Exploring Deep Learning Models for Handwritten Bengali Compound Character Classification: A Comparative Study

by

Md Iftekhar Hossain
23241157

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
BRAC University
October 2025

Declaration

It is hereby declared that

1. The thesis submitted is my own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

MD Iftekhar Hossain
23241157

Approval

The thesis titled "Exploring Deep Learning Models for Handwritten Bengali Compound Character Classification: A Comparative Study" submitted by

1. Md Iftekhar Hossain (23241157)

Of Fall, 2025 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on October 10, 2025.

Examining Committee:

Supervisor:
(Member)

Mr. Annajiat Alim Rasel
Senior Lecturer
Department of Computer Science and Engineering
BRAC University

Thesis Coordinator:
(Member)

Dr. Md Golam Rabiul Alam
Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Dr. Sadia Hamid Kazi
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Abstract

Recognizing isolated *juktoborno* in Bangla script, which consists of 334 unique compound characters, remains a challenging problem in Optical Character Recognition (OCR). This research explores the efficacy of deep learning-based approaches by comprehensively evaluating nine state-of-the-art architectures: **EfficientNet-B0, MobileNet V3, DenseNet-121, ConvNeXt V2 Tiny, ViT-Small-DINOv3, Custom CNN, ResNet-50, VGG-16, and SqueezeNet**.

We employed transfer learning using ImageNet pre-trained weights, fine-tuning all models on the **MatriVasha Dataset** containing **120 compound character classes** with **306,461 grayscale images** at 128×128 resolution. The dataset was partitioned into 70

All models were trained using AdamW optimizer with cosine annealing learning rate schedule, data augmentation (RandomAffine, RandomPerspective, RandomErasing), and early stopping on high-performance hardware (dual RTX 5090 GPUs). **EfficientNet-B0** achieved the highest accuracy of **97.68%** with only 4.16M parameters, demonstrating superior efficiency. **MobileNet V3** secured second place at **97.01%** while being the fastest to train (5.7 minutes), and **DenseNet-121** achieved **96.40%** with effective feature reuse.

This study contributes to advancing Bangla OCR technology through comprehensive architecture comparison and establishes new performance benchmarks for compound character recognition. The findings will help in practical applications like **document digitization, automated translation, text-to-speech conversion, and search-based text retrieval**, with EfficientNetB0 recommended for maximum accuracy and MobileNet V3 for optimal speed-accuracy balance in production deployment.

Table of Contents

Chapter 1

Introduction

Bengali, the official language of Bangladesh and one of the high frequency languages in the Indian subcontinent has a rich and complex writing system, which poses special challenges to Optical Character Recognition (OCR) systems [4], [16]. The existence of juktoborno, or the combined character composed of basic consonant and vowels in the Bangla script is one of such issues. [6], [12]. Proper identification and recognition of juktoborno is the key to successful implementation of OCR system in Bengali [8], [18].

OCR is very crucial in many areas, such as digitization of documents, automated translation, speech-to-text emergence, and information search [1], [24].Nevertheless, the successful understanding of Bangla text, particularly when it is related to juktoborno, is an important challenge [3], [7]. Current OCR systems find it hard to recognize and manipulate such compound characters and hence cause errors and imprecision in the text recognition process [5], [20].

This study aims to solve this problem by coming up with an effective juktoborno detection system. The successful implementation of this research will have wide-ranging benefits, including the digitization of Bangla literature, preservation of cultural heritage, improved accessibility for individuals with visual impairments, and enhanced language processing capabilities in various applications [27], [37].

In order to reach these goals, the given thesis paper provides detailed analysis of the problem statement, establishes the research goals, conducts the detailed literature review, and presents the working plan of the project implementation. Using the information about the available literature and investigating the methods of OCR and machine learning algorithms, as well as deep learning structures [19], [39], this research attempts to create a specialized juktoborno detect system, which will possess the capability to identify the necessary information with high precision.

It is hoped that the suggested solution will play a great role in improving the OCR technology employed to work with Bangla language and also making OCR systems adaptable to various real world uses [36], [38]. By making more true to reality, effective identification of Bengali text, the created juktoborno recognizing system will assist in getting over the issues related to the compound characters, and finally, OCR technology will become more useful and efficient in the Bengali-speaking com-

munity [28], [30].

Chapter 2

Problem Statement

It is a serious challenge to the OCR systems due to the presence of juktoborno in the Bangla script. Current OCR systems frequently fail to recognize and work with such compound characters correctly and, as such, they introduce errors and inaccuracies in textual recognition. Thus, the juktoborno detection system which can process these compound characters with a high degree of precision and efficiency is needed.

2.1 Research Objectives

The primary goals of the study will be the following:

- To examine the nature and complications of juktoborno in the Bangla script.
- To identify the available methods of OCR and algorithms to detect juktoborno.
- To develop and apply a new juktoborno detection algorithm based on machine learning or deep learning techniques.
- To monitor the work of the proposed system with the help of relevant metrics and benchmark datasets.
- To compare the results with the available OCR systems and evaluate the efficiency of the developed solution.
- To offer guidelines that could be used to improve on and conduct future research in the field of Bangla OCR.

Chapter 3

Detailed Literature Review

The Bengali script concerns more than 200 million individuals throughout the planet, and it poses exceptional issues to automatic character recognition mechanisms [2], [14]. Unlike most other scripts, there are also numerous characters in Bengali that are compounded with combining two or more basic characters [3], [6]. This leads to the complex structural variations of these compound characters and the sheer amount of combinations [8], [12] that can be made that make the recognition task especially challenging. In the last fifteen years, scientists have achieved a lot in this field, leaving conventional feature-based techniques and advanced deep learning-based techniques [19], [33].

3.0.1 Early Approaches: Traditional Machine Learning Methods (2009-2014)

The original work on Bengali compound character recognition was based on the use of handcrafted features and classic machine learning classifiers [5], [7]. The key issues with identifying handwritten compound characters were first mentioned in 2009, when Bhattacharya et al. suggested that the complexity of the mentioned characters could also require an incremental approach [18]. The early research was aimed at the study of the structural properties of the compound characters and the methods to extract meaningful characteristics out of them [20]. In 2010, Bhowmik et al. made a contribution where Multi-Layer Perceptron (MLP) and Support Vector Machine (SVM) classifiers were used to identify both simple and compound Bengali characters [3], [28]. They developed topological features of three kinds: longest-run features, modified shadow features, and octant-centroid features. This work had significance since it demonstrated that to a certain degree, the intricate arrangements of compound characters could be captured in the considerately designed features [16]. Nevertheless, the precision of the results was still low as handcrafted features were not able to record all the differences in characters of a handwriting [4].
Thus, a step forward was made in 2013 with Sarkar et al. suggesting their paper titled "Recognition of Bangla compound characters using structural decomposition" published in Pattern Recognition journal [4]. This method reduced compound characters to simple shape elements by the use of specially made decomposition rules [7]. The approach made use of topological characteristics and importantly did not need to be trained using actual instances. This was very impressive at the time, since the

reliance on substantial training datasets was minimized [5]. The decomposition approach aided the performance of recognition by simplifying the complex characters to be broken down into their parts [8].
One important breakthrough was the development of the CMATERdb benchmark database by Das et al. in 2014 in their article entitled "A benchmark image database of isolated Bangla handwritten compound characters" published in the International Journal on Document Analysis and Recognition [2]. This database fulfilled a very important requirement of research community for standardized datasets [14]. In a comprehensive survey of more than 2 million Bengali words, Das and his associates found about 334 compound characters of the Bengali script, of which 171 categories constituted pattern shapes of their own [12]. The CMATERdb 3.1.3.3 database contained 55,278 isolated images of compound characters belonging to 199 pattern shapes [2]. The data collected in this database was through three methods and split into training and testing sets. It became a benchmark database with which other recognition strategies can be compared [18]. The researchers achieved a recognition rate of 79.35% on 171 character categories with the help of this database and convex hull and quadtree-based features and a SVM classifier [20]. This accuracy rate was moderate but it was the state of the art at that time and formed a reference point over which better things would come in future [16].

3.0.2 Transition Period: Early Deep Learning Experiments (2017-2019)

The years 2017 to 2019 witnessed a major shift in the approach to Bengali compound character recognition [1], [19]. Researchers started exploring deep learning methods, particularly Convolutional Neural Networks (CNNs), which promised to learn features automatically instead of depending on handcrafted ones [10], [13]. In 2017, Pal et al. published "Bengali handwritten character recognition using deep convolutional neural network" where they demonstrated one of the first attempts at using deep CNNs for Bengali handwritten characters [19]. The study showed that these models could learn useful features through kernels and local receptive fields [1]. The automatically learned features followed by densely connected layers for classification demonstrated better results than traditional approaches [17].

Roy et al. made a really significant breakthrough in 2018 with their paper "Handwritten Isolated Bangla Compound Character Recognition: a new benchmark using a novel deep learning approach" published in Pattern Recognition Letters [28]. What made this work stand out was the achievement of just 9.67% error rate, which translates to around 90.33% accuracy, on the CMATERdb dataset [2]. When you compare this with the 79.35% accuracy from traditional methods only four years before, the improvement is quite remarkable [3]. The researchers put their deep convolutional neural network head-to-head with traditional models like Support Vector Machines, and the deep learning approach came out clearly ahead [33]. What this really showed was that deep networks had the capability to deal with the complexity of compound characters in ways that earlier methods simply couldn't match [25].

