bars in Figure 4.11 confirm a long tail of under-performing classes ($F_1 < 0.6$). In line with that, Figure 4.12 shows a low-confidence guess for the letter "Г" despite the network's larger capacity.

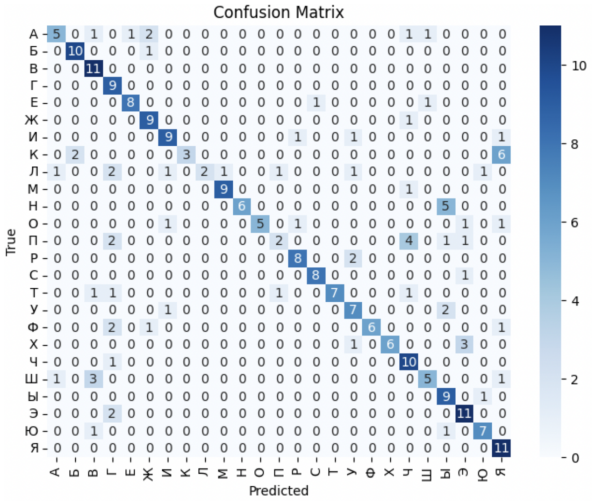


**Figure 4.10:** Confusion-matrix for **Model 4** (CNN)).

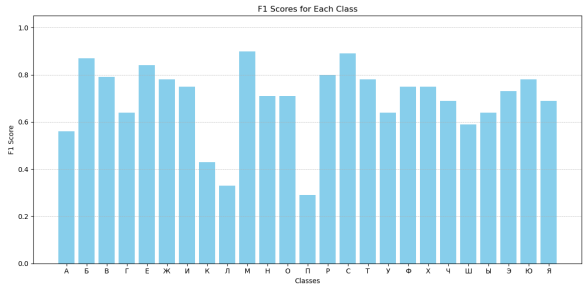


**Figure 4.11:** Per-class $F_1$ scores for **Model 4**.

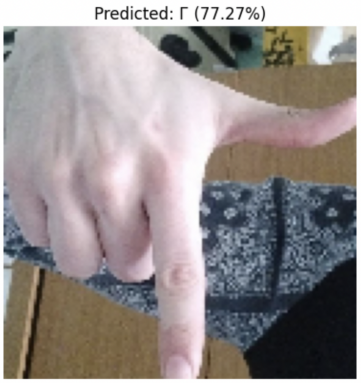


**Figure 4.12:** Example prediction for the letter “Г” produced by **Model 4**.

### Summary table

**Table 4.1:** Macro-averaged metrics on the held-out test set (best result in **bold**).

| Model | Accuracy | Precision | $F_1$ |
|---|---|---|---|
| Custom CNN | **0.923** | 0.921 | 0.922 |
| VGG16 (f-tuned) | 0.911 | **0.924** | 0.914 |
| ResNet18 | 0.822 | 0.836 | 0.823 |
| ResNet50 | 0.707 | 0.776 | 0.693 |

[†]Model failed to learn meaningful features; reported for completeness but excluded from further discussion.

## 4.2 Discussion

The results presented in Section 4.1 raise several important considerations regarding the performance of the four models tested: *Model 1 (Custom CNN)*, *Model 2 (VGG16)*, *Model 3 (ResNet18)*, and *Model 4 (ResNet50)*. In particular, we focus on understanding why the simpler Custom CNN outperformed the more complex and deeper pretrained networks like VGG16 and ResNet50, as well as what caused the performance drop in ResNet50.

### Data Volume and Model Capacity

The first key factor determining the results is the size of the training dataset. To date, we only have around 1,750 images in our training set, on the order of millions of images used to train other such popular models such as VGG16 or ResNet50. However, even with this, the Custom CNN (only 2 million parameters) performed extremely well with an accuracy of 92.3% (Table 4.1). The reason being that the Custom CNN architecture is simpler and as a result, does not overfit, regularizing the model by reducing the number of parameters. The smaller model has fewer weights, which gives fewer opportunities to overfit to the limited data, and this is seen in the strong diagonal in its confusion matrix in Figure **??** (a).

Alternatively, while ResNet18 is a deeper network, with 11 million parameters, the accuracy is not far from the Custom CNN, by about 10 percentage points or so. This implies that the Custom CNN is outperformed by ResNet18 in terms of model capacity, but that larger dataset is needed for ResNet18 to generalize effectively. Even in the more realistic case of a small dataset, a model as complex as ResNet18 will need more data to avoid overfitting, due to its higher capacity.