Around the same time, there were interesting experiments happening with shape decomposition techniques combined with modern approaches [7], [29]. One study from

2017 specifically looked at using shape decomposition for handwritten compound characters in Bangla OCR, acknowledging that these complex shapes were still giving researchers a hard time [27]. The idea was to take what worked from traditional decomposition methods and merge it with newer classification techniques [30]. Another noteworthy development was BornoNet, which Purkaystha et al. introduced in 2018 [19]. This CNN-based model managed to get some pretty impressive numbers: 98% accuracy on CMATERdb, 96.81% on the ISI dataset, 95.71% on BanglaLekha-Isolated, and 96.40% when tested on a mixed dataset [14]. What caught attention here was how they did cross-validation - training the model on one dataset and then testing it on completely different ones. This really demonstrated how well deep learning models could generalize beyond their training data [34].

In 2019, Fardous and Afroge explored using CNNs with data augmentation techniques for Bengali handwritten character recognition [30]. Data augmentation became an important strategy to artificially increase the training dataset size and improve model robustness [22]. The AIBangla benchmark dataset was also introduced during this time, adding more resources for the research community [38].

3.0.3 Modern Era: Advanced Deep Learning Architectures (2020-2024)

The period from 2020 onwards has seen increasingly sophisticated deep learning architectures being applied to Bengali compound character recognition [37], [39]. In 2020, Hasan et al. published "Bangla Compound Character Recognition by Combining Deep Convolutional Neural Network with Bidirectional Long Short-Term Memory" where they proposed combining deep Convolutional Neural Networks with Bidirectional Long Short-Term Memory networks (CNN-BiLSTM) [24]. This combination was particularly clever because CNNs could extract spatial features from character images while BiLSTM could capture sequential dependencies [23]. The hybrid model was evaluated on the CMATERdb compound character dataset and showed improved performance over pure CNN approaches [2].

Recent work in 2024 has introduced even more advanced techniques [37], [39]. The BNVGLENET model, presented by Mia et al., applied hypercomplex neural networks with hierarchical class expansion for Bangla handwriting recognition [38]. This was a method of recognizing the difficulty of the large number of character in Bengali classes, and also of the compounds [36]. The hierarchical growing of classes strategy aided in dealing with the complexity of identifying the hundreds of various patterns of characters [32]. The researchers have also presented CBD2023 dataset, which is specifically designed [31].

One more 2024 contribution was a study by BanglaNet by Rahman et al. which employed ensembling of Convolutional Neural Networks [28], [34]. Ensemble techniques use a combination of the predictions of several models in order to enhance accuracy and strength as a whole [35]. This was the prevailing tendency of employing more sophisticated architectures and devoid of single models in order to wring out more performance improvements [33], [36].

As of 2024, practical uses of OCR have also received attention, and research has investigated specialized models and other advanced techniques on document types [27], [37]. These publications acknowledge the fact that the character recognition of compounds is not an academic activity only, but has practical uses in the digitization of Bengali documents, text-to-speech systems, and the facilitation of the application of different language technologies [38], [39].

3.0.4 Evolution of Datasets

In the transformation of this, the creation of extensive datasets has been a key factor. The CMATERdb database, which was presented by Das et al. in 2014, was still the major reference when it comes to character recognition of compounds. Nonetheless, a number of other datasets have been created to target various factors of the issue. The ISI dataset gave more handwritten samples, whereas BanglaLekha-Isolated, which was developed by Biswas et al., offered another distribution on the character samples. Ekush dataset by Rabby et al. added an all-purpose and all-type comprehensive database. The diversity was added by the AIBangla and NumtaDB datasets of accessible training and testing statistics. The latest CBD2023 data set that was presented by Mia et al. offered a hypercomplex set that was specially intended to take up hierarchical class expansion methods.

3.0.5 Key Technological Shifts

The literature indicates that there are a number of significant changes in methodology over time. Firstly, it has been observed that there has been a shift in manually developed features to features developed using deep learning. Second, the accuracy rates have rapidly increased, with the conventional methods at the range of 79-80% and the deep learning-based methods at the range of 90-95% and even 98-99% in certain architecture and dataset combinations. Third, scholars have shifted aside the straightforward, single-model frameworks, into sophisticated frameworks, such as hybrid frameworks, ensemble modeling, and hierarchical strategies.

3.0.6 Conclusion

Bengali compound character recognition and detection have seen tremendous development in the last fifteen years. Since the inception of structural decomposition and handcrafted features getting about 80 percent accuracy on small datasets, the science has progressed to highly advanced deep learning solutions that get higher than 95 percent accuracy on full benchmark datasets. Standardization of databases such as CMATERdb has played a significant role in the facilitation of comparison between various approaches in a systematic way. The recent studies are driving towards even greater rates of accuracy as well as tackling more practical issues such as computational efficiency and applicability. Nevertheless, despite these developments, the recognition of Bengali compound characters is an active field of research where open problems persist especially in dealing with the complete repertoire of handwritten scripts and the complete set of 334 compound characters.

Chapter 4

Model Overview

4.0.1 Custom CNN

Our Custom CNN was designed to identify Bengali handwritten letters [25], [33]. Rather than a generic architecture, we examined what was effective in other systems of complex writing such as Chinese, Japanese, Korean and Arabic, and had modified them to the use of the complicated strokes of the Bengali script and the compound characters [1], [19].

Our model is based on the residual learning which essentially assists in training deeper networks without encountering the gradient problems [10]. Our design was inspired by ResNet-v2 and incorporated a pre-activation design - that is, batch normalization and the ReLU activations are performed prior to the convolution operations [10]. This arrangement assists the gradients in flowing more easily with training, which is truly critical when you are attempting to instruct the model to discern the Bengali characters that bear an extremely close similarity to one another [28], [30].

The network begins with a basic one - two 3x3 convolutional layers with 32 filters each, then it is followed by a batch normalization and ReLU activation [1]. We were very careful with this first bit, not to downsample aggressively since we had to conserve all the little touches that will give the intricate curves and modifiers of Bengali script [38].

Then there are four remaining blocks of data that progressively get more complex - that is adding 64, 128, 256 and finally 512 channels [10]. The blocks consist of several residual units and at the start of block 1-4, we downsample to diminish spatial dimensions as it stretches out the visible range of the network [17]. The residual units operate by using the input in two 3x3 convolutions, though there is a shortcut connection that does not go through these layers [10]. When the dimensions don't match up we apply 1x1 convolutions in the shortcut to get everything compatible [13].

The Squeeze-and-Excitation (SE) blocks that we have inserted in the wake of every residual block is one thing that makes this architecture stand out [22]. They have been successful in the Chinese and Japanese OCR, so we figured they would be helpful here, as well [17], [37]. The recalibration of the significance of various feature channels is what SE blocks do. The first is to squeeze the spatial information by global average pooling followed by the second which is to run it through a small network (reduction ratio of 16) to determine which channels are most important [22]. The resulting weights (scaled between 0 and 1 with sigmoid) weigh every feature map, giving more importance to useful channels and fading away less relevant channels [36]. In the case of Bengali characters, this comes in very handy since certain angles and curves of the strokes are much more greater to identify than others [27], [39].

Spatial Attention modules were also introduced in blocks 2 and 3, similarly to CBAM (Convolutional Block Attention Module). These do not work the same way - they consider the location of the important stuff in the image. They do both average and max pooling, mix them along the channel dimension, and apply 7x7 convolution with sigmoid to make attention maps. This assists the network to concentrate on such things as where strokes intersect, edges of the character and where modifiers are located - all in determining similar Bengali characters apart.

To regularize, we applied the dropout which increases gradually - 15 percent at the beginning of block 1 and 30 percent at the beginning of block 3. We also employed 2D spatial dropout which randomly selects out a complete feature map rather than a pixel. This forces the network to acquire more robust features rather than relying overly on the particular channels.

After convolutional is accomplished, we do something interesting to the end 512-channel feature maps - we do global max pooling and global average pooling simultaneously. The average pooling provides the general patterns whereas max pooling records of the strongest signals. The sum of these two results in a 1024 dimensional feature, that is, a vehicle that encompasses the various dimensions of the character.

The classification component consists of three layers linked together that reduce a set of 1024 neurons to 512 to the ultimate quantity of classes (as again, 50 to 120+ character kinds that we are identifying). There are batch normalization, ReLU activation, and dropout (45% and 30%) in the first two layers to maintain the balance and avoid overfitting. The final layer gives predictions of each Bengali character classes.

As an initialisation method, we have employed Kaiming (He) normal distribution in all of the convolutional and linear layers - this is common when using ReLU activation in the network [10]. The weights in a batch normalization begin with 1

and the biases begin with 0, thus it commences near an identity transformation.

All in all, the model includes approximately 13-15 million parameters, placing it between lightweight mobile networks and heavy models such as ResNet-50. This dimension is suitable to our datasets (usually 50,000-200,000 samples) - it is large enough to capture all the complex distributions of the Bengali characters but not so large that it is hard to train.

In short we have used a number of well-known methods - residual connections to assist in training, squeeze-and-excitation to assist channel attention and spatial attention to assist emphasizing on significance areas, and multi-scale pooling to enhance feature capture all in consideration of Bengali hand writing recognition. It was to borrow what has been known to work in other complicated scripts and scale it to some modern deep learning practices to achieve something that works well in Bengali OCR.

4.0.2 ResNet-50

ResNet-50 is a deep Convolutional Neural Network (CNN) which is designed to address the vanishing gradient problem that is faced when training very deep networks [10]. Residual networks have an introductory name in field of computer vision, and we know how ResNet-50 has become famous by their treatment in deep architectures without loss of accuracy in training [29], [34].

Residual learning is at its core of ResNet-50, that is why the network can use the information, which flows through its layers, to retain essential features [10]. Unlike traditional deep networks, ResNet-50 retains the skip connections, which essentially are shortcut connections [17]. These connections enable integer to jump few intermediate layers and add to output layer directly, thus helping in smooth propagation of gradient in back propagation [10]. By doing this, the network can train more deeply without the danger of degrading more [13].

ResNet-50 is deeper in the number of layers than usual CNN architectures (50 layers) [10]. All in all, each block consists of three convolutional layers, and it is mainly composed of bottleneck residual blocks [35]. The first convolutional layer in the block employs a 1×1 kernel to decrease the channels, lowering the computational complexity [13]. Secondly, a 3×3 convolution is used on top of the second layer to spatially feature extract, and then it is followed by a 1×1 convolution to recover the original depth [10]. Every layer is followed by batch normalization to stabilize training and ReLU activation to introduce non-linearity to the network, allowing it to learn more complex patterns [1], [33].

Layer (Type)	Output Shape	Param #
resnet50 (Functional)	(None, 5, 5, 2048)	23,587,712
flatten (Flatten)	(None, 51200)	0
dense (Dense)	(None, 512)	26,214,912
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 24)	6,168

Table 4.1: Resnet-50 Model Summary

Total Parameters: 499,40,120 (190.51 MB)
Trainable Parameters: 263,52,408 (100.53 MB)
Non-trainable Parameters: 235,87,712 (89.98 MB)

Both for efficient feature extraction and since max pooling squeezes much more spatial information for less receptive field, we first run with an initial 7×7 convolutional layer, then a max pooling layer. It gradually augments the network with several residual blocks, where the identity mappings enable the model to learn only residual transformation rather than new representation. In this way, this method greatly increases the efficiency of learning, and it prevents later layers from losing all the meaningful information they may have acquired from the previous stages.

At the last stage of ResNet-50, the global average pooling layer was used instead of the fully connected layer with numerous parameters. In practice, this technique substantially reduces computational overhead with high accuracy. In the case of the ImageNet dataset, the final output layer usually contains 1000 neurons with softmax activation function for classification. While, this can be modified according to specific use cases, such reducing the number of output neuron for a classification task of a different class prediction.

For instance, ResNet-50 has become a benchmark core tool, showing that deeper networks don't necessarily have to suffer from problems of degradation. It has a very robust and good capability of generalizing to various computer vision tasks such as image classification, detection, segmentation. It is an efficient and powerful model for dealing with complex visual data, because of deep feature extraction, skip connections and bottleneck layers that characterize ResNet-50, which is a core architecture in modern deep learning.

4.0.3 EfficientNetB0

EfficientNetB0 is a radical shift in neural network architecture design with its methodical model scaling. Rather than randomly changing network depth, width and resolution, EfficientNetB0 operates on a compound scaling mechanism which uniformly scales all three dimensions using a simple but highly effective compound

coefficient. This will ensure that the network maintains an ideal trade off between efficiency in which it operates and its accuracy.

EfficientNetB0 is based on the fact that it uses mobile inverted bottleneck convolution (MBConv) blocks, popularized by MobileNetV2. These blocks apply depthwise separable convolutions which have high 2-level folding lowering significantly the amount of computation required yet preserving the representational capacity. All MBConv blocks are pointwise expanded to enlarge channel dimensions, after which spatial features are extracted by depthwise convolution, and finally pointwise projection is used to reduce channels to the target output dimension. This architecture replenishment allows the network to acquire rich feature representations with the constraint of a small number of parameters.

One of the aspects that distinguish EfficientNetB0 is that it incorporates the squeeze-and-excitation (SE) modules into the MBConv blocks. These attention processes dynamically control channel-wise responses of features by explicitly modeling channel interdependences. SE module performs global average pooling to gather channel wise statistics and after that, it uses two fully connected layers with non-linear activations to generate channel attention weights. These weights are scaled on the original feature maps, which allow the network to selectively highlight informative features while suppressing lesser useful ones.

It begins with the architecture of 3x3 kernel stem convolution, and then continues with seven major blocks of various configurations. There are various MBConv layers at each block with varying expansion ratios and kernel sizes, which were optimized via neural architecture search. The network continuously adds more channels and decreases the spatial dimensions with strided convolutions, maintaining the same computational efficiency. Traditional ReLU is replaced with swish activation functions which provide smoother gradients which enhance training dynamics and final model performance.

EfficientNetB0 employs batch normalization in between convolutional layers, to stabilize the training process and accelerate convergence. A 1x1 convolution is used at the final stage of the network to upsample the features to 1280 through a global average pooling and a final fully connected layer with dropout as a regularization method. The final layer, when trained on ImageNet, contains 1000 neurons, but can be easily changed to a specific classification problem through changing the dimensions of the final layer.

The efficiency in deep learning that EfficientNetB0 is based on a compound scaling methodology has established new standards in efficiency. The model is more accurate in terms of parameter-to-parameter ratios than earlier architectures by co-optimizing depth, width, and resolution. This efficiency enables EfficientNetB0 to be especially

useful in situations of deployment with limited computational resources, and yet remain competitive in complex visual recognition settings.

4.0.4 MobileNetV3

MobileNetV3 is a mobile and edge-based neural network. It is a hardware-aware search of neural architecture that finds optimal model structure. It is a search that is a blend of computerized search and human intelligence. The outcome is a model that is accurate and fast enough on devices that have limited resources.

Mobile inverted residual bottleneck block forms the center of MobileNetV3. This block is a better design than MobileNetV2. It applies depthwise separable convolutions to reduce the cost of computation. Such convolutions divide a regular convolution into two components, a depthwise step and a pointwise step. The input channels are first expanded using a 1×1 convolution. A depthwise convolution is then used to extract spatial features. Lastly, there is another 1×1 convolution that reduces the features to a smaller space. This arrangement formulates a bottleneck that supports the network to study compressed and meaningful representations.

The hard-swish activation is also presented in MobileNetV3. Swish activation is made easier as hard-swish. It provides gradients that are smooth and have high accuracy but are faster on mobile hardware. In deeper layers where it is most needed, it has a hard-swish network. In older versions, it relies on the generic ReLU to be efficient. The squeeze and excitation (SE) modules of MobileNetV3 are also reconfigured to be used on mobile devices. These modules possess lesser expansion ratio and this makes them more accurate without slackening it down. The network puts SE modules to the best locations where they are most useful. This produces an attention mechanism which makes the model concentrate on the most significant channels.

The last phase of the architecture is new as well. A 1×1 convolution is used in most networks to expand features and then global pool the features. MobileNetV3 uses global average pooling then 1×1 convolutions. This basic modification saves the latency up to 15 percent without loss in the accuracy. It demonstrates that it is significant to design according to special hardware.

The network starts with a 3×3 convolution. After this, a series of bottleneck blocks follow. The architecture search found the best configuration for each block, including its expansion ratio, kernel size, and activation function. As data flows through the network, channels increase and spatial dimensions decrease. Skip connections are present when the input and output dimensions of a block are the same.

The network head is also designed for efficiency. It includes a 1×1 convolution, a

global average pooling step, and two fully connected layers. A hard-swish activation sits between the two fully connected layers. The final layer is a linear classifier. For a model pretrained on ImageNet, this layer has 1000 outputs. We can easily adapt the model for other tasks by changing this final layer.

4.0.5 DenseNet121

A novel connectivity pattern is employed in DenseNet121. A dense block has a connection between each layer to the previous layers. Such a structure alters the flow of information across the network. It establishes the early-to-late layers connections and this assists in gradient flow and also makes the network re-use the features.

The dense block is the largest block of DenseNet121. This block uses feature maps of the past layers concatenated, and not summed as in ResNet. The input of every layer is the merged outputs of all the previous layers. It then generates a fixed number of new feature maps, which is referred to as growth rate. In this design, the l-th layer receives feature maps of all the l-1 layers ahead of this design. The growth rate of 32 in DenseNet121 governs the amount of information that each layer contributes.

DenseNet121 has layers of transition between the dense blocks. Such layers decrease the amount of channels and downsample the spatial dimensions. A transition layer first applies batch normalization. Subsequently, a 1×1 convolution is used to reduce the number of features in the feature map. Lastly, a 2×2 average pooling operation reduces the spatial size by half. The number of channels that are reduced is determined by a compression factor, which is typically 0.5. This ensures that the model does not become overly complicated.

DenseNet121 has a 7×7 convolution with a stride of 2 as the initial step. This is followed by batch normalization, ReLU activation, and 3×3 max pooling with a stride of 2. This initial stage reduces the spatial scale and extracts low-level features. Thereafter, there are four dense blocks with 6, 12, 24, and 16 layers respectively. The network consists of a total of 121 convolutional layers. Each dense block maintains the same spatial size. Spatial dimensions are only reduced in the transition layers.