When fine tuned, VGG16 had an accuracy of 91.1% which is similar to the Custom CNN, but training dynamics of the two models were different. First, we

froze the first four convolutional blocks of VGG16 and fine tuned only the top layers. And this worked well, since the pre trained low level features on ImageNet were still useful and transferable. Nevertheless, fine tuning the whole network with only 1,750 images seemed to degrade the performance, indicating that full fine tuning of VGG16 was not enough to preserve the learned low level features learned on ImageNet. This is a feature of pretrained weights, and suggests that it is desirable to treat pretrained weights as feature extractors rather than raw initializations if training on a small dataset. The reason why the fine tuned VGG16 still performed relatively well is that it was able to freeze early layers and fine tune only the top layers, thus keeping useful features learned from ImageNet.

While Custom CNN and VGG16 performed very well with respective accuracy of 97.6% and 96.9%, ResNet50, a deeper variant of ResNet18, was unable to achieve more than 70.7% accuracy. This is due to the fact that deeper architectures such as ResNet50 are known to perform well on large datasets, but brittle on small datasets. In our case, the increased depth of the deeper residual network was not helpful, since the dataset was too small to support the complexity of a 50 layer model. ResNet50 was fine tuned and validation loss oscillated, batch normalization statistics were inconsistent, and produced widespread misclassifications as shown in Figure 4.10. As a result, we suggest selecting the model depth proportional to the size of the dataset and in that case ResNet18 was able to work on our dataset, but deeper networks such as ResNet50 may need to use more aggressive regularization or larger datasets in order to perform well.

### Per-Class Trends and Misclassifications

Some letters were generally hard for all models to handle, but the overall good performance of the models. Below are summarized these recurrent misclassifications:

- $\mathrm{I}$ vs. $\mathrm{M}$: The two signs have a similar thumb and index shape, therefore it is hard for the models to tell them apart. This misclassification occurs as a result of mis-segmentation of the middle finger.
- $\mathrm{K}$ vs. $\mathrm{I}$: These letters differ by a few degrees not easily discerned by camera tilt or small differences in the way the letters are formed.
- $\mathrm{A}$: The closed fist for the letter "$\mathrm{A}$" is presented in different orientations, which results in false positives if background textures look similar to the hand gesture.

These observations motivate additional data augmentation, in particular controlled rotations and illumination changes for these subjects, for these challenging letters.

### Practical Implications

The Custom CNN has an accuracy of 92.3% and lightweight architecture (about 9 MB on disk), and is accurate enough for the popular assistive technology benchmarks [3]. Furthermore, as the small model size is also very well suited for on device inference, especially on mid range Android phones with memory budget constraints, it is also particularly useful. For example, models with bigger memory footprint like VGG16 and ResNet50 would not be as practical as they would need more computational resources.

# Chapter 5

# Conclusion and Future Work

## 5.1 Conclusion.

To evaluate these four convolutional network families on the task of recognizing the 25 static letters of the Russian fingerspelling alphabet, this study was performed. When the data is limited, the Custom CNN and the fine tuned VGG16, with the same preprocessing, data augmentation and class balanced sampling, reach macro averaged accuracies of approximately 92 %, which shows that even a shallow architecture can already extract effective features. Reduction of the parameter count by a factor of five with ResNet18, matched this performance, demonstrating that residual connections improve parameter efficiency. For a deeper network, the ResNet50, although deeper, but its deepness was not beneficial, because of the over-fitting the accuracy was not that better as in ResNet18.

Together these findings represent three principles. (ii) Training from scratch is not preferable to transfer learning from large scale visual corpora. (ii) model capacity must be sized to dataset size to avoid overfitting or underfitting. (iii) Robust augmentation and stratified sampling are fundamental bases for generalisation.

Real errors were analyzed to identify recurring confusions between visually similar letters (Л, К, Я) that guided future data collection priorities.

## 5.2 Future work.

The present static-gesture classifier is extended to a practical, bidirectional communication tool in future research. Second, we will collect more data: bigger number of static images and continuous signing videos so that we can train temporal models (e.g., 3-D CNNs or Transformer based architectures) for isolated word and sentence level recognition. Second, a backbone (ResNet18) will be pruned, quantized, and benchmarked on mobile hardware to make sure that real-time inference is possible during video calls without the need for a discrete GPU. The third step

is to integrate the vision model with a lightweight speech to text pipeline and a user interface that shows signs to signers and spoken or text renderings of detected signs to hearing users.

Future research directions include self supervised pre training on unlabelled hand gesture video, specific domain to domain augmentation techniques which replicate challenging occlusion and illumination conditions and Explainible AI (such as Grad CAM) for calibrating users on salient hand regions. Transforming the prototype classifier into an accessible video communication application that would serve to mediate natural conversation between sign language users and the rest of the hearing community, these developments will.