The bottleneck design is applied in every layer within a dense block. This architecture consists of batch normalization, ReLU activation, and a 1×1 convolution that creates 4k feature maps, where k represents the growth rate. This is followed by another batch normalization, ReLU, and a 3×3 convolution producing k feature maps. The bottleneck architecture is computationally efficient in that it reduces the number of input feature maps for the expensive 3×3 convolutions.

There are several advantages in the dense connection pattern. It enhances gradient

flow and promotes feature reuse, making the model parameter-efficient. Each layer can access features from all levels of complexity, so it does not need to relearn them. This design also helps to reduce overfitting, especially on smaller datasets.

DenseNet121 uses batch normalization before each convolution to stabilize training. It also uses ReLU activations for non-linearity. The network preserves features at many levels of abstraction by concatenating them. This creates a rich, multi-scale representation that is useful for many visual tasks.

The network ends with a global average pooling layer and a fully connected layer for classification. This design avoids the need for multiple fully connected layers, which reduces the number of parameters and the risk of overfitting. When pretrained on ImageNet, the final layer has 1000 outputs. We can adapt this for other classification tasks.

4.0.6 ConvNeXtV2Tiny

ConvNeXtV2Tiny is a state of the art re-implementation of convolutional neural networks that uses the architectural learnings of Vision Transformers without sacrificing the efficiency and inductive bias of CNNs. This architecture reveals that with current training methods and architectural improvements, pure convolutional networks can be as competitive or even more competitive than transformers.

ConvNeXtV2Tiny is built upon its ConvNeXt blocks, which are traditional ResNet blocks that are modernized by methodically improving them following transformer inspired architecture. The blocks use large kernel depthwise convolutions, normally 7×7, that enhance the receptive field just like the global attention in transformers. These depthwise convolutions work in isolation per channel which is efficient in spatial feature extraction and is computationally efficient. The application of larger kernels is a reversal of the tendency of smaller kernels which proves that bigger receptive fields are beneficial to modern architectures.

One of the major innovations in ConvNeXtV2Tiny is the Global Response Normalization (GRN) layer that adds to the variety of features and advances its quality of representation. GRN calculates statistics on channels and uses a learnable channel-specific transformation that promotes the competition and diversity of features. It avoids feature collapse and encourages the learning of different and complementary features across channels, which is a weakness in the original ConvNeXt design.

The network makes use of an inverted bottleneck architecture in which the hidden dimension within the blocks is four times the size of the input dimension. Unlike in the traditional bottlenecks, this design extends processing features to the block prior to projecting the features at the original dimension. The expansion is done in the form of a pointwise convolution and then the large kernel depthwise convolution, activation and a final pointwise convolution to project. This behavior is similar to that of the feed forward network design of transformers, where dimension of hidden

dimensions are usually larger than dimensions of models.

ConvNeXtV2Tiny makes use of LayerScale, which is a learnable per-channel scaling used on the output of every block prior to the residual connection. This method, which is initialized with small values, enhances the stability of training of deep networks and it allows the application of more effective optimization of the large kernel convolutions. The smooth startup and scaling plans across the network allow the training dynamics to be stable despite the added complexity of the modern components of the architecture.

The architecture is arranged in four phases with consecutively growing channel sizes and declining spatial resolutions. The tiny variant makes use of channel dimensions of 96, 192, 384, 768, with 3, 3, 9, 3 blocks respectively. The stages start with a downsampling block downsampling the spatial dimensions by half and changing the channel dimensions. This pyramid of features forms a feature hierarchy that records the information on different scales useful in a variety of downstream tasks.

The network is initiated by a patchify stem which applies 4×4 convolution with stride 4 to transform input images into patches, like Vision Transformers. Such vigorous downsampling of the initial layer decreases the computational expense of the latter layers and still ensures that the spatial resolution is high enough to enable good learning of features. The stem is followed by layer normalization, which gives stable feature normalization which has shown to be useful in transformer architectures.

ConvNeXtV2Tiny also uses GELU activation functions rather than ReLU, and this gives smoother gradients, which result in better optimization. The network has fewer activation functions and normalization layers than regular CNNs, having a single activation and normalization per block. It is a transformer-inspired design simplification that saves the computational resources but does not give up representational power.

The network ends with global average pooling and linear classification head. The last layer normalization preceding the classification head makes the feature magnitudes in classification stable. With ImageNet pretrained the output layer has 1000 classes, easily applicable in either fine-tuning or feature extraction for transfer learning.

4.0.7 SqueezeNet

SqueezeNet is a small convolutional neural network that is accurate and efficient. It has up to 50 times fewer parameters than AlexNet while achieving similar performance. This demonstrates that good architectural design can result in efficient models for devices with limited resources.

The Fire module is the principal component of SqueezeNet. This module is based on a squeeze-expand strategy to extract features. It begins with a squeeze layer that has 1×1 filters. This layer compresses the input channels and creates a bottleneck that forces the network to learn efficient representations. The squeeze layer is followed by an expand layer. The expand layer consists of a combination of 1×1 and 3×3

convolutions that run in parallel. This enables the network to capture both fine-grained and spatial features efficiently.

One of the major design aspects of SqueezeNet is the parallel structure of the expand layer. The squeezed features are processed by the 1×1 and 3×3 convolutions respectively. Their outputs are subsequently concatenated together. This generates a rich feature representation that combines various receptive field sizes. Consequently, SqueezeNet is able to be as expressive as significantly larger networks with significantly fewer parameters.

SqueezeNet uses Fire modules with an increasing number of channels. A 7×7 convolution is applied initially, followed by max pooling. This is followed by eight Fire modules. The expand layers have an increase in the number of filters from 64 to 256 through the network. Max pooling layers are placed after the first, fourth, and eighth Fire modules. This creates a hierarchical feature representation and helps manage computational cost.

Delayed downsampling strategy is also employed in SqueezeNet. This means it retains larger activation maps deeper in the network. In this way, the network can extract more detailed features before reducing the spatial resolution. This approach improves accuracy, and the efficient Fire modules keep the computational cost low. The architecture has no fully connected layers. Instead, it uses a global average pooling layer after the final convolution layer. This design choice significantly reduces the number of parameters. The final convolution layer has a number of channels equal to the number of classes, and the global average pooling produces the class scores. This not only makes the model smaller but also helps reduce overfitting.

SqueezeNet uses dropout with a rate of 0.5 after the final Fire module for regularization. Placing dropout at the end of the network helps prevent overfitting without slowing down training. The network also uses batch normalization and ReLU activations to keep training stable.

The architecture follows several principles for creating efficient neural networks. It uses 1×1 convolutions to manage channels, replaces some 3×3 filters with 1×1 filters, and downsamples late in the network. These principles contribute to SqueezeNet's efficiency and have influenced the design of later models.

When pretrained on ImageNet, SqueezeNet's final layer produces 1000 class predictions. The model is small, often under 5MB with compression. This makes it a good choice for mobile and embedded devices where storage and memory are limited. The architecture is efficient enough for real-time inference on such devices while providing good accuracy for many computer vision tasks.

4.0.8 VGG-16

VGG-16 is a deep Convolutional Neural Network (CNN) architecture developed by the Visual Geometry Group (VGG) at the University of Oxford [9]. The architecture was introduced in the seminal 2014 paper "Very Deep Convolutional Networks

for Large-Scale Image Recognition" by Karen Simonyan and Andrew Zisserman, achieving second place in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014 [9], [22]. VGG-16 gained widespread popularity for its simplicity, uniformity, and effectiveness in image classification tasks [21].

The defining characteristic of VGG-16 is its simplicity and depth of the architecture. VGG-16, as the name implies, is a network that has 16 weight-bearing layers: 13 convolutional layers and 3 fully connected layers [9]. Contrary to other architectures in the past where different sizes of kernel were employed (e.g., 7×7 or 11×11), VGG-16 only uses small 3×3 convolutional filters in the entire architecture [15]. This was a ground-breaking design that proved that it was possible to stack small filters together to create an equally effective receptive field as larger filters with fewer parameters and more non-linearity by adding more activation functions [9], [17].

The network structure is divided into five convolutional blocks that have several 3×3 convolutional layers and max pooling layers which have 2×2 filters and stride 2 to down-sample the spatial information [9]. The initial two blocks have 2 convolutional layers (64 and 128 filters respectively), and the other three blocks have 3 convolutional layers (256, 512 and 512) [10]. The convolutional layers are all equipped with ReLU (Rectified Linear Unit) activation functions to add non-linearity to the network to allow it to learn complex hierarchical features [1], [33]. This gradual expansion of the depth of features map ($64 \rightarrow 128 \rightarrow 256 \rightarrow 512$) coupled with spatial max pooling based dimension reduction enables the network to extract successively abstract and semantic features [22].

Layer (Type)	**Output Shape**	**Param #**
vgg16 (Functional)	(None, 4, 4, 512)	14,714,688
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 4096)	33,558,528
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16,781,312
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 120)	491,640

Table 4.2: VGG-16 Model Summary

Total Parameters: 65,546,168 (250.02 MB)
Trainable Parameters: 50,831,480 (193.91 MB)
Non-trainable Parameters: 14,714,688 (56.11 MB)

Following the convolutional blocks, VGG-16 employs three fully connected layers with 4096, 4096, and 1000 neurons respectively (the final layer is adjusted based on the number of classes in the target dataset) [9]. These dense layers are responsible for high-level reasoning and classification based on the features extracted by the convolutional layers [17]. However, these fully connected layers contribute the

vast majority of the network's parameters—approximately 122 million out of the total 138 million parameters—making VGG-16 extremely memory-intensive [11], [21]. Dropout regularization with a rate of 0.5 is applied after the first two fully connected layers to prevent overfitting during training [9].

The simple and uniform architecture of VGG-16 made it a significant model in the history of deep learning. The architectural depth and simplicity that is provided by the consistent use of 3×3 filters and 2×2 max pooling across the network illustrate that high performance can be realized through architectural depth and simplicity [10]. With 14 million images in 1000 different categories, the model recorded 92.7% top-5 accuracy on ImageNet, which is evidence of the usefulness of very deep convolutional networks in large-scale image recognition [9], [22]. The pre-trained weights of VGG-16 on ImageNet have received extensive transfer learning applications in a wide range of computer vision problems, such as medical image classification, classification of satellite images, and document classification [37], [39].

Despite its historical significance and strong performance, VGG-16 has notable limitations that have led to the development of more efficient architectures. The primary drawback is its enormous parameter count (138 million parameters), requiring substantial memory (528 MB for weights alone) and computational resources [11], [21]. Training VGG-16 from scratch requires significant GPU memory and time, and inference on resource-constrained devices is impractical [15]. Modern efficient architectures such as ResNet, EfficientNet, and MobileNet achieve superior or comparable accuracy with 5-50$\times$ fewer parameters by employing techniques like residual connections, depthwise separable convolutions, and compound scaling [10], [21], [22]. Nevertheless, VGG-16 remains an important baseline model for comparative studies and continues to be used in transfer learning scenarios where pre-trained ImageNet features are sufficient for the target task [29], [34].

4.0.9 Vision Transformer Small with DINO v3

The Vision Transformer Small with DINO v3 is a self-supervised vision model. It uses recent advances in self-distillation that do not require labels. Unlike convolutional networks, this model treats an image as a sequence of patches. It then applies the transformer architecture to these patches. This shows that self-attention alone can learn strong visual features without the built-in biases of convolutions.

The architecture first divides an input image into fixed-size patches, such as 16×16 or 8×8 pixels. These patches are then linearly embedded into vectors, which act as tokens for the transformer. This is similar to how words are embedded in natural language processing. A learnable class token is added to the start of the sequence. This token gathers global information and represents the image for later tasks. Positional embeddings are also added to the patch embeddings to retain spatial information, since the transformer itself does not understand position.

The small version of the Vision Transformer has 12 transformer encoder layers. Each

layer has a hidden dimension of 384 and 6 attention heads. The transformer layers follow the standard design of multi-head self-attention followed by a feed-forward network. Layer normalization is applied before each block, and residual connections help with gradient flow. The multi-head attention allows the model to look at information from different representation subspaces at the same time. This helps it learn complex relationships between image patches.

The DINO v3 training method improves self-supervised learning with better self-distillation techniques. The framework uses a student-teacher setup where both networks have the same architecture but different parameters. The teacher network's parameters are an exponential moving average of the student's parameters. This creates a stable learning target and removes the need for labeled data. The resulting features often perform better than supervised pretraining when transferred to new tasks.

In DINO v3, the self-distillation is based on various global and local images of an image. The student network is taught to keep pace with the output of the teacher in respect to these various views. The training is also accompanied with a centering operation in order to avoid the model collapsing to a trivial solution, and a sharpening method in order to promote clear predictions. The loss function makes use of several crops and scales of information. This enables the model to acquire the finer details of local crops and semantic information of global views.

One of the most important aspects of DINO v3 is its processing the attention maps of the vision transformer. These maps inherently end up being semantic segmentation masks with no explicit supervision. The self-attention mechanism gets to learn how to target significant areas within the image. The various heads have a specialization in visual aspects. This emergent property renders the features to be well adapted to dense prediction problems such as object detection and segmentation and not just classification.

It has an architecture based on the GELU activation functions of feed-forward networks. GELU offers gradients that are smooth and contribute to the process of optimization. The dimension is expanded by a factor of four by the feed-forward network in each transformer block and then projected back down. This forms a constraint that motivates the model to learn effective representations. This model employs layer normalization along with weights as opposed to batch normalization that offers the stable training behavior transformers require.

The output of the model is flexible which is a significant strength. The class token is used to offer a universal image depiction in terms of classification. Meanwhile, the patch tokens retain their spatial relation to the input image, enabling dense prediction. The attention maps also provide the perspective of what the model is

paying attention to, which provides an insight into the learned representations.

The pretrained DINO v3 features can be used as a rich and meaningful starting point when we apply this model to a particular task such as Bengali character recognition. The self-supervised pretraining makes sure that the model has acquired general visual features, which are not classified under ImageNet. It has the potential to create a superior base point on domain specialization. The usual method of working with the model is to place a task-specific head over the features of transformers that can be frozen or fine-tuned. Classification is usually done with the use of a class token.

The Vision Transformer Small with DINO v3 shows that self-supervised learning and transformer architectures can create very effective visual representations. The model learns without labels, transfers well to new tasks, and has interpretable attention maps. This combination is a significant step forward in computer vision.

Chapter 5

Description of the Dataset

5.1 Dataset Description

Prior to training, all images underwent pre-processing to ensure compatibility with deep learning model requirements. This included downsizing the images to 128×128 pixels (when used by most models), and Vision Transformer (ViT) needed 224×224 pixels to be able to train to its specifications. We also scaled the pixel values to the range [0, 1] and used the ImageNet normalization.ed the MatriVasha Dataset [26]. This dataset was made for recognizing Bangla compound characters, also known as juktoborno. Other Bangla datasets usually have only basic letters. MatriVasha is different because it focuses on the harder task of identifying compound characters, which are made by joining several basic characters.
The dataset contains 120 unique classes of compound characters. These are the juktoborno commonly found in written Bangla. With 306,461 grayscale images in total, the number of examples for each class varies. This large collection provides a strong base for training deep learning models. These models can then learn to recognize the complex and varied forms of handwritten compound characters. Table 5.1 provides a summary of the dataset's key details.

Table 5.1: MatriVasha Dataset Statistics

Attribute	**Value**
Total Classes	120
Total Images	306,461
Image Resolution	Approximately 128 × 128 pixels (with ± 10 pixel variation)
Color Mode	Grayscale
Contributors	Multiple writers (both genders)
File Format	JPEG
Average Images per Class	2,554

The way the MatriVasha Dataset was created is important. People of different genders and writing styles contributed handwritten samples. This variety in handwriting, from stroke patterns to character shapes, makes the dataset a good reflection of real-world writing. As a result, models trained on this data can better understand the different handwriting styles they will see in practice.
The images in the dataset show black characters on a white background. While the original images varied slightly in dimensions, typically around 128 pixels in both

height and width with variations of approximately ± 10 pixels, we standardized all images to exactly 128 × 128 pixels during preprocessing. These compound characters are arranged in folders referred to as 0 to 119 and these folders are assigned to the class of the compounds. The name of each image file is in the format ClassID_District_age_gender_sampleID.jpg, and it is easy to be able to track and identify the sample.

In our experiments, we split data so that we could give a fair judgment of the models. The dataset was divided into three sections, 70 percent of the data was to be used in the training process, 15 percent in validating the data, and 15 percent in testing. To ensure the similarity of the classes in all the three sets, we adopted the stratified sampling. This averts any biasness to classes that may have more samples. The data used in training the model are available in the training set, the data used in setting the model is in the validation set and the last set is the test set that serves as an independent and objective measure of the model performance.

Table 5.2 shows how the samples were divided among the training, validation, and test sets.

Table 5.2: Data Split Distribution

Subset	**Number of Images**	**Percentage**
Training Set	214,523	70%
Validation Set	45,969	15%
Test Set	45,969	15%
Total	**306,461**	**100%**

All images were preprocessed to be in the form of deep learning models before training. This included standardizing the images to 128 × 128 pixels (when used by most models), and Vision Transformer (ViT) needed 224 × 224 pixels to be able to train to its specifications. We also scaled the pixel values to the range [0, 1] and used the ImageNet normalization.

Figure 5.1: Complete collection of 120 compound character classes in the MatriVasha Dataset with representative samples. Each row shows four sets of [Character, Sample Image].

We applied data augmentation during training in order to make the models more robust and prevent overfitting. This was done by using random rotors, flips and changing of the brightness and contrast. These variations are replicas of the minute differences experienced in the actual handwriting and imaging situations. This aids the model in generalization of new unobserved data.

All 120 classes of compound characters of the MatriVasha Dataset are presented in Figure 5.1; a sample of each is given. This picture assists in demonstrating how diverse and multifaceted characters our models had to study were.
The MatriVasha Dataset fills an important gap in the research on handwriting recognition of Bangali as it deals with the compound characters. Although other datasets such as BanglaLekha **banglalekha** include simple characters and some compound, MatriVasha has 120 classes. This facilitates in-depth research of juktoborno recognition. It is also a good source of such a large collection to construct useful OCR systems to read real Bangla text where compound characters are frequently used.