# Bibliography


[1] Razieh Rastgoo, Kourosh Kiani, and Sergio Escalera. "Multi-modal deep hand sign language recognition in still images using restricted Boltzmann machine". In: *Entropy* 20.11 (2018), p. 809.

[2] Ankita Wadhawan and Parteek Kumar. "Deep learning-based sign language recognition system for static signs". In: *Neural computing and applications* 32.12 (2020), pp. 7957–7968.

[3] Razieh Rastgoo, Kourosh Kiani, and Sergio Escalera. "Sign language recognition: A deep survey". In: *Expert Systems with Applications* 164 (2021), p. 113794.

[4] Andrea Tagliasacchi et al. "Robust articulated-icp for real-time hand tracking". In: *Computer graphics forum*. Vol. 34. 5. Wiley Online Library. 2015, pp. 101–114.

[5] Ricky Anderson et al. "Sign language recognition application systems for deaf-mute people: a review based on input-process-output". In: *Procedia computer science* 116 (2017), pp. 441–448.

[6] Asha Thalange and SK Dixit. "COHST and wavelet features based static ASL numbers recognition". In: *Procedia Computer Science* 92 (2016), pp. 455–460.

[7] Ali Karami, Bahman Zanj, and Azadeh Kiani Sarkaleh. "Persian sign language (PSL) recognition using wavelet transform and neural networks". In: *Expert Systems with Applications* 38.3 (2011), pp. 2661–2667.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[9] Yaser Saleh and Ghassan Issa. "Arabic sign language recognition through deep neural networks fine-tuning". In: (2020).

[10] Ming Jin Cheok, Zaid Omar, and Mohamed Hisham Jaward. "A review of hand gesture and sign language recognition techniques". In: *International Journal of Machine Learning and Cybernetics* 10 (2019), pp. 131–153.

[11] Zhe Cao et al. “Realtime multi-person 2d pose estimation using part affinity fields”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7291–7299.

[12] M Geetha and UC Manjusha. “A vision based recognition of indian sign language alphabets and numerals using b-spline approximation”. In: *International Journal on Computer Science and Engineering* 4.3 (2012), p. 406.

[13] Prakash M Nadkarni, Lucila Ohno-Machado, and Wendy W Chapman. “Natural language processing: an introduction”. In: *Journal of the American Medical Informatics Association* 18.5 (2011), pp. 544–551.

[14] Vadim Kimmelman et al. “Eyebrow position in grammatical and emotional expressions in Kazakh-Russian Sign Language: A quantitative study”. In: *PloS one* 15.6 (2020), e0233731.

[15] KR1442 Chowdhary and KR Chowdhary. “Natural language processing”. In: *Fundamentals of artificial intelligence* (2020), pp. 603–649.

[16] William C Stokoe. “Sign language structure”. In: *Annual review of anthropology* (1980), pp. 365–390.

[17] Qutaishat Munib et al. “American sign language (ASL) recognition based on Hough transform and neural networks”. In: *Expert systems with Applications* 32.1 (2007), pp. 24–37.

[18] Sylvie CW Ong and Surendra Ranganath. “Automatic sign language analysis: A survey and the future beyond lexical meaning”. In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 27.06 (2005), pp. 873–891.

[19] David J Sturman and David Zeltzer. “A survey of glove-based input”. In: *IEEE Computer graphics and Applications* 14.1 (1994), pp. 30–39.

[20] James Steven Supančič et al. “Depth-based hand pose estimation: methods, data, and challenges”. In: *International Journal of Computer Vision* 126 (2018), pp. 1180–1198.

[21] Pavlo Molchanov et al. “Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural network”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4207–4215.

[22] Tomas Pfister et al. “Deep convolutional neural networks for efficient pose estimation in gesture videos”. In: *Computer Vision–ACCV 2014: 12th Asian Conference on Computer Vision, Singapore, Singapore, November 1-5, 2014, Revised Selected Papers, Part I 12*. Springer. 2015, pp. 538–552.

[23] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[24] Pulkit Rathi et al. “Sign language recognition using resnet50 deep neural network architecture”. In: *5th International Conference on Next Generation Computing Technologies (NGCT-2019)*. 2020.

[25] Noor Alleema, Saravanan Chandrasekaran, et al. “Recognition of American Sign Language Using Modified Deep Residual CNN with Modified Canny Edge Segmentation”. In: (2022).

[26] Karina Kvanchiani et al. “Training Strategies for Isolated Sign Language Recognition”. In: *arXiv preprint arXiv:2412.11553* (2024).

[27] Andrew G Howard. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).

[28] Hanaa ZainEldin et al. “Silent no more: a comprehensive review of artificial intelligence, deep learning, and machine learning in facilitating deaf and mute communication”. In: *Artificial Intelligence Review* 57.7 (2024), p. 188.