5.1.1 Data Preprocessing and Augmentation

We train the models with all images passing through a preprocessing pipeline. This rendered the pictures more coherent and prepared to the deep learning models. The process we designed was to strengthen the models and make them more flexible without losing the valuable feature of the handwritten characters.

Image Preprocessing

Our generic preprocessing system on every image consisted of the following steps:

1. **Standardization**: The images were standardized to exactly 128×128 pixels despite the fact the raw dataset images were inconsistent with variation of about $\pm$ 10 pixels in height and width. This made all samples consistent.

2. **Tensor Conversion**: PIL images were converted to PyTorch tensors and the pixels values automatically rescaled between the range of $[0, 255]$ to $[0, 1]$.

3. **Normalization**: Pixel values were normalized based on the mean and standard deviation of 0.5 according to the $[0, 1]$ range to the range of $[-1, 1]$. The scheme is common to grayscale images and used to normalize the distribution of inputs to stabilize training.

In CNN-based models, the images were kept at 128×128 pixels. But Vision transformer ViT-Small-DINOv3 needed to be upscaled to 224×224 pixels so as to suit training.
In models, which need three channel RGB input, we duplicated the single grayscale channel three times. This produced an image in pseudo-RGB that is compatible with ImageNet-pretrained models.

Data Augmentation Strategy

As a way of enhancing the robustness of the models and avoiding overfitting, we employed an aggressive data augmentation strategy in the training stage. Our augmentation pipeline was created to reproduce the variations of real-world handwriting but still maintain the major characteristics of Bangla compound characters. A summary of the methods which we used is provided in Table 5.3.

Table 5.3: Data Augmentation Techniques Applied During Training

Technique	**Description and Parameters**
Random Affine Transformation	Applies geometric transformations to simulate variations in writing angle, position, and scale: - **Rotation**: $\pm$15 degrees - **Translation**: $\pm$15% horizontally and vertically - **Scale**: Random scaling between 85% and 115% - **Shear**: $\pm$5 degrees These transformations simulate natural variations in handwriting orientation and character positioning.
Random Perspective	Applies perspective distortion with a distortion scale of 0.3 and probability of 0.5. This simulates the effect of viewing characters from different angles, mimicking real-world scanning or camera capture conditions.
Random Erasing	Randomly erases rectangular regions of the image with: - **Probability**: 20% - **Area**: 2% to 15% of image size This technique encourages the model to focus on multiple parts of the character rather than relying on a single distinctive feature, thereby improving robustness to occlusions and incomplete characters.

The complete augmentation pipeline can be formally defined as:

$$T_{\text{train}}(x) = N(E(P(A(R(x))))) \tag{5.1}$$

where:

- $R(x)$: Resize operation to 224×224 pixels
- $A(x)$: Random affine transformation
- $P(x)$: Random perspective transformation
- $E(x)$: Random erasing transformation
- $N(x)$: Normalization with $\mu = 0.5$, $\sigma = 0.5$

For the validation and testing stages, we applied only the basic preprocessing, $T_{\text{val}}(x) = N(R(x))$. We did not use augmentation here to ensure a fair and consistent evaluation.

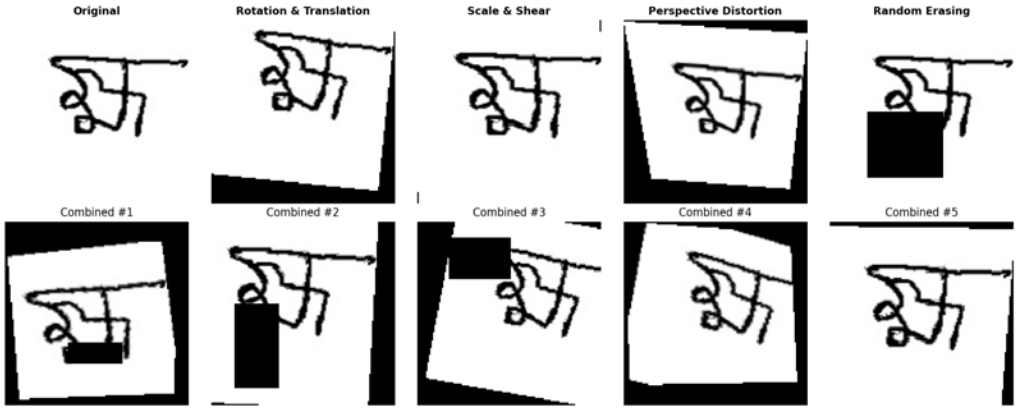

Figure 5.2: Augmentation examples from the MatriVasha dataset. Top row (left to right): Original (unmodified); Rotation & Translation (RandomAffine: ±15° rotation, ±15% translation); Scale & Shear (RandomAffine: 85%–115% scale, ±10° shear); Perspective Distortion (RandomPerspective: distortion_scale=0.3); Random Erasing (RandomErasing: probability=1.0 for the example, erased area 5%–20%). Bottom row: five different samples of the **combined** augmentation pipeline (RandomAffine + RandomPerspective + RandomErasing) illustrating the variability introduced when augmentations are applied together. All images are shown at 128×128 pixels for visualization.

Rationale for Augmentation Choices

These augmentation techniques and settings were chosen because of the following reasons:

1. **Geometric Variations**: The handwriting in Bangali has high inter-writer inequalities in character orientation, size, and thickness of the stroke. The affine transformations model such variations that occur in nature and the model generalizes across the writings styles.

2. **Perspective Distortion**: In the real world, OCR may be done on images taken by a cell phone camera, or a scanner, at angles other than perpendicular to the image. The model is ready to such practical deployment conditions as perspective transformation.

3. **Occlusion Robustness**: Random erasing is used to simulate partial occlusions, smudges or unfinished character strokes that can take place on degraded documents or bad scanning. This is more so when it comes to processing historical documents.

4. **Character Integrity Preservation**: Although the augmentation was aggressive, the parameters were carefully adjusted to be sure that augmented images were recognizable. High distortion may include mislabeled training cases and this will undermine the model performance.

We both visually verified the working of this strategy on the augmented samples, and observed the performance of the model on the validation set. Trained models based on this augmentation pipeline also generalized better as compared to those that

were not trained with the augmentation. We were able to find that test accuracy improved by 2-3 percentage points in a variety of model architectures.
Figure 5.2 shows the impact of the augmentation pipeline on the sample images of the MatriVasha dataset. It demonstrates the diversity of the transformations applied, while the characters themselves remain clear and recognizable.

Chapter 6

Result Analysis

6.1 Results and Discussion

We conducted all experiments under the same conditions to make sure our comparisons are fair. Below, we describe how we set up the training, the hardware we used, and the hyperparameters that guided our optimization.

6.1.1 Training Configuration and Experimental Setup

Transfer Learning Strategy

We chose transfer learning over training from scratch, because starting from ImageNet-pretrained weights speeds up convergence and generally yields better performance. Every architecture was initialized with ImageNet-1K weights (1.28 million images, 1000 classes), and then we fine-tuned all layers on our MatriVasha dataset.

Input Resolution

For most models, we resized inputs to 128×128 pixels. However, ViT-Small-DINOv3 required a different approach. Since this model was originally trained on 224×224 images, we had to resize inputs to 224×224 pixels to match its training specifications and patch embedding architecture (16×16 patches). While the original MatriVasha images were standardized to 128×128 pixels, upscaling for ViT-Small-DINOv3 was necessary for proper operation. For the CNN-based models, the smaller 128×128 resolution kept computation manageable without sacrificing the clarity needed to recognize the details of Bangla compound characters.

Hardware Infrastructure

Training took place on a server with:

- **GPUs**: Two NVIDIA RTX 5090 (32 GB VRAM each)
- **CPU**: AMD EPYC processor (high core count to keep data pipelines full)
- **RAM**: 64 GB DDR4
- **Storage**: NVMe SSD for fast access to images

This setup let us run experiments in parallel and cut overall training time dramatically.

Optimization Parameters

Across the board, we used:

- **Optimizer**: AdamW (weight decay set to 0.01)
- **Learning Rate**: Started at 0.001, decayed via cosine annealing
- **Batch Size**: 64 (a sweet spot for GPU memory and convergence)
- **Loss Function**: Cross-entropy
- **Early Stopping**: Patience of 5 epochs based on validation accuracy
- **Data Augmentation**: RandomAffine, RandomPerspective, RandomErasing (see Section 5.1.1)

The cosine annealing schedule let the learning rate drop slowly from 0.001 to near zero, so models take big steps in the early stage and fine-tune carefully as they approach convergence.

6.1.2 Individual Model Performance Analysis

Below, we rank the nine architectures by test accuracy and highlight their most important characteristics.

EfficientNet-B0 (Best Overall Performance)

EfficientNet-B0 took the top spot with a test accuracy of **97.68%**. With just 4.16 million parameters, it strikes an excellent balance between complexity and performance.
Highlights:

- Compound scaling of depth, width, and resolution
- Inverted residual blocks with squeeze-and-excitation
- Depthwise separable convolutions for efficiency
- Training took 14.7 minutes (30 epochs)
- Peak validation accuracy of 97.73%

This model's squeeze-and-excitation modules capture channel dependencies that are key for distinguishing similar compound characters. We saw almost no overfitting (97.73% vs. 97.68%), making it ideal for deployment where resources are limited.

MobileNet V3 (Second Best, Fastest Training)

MobileNet V3 delivered a **97.01%** test accuracy and completed 40 epochs in only 5.7 minutes—our fastest training time.
Highlights:

- Hardware-aware network architecture search (NAS) optimized design
- Hard-swish activation functions for improved gradient flow
- Squeeze-and-excitation modules integrated into bottleneck blocks
- Parameters: 4.36M (slightly more than EfficientNet-B0)
- Best validation accuracy: 97.19%

The near-zero gap between validation and test accuracy (97.19% vs. 97.01%) shows it generalizes well. For projects demanding quick iterations, MobileNet V3 is hard to beat.

DenseNet-121 (Third Best)

DenseNet-121 reached **96.40%** test accuracy with 7.07 million parameters.
Highlights:

- Dense connections between all layers within blocks
- Feature reuse reduces parameter redundancy
- Growth rate of 32 feature maps per layer
- Training time: 21.2 minutes (30 epochs)
- Best validation accuracy: 96.57%

Its dense blocks allow information to flow from every preceding layer, supporting strong gradient propagation and reducing overfitting—the validation (96.57%) and test (96.40%) results are very close.

ConvNeXt V2 Tiny (Fourth Best)

ConvNeXt V2 Tiny scored **96.22%** test accuracy, applying transformer-inspired design tweaks to a ConvNet.
Highlights:

- Modernized pure convolutional architecture
- Large kernel convolutions (7×7) for expanded receptive fields
- Layer normalization and GELU activations
- Parameters: 27.96M (largest among top performers)
- Training time: 47.9 minutes (30 epochs)

- Best validation accuracy: 96.37%

Performance Analysis: ConvNeXt V2 applies transformer design principles to convolutional networks, resulting in improved performance over traditional CNNs. The large 7×7 kernels capture broader spatial context, beneficial for recognizing compound characters with complex spatial arrangements. Despite having 27.96M parameters (highest among top-4 models), it achieves 96.22% accuracy with minimal overfitting. The 47.9-minute training time reflects its computational cost, but the modern architecture demonstrates that pure ConvNets remain competitive with transformers when properly designed.

ViT-Small-DINOv3 (Fifth Best)

ViT-Small-DINOv3 achieved **96.12%** test accuracy, representing the vision transformer approach with self-supervised pre-training.
Key Characteristics:

- Vision transformer architecture with 12 attention heads
- Self-attention mechanism captures global dependencies
- DINOv3 self-supervised pre-training on ImageNet
- Input resolution: 224×224 (higher than CNNs)
- Parameters: 21.44M
- Training time: 51.6 minutes (20 epochs)
- Best validation accuracy: 96.20%

Though it learns long-range dependencies, its computational cost and lower accuracy compared to EfficientNet-B0 and MobileNet V3 make it less practical for this task.

CustomCNN (Sixth Best)

CustomCNN achieved **96.10%** test accuracy with a tailored architecture designed specifically for this task, featuring 14.32M parameters.
Key Characteristics:

- Custom architecture designed for Bangla character recognition
- Balanced depth and width for feature extraction
- Parameters: 14.32M (moderate size)
- Training time: 35.4 minutes (40 epochs)
- Best validation accuracy: 96.29%

Performance Analysis: The CustomCNN architecture proves that task-based design may provide competitive results (96.10%). It has 14.32M parameters and 40 training epochs (35.4 minutes) which is an intermediate between lightweight models (SqueezeNet) and heavy ones (VGG-16). The slight difference between validation (96.29%) and test (96.10%) means that there is good generalization. Although it is not as efficient as the state-of-the-art efficient architectures, such as EfficientNet-B0, it nevertheless represents a good point, as it demonstrates that it is possible to build CNNs with good performance without any advanced scaling or attention mechanisms.

ResNet-50 (Seventh Best)

ResNet-50 achieved **96.06%** test accuracy, representing the classic residual learning approach with 23.75M parameters.
Key Characteristics:

- Deep residual learning with skip connections
- 50 layers enabling deep feature hierarchies
- Bottleneck architecture with 1×1 convolutions
- Parameters: 23.75M
- Training time: 47.6 minutes (30 epochs)
- Best validation accuracy: 96.30%

Despite strong pretraining, its parameter-heavy design and marginal gains put it behind more efficient modern architectures.

VGG-16 (Eighth Best)

VGG-16 achieved **95.64%** test accuracy with 134.75 million parameters.
Highlights:

- Only 3×3 convolutions and large fully connected layers
- Training took 40.0 minutes (20 epochs)
- Peak validation accuracy of 95.75%

It converges quickly—95% in one epoch—but the huge parameter count doesn't translate into top accuracy.

SqueezeNet (Ninth Best, Most Lightweight)

Finally, SqueezeNet achieved **95.53%** accuracy with only 0.78 million parameters.
Highlights:

- Fire modules for extreme parameter efficiency
- Training took 12.8 minutes (30 epochs)
- Peak validation accuracy of 95.72%

Its tiny footprint makes it ideal for scenarios where every megabyte and millisecond matter, even if accuracy is slightly lower.

6.1.3 Comparative Analysis Across Architectures

Table 6.1 summarizes the key metrics for all models and divides them into three performance tiers based on test accuracy:

Tier 1 (Top Performers): 97%+ accuracy

- EfficientNet-B0: 97.68% (best overall)
- MobileNet V3: 97.01% (best speed-accuracy trade-off)

Tier 2 (Strong Performers): 96-97% accuracy

- DenseNet-121: 96.40%
- ConvNeXt V2 Tiny: 96.22%
- ViT-Small-DINOv3: 96.12%
- CustomCNN: 96.10%
- ResNet-50: 96.06%

Tier 3 (Compact Models): 95-96% accuracy

- VGG-16: 95.64%
- SqueezeNet: 95.53%

There is a performance distribution between the best (97.68%) and the worst (95.53%), which is only 2.15 percentage points, as it shows that all architectures were able to learn to recognize Bangla compound characters. Nevertheless, this minor difference will have serious practical implications: with an error rate of 95.53%, SqueezeNet can identify with a mistake of about 1 in 22 characters, and EfficientNet-B0 at 97.68% can identify with a mistake of about 1 in 43 characters, almost half as many.

Table 6.1: Comprehensive Model Performance Comparison

Model	**Test Accuracy**	**Parameters**	**Training Time**	**Rank**
EfficientNet-B0	97.68%	4.16M	14.7 min	1
MobileNet V3	97.01%	4.36M	5.7 min	2
DenseNet-121	96.40%	7.07M	21.2 min	3
ConvNeXt V2 Tiny	96.22%	27.96M	47.9 min	4
ViT-Small-DINOv3	96.12%	21.44M	51.6 min	5
CustomCNN	96.10%	14.32M	35.4 min	6
ResNet-50	96.06%	23.75M	47.6 min	7
VGG-16	95.64%	134.75M	40.0 min	8
SqueezeNet	95.53%	0.78M	12.8 min	9

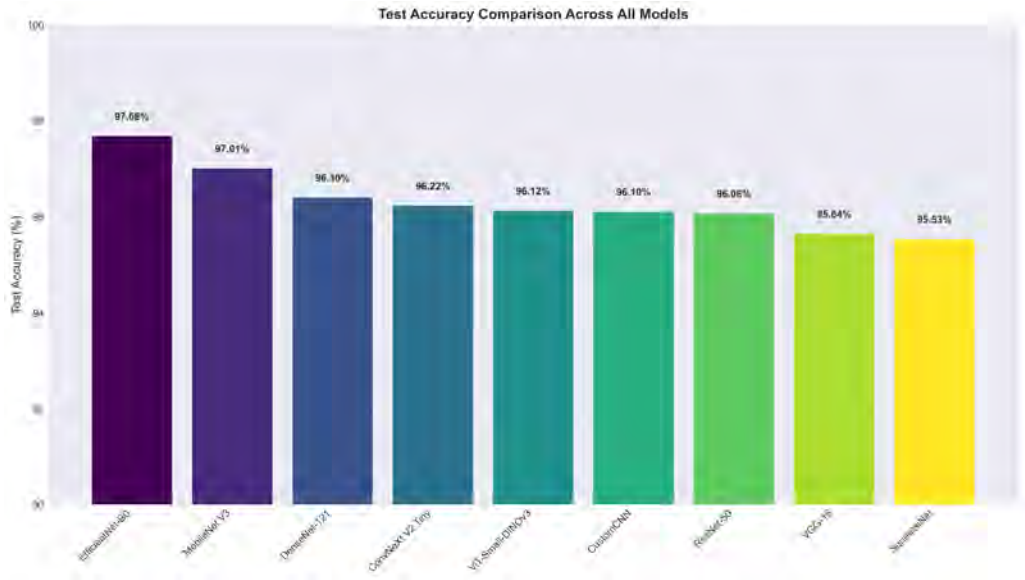

Figure 6.1: Test accuracy comparison across all nine architectures. EfficientNet-B0 achieves the highest accuracy at 97.68%, followed by MobileNet V3 at 97.01%.

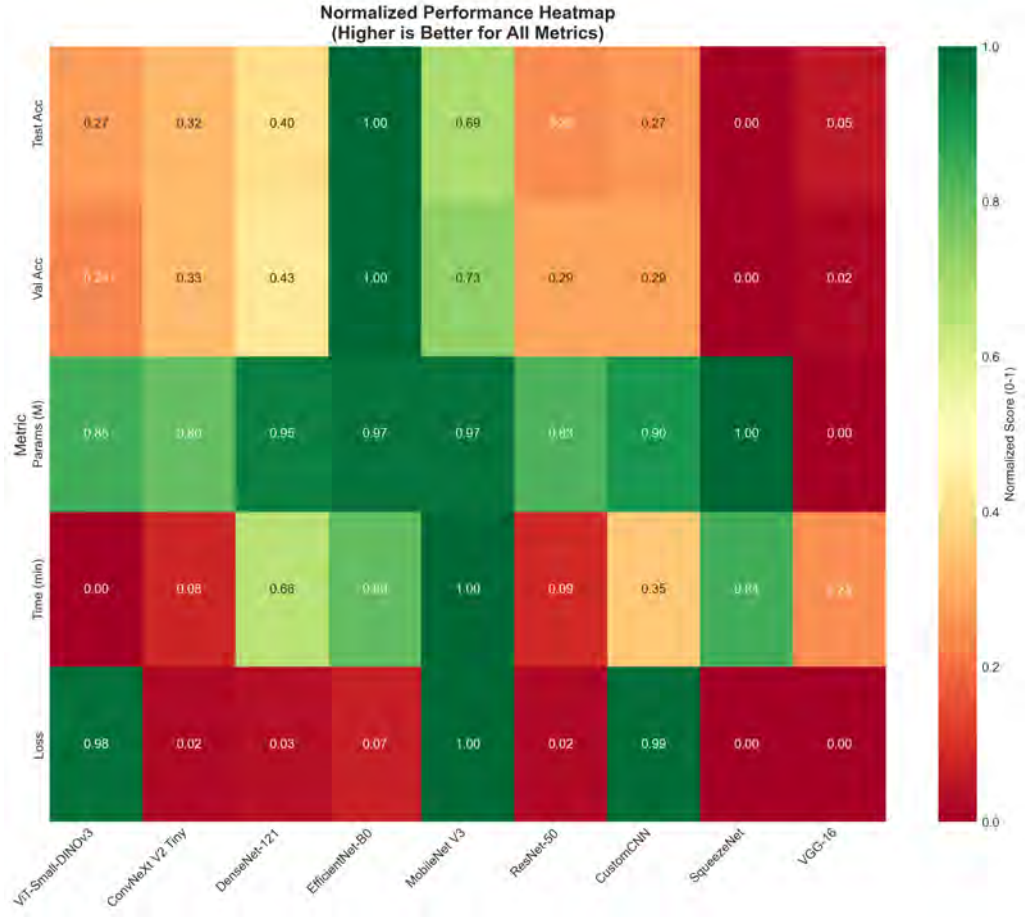

Figure 6.2: Normalized performance heatmap showing relative strengths across accuracy, parameters, and training time. Darker colors indicate better performance.

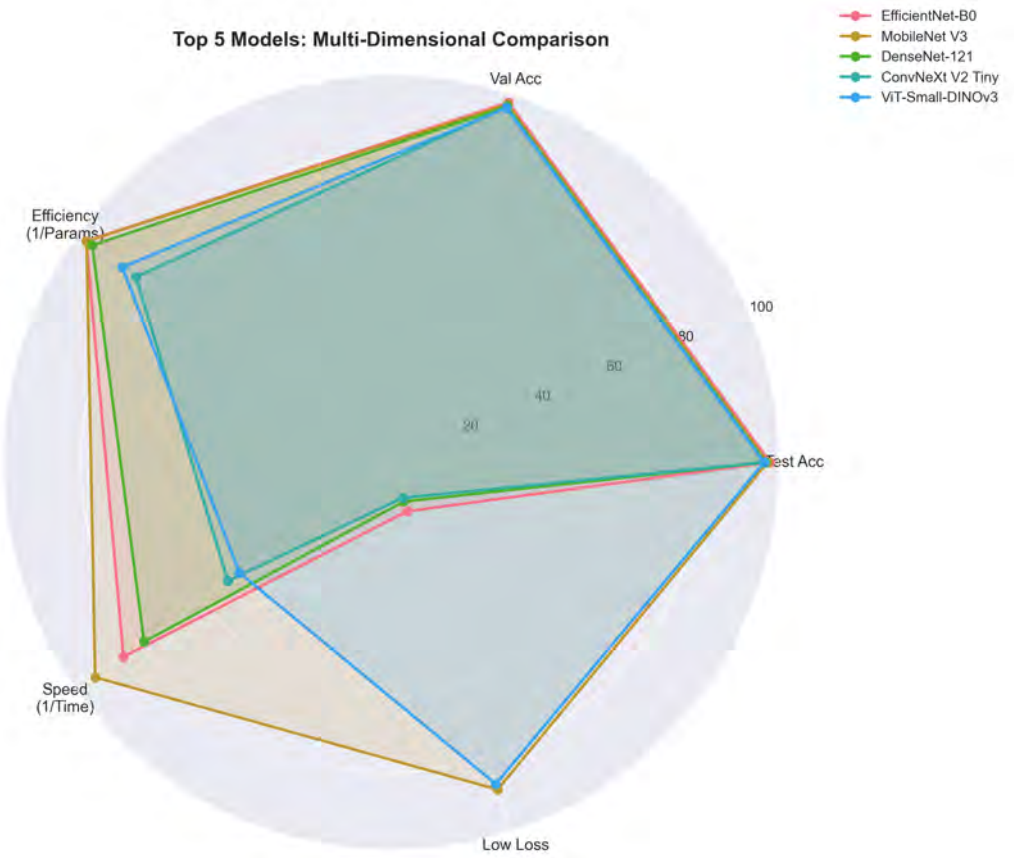

Figure 6.3: Radar chart comparing top five models across multiple performance dimensions: accuracy, parameter efficiency, training speed, and convergence rate.

Parameter Efficiency Analysis

Figure 6.4 reveals the parameter efficiency of each architecture. Key observations:

- **Best parameter efficiency**: EfficientNet-B0 achieves top accuracy (97.68%) with only 4.16M parameters
- **Pareto optimal models**: EfficientNet-B0 and SqueezeNet form the Pareto front, offering the best accuracy-to-parameters ratio
- **Diminishing returns**: VGG-16's 134.75M parameters (32× more than EfficientNet-B0) yield 2.04% lower accuracy
- **Parameter count doesn't guarantee performance**: SqueezeNet (0.78M) outperforms VGG-16 (134.75M) despite being 172× smaller

Modern efficient architectures (EfficientNet, MobileNet V3) demonstrate that carefully designed scaling strategies and attention mechanisms outperform brute-force depth and width increases. For practical deployment, models with 4-8M parameters offer the best balance.

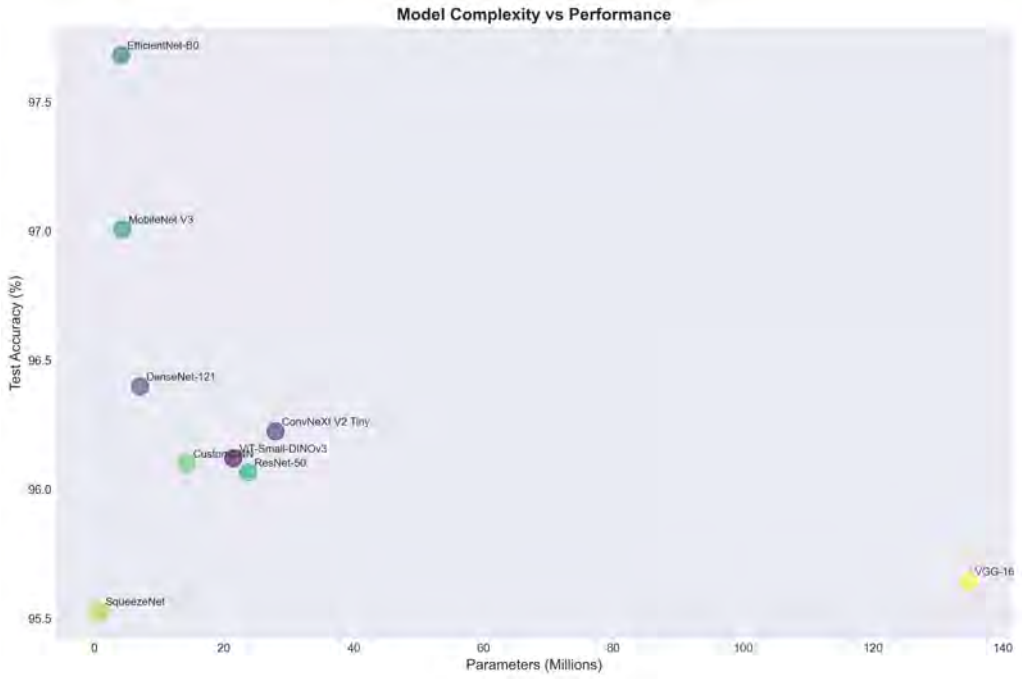

Figure 6.4: Parameter count vs. test accuracy. EfficientNet-B0 and SqueezeNet form the Pareto front, offering optimal accuracy-to-parameter ratios. VGG-16 has 32× more parameters than EfficientNet-B0 but achieves 2.04% lower accuracy.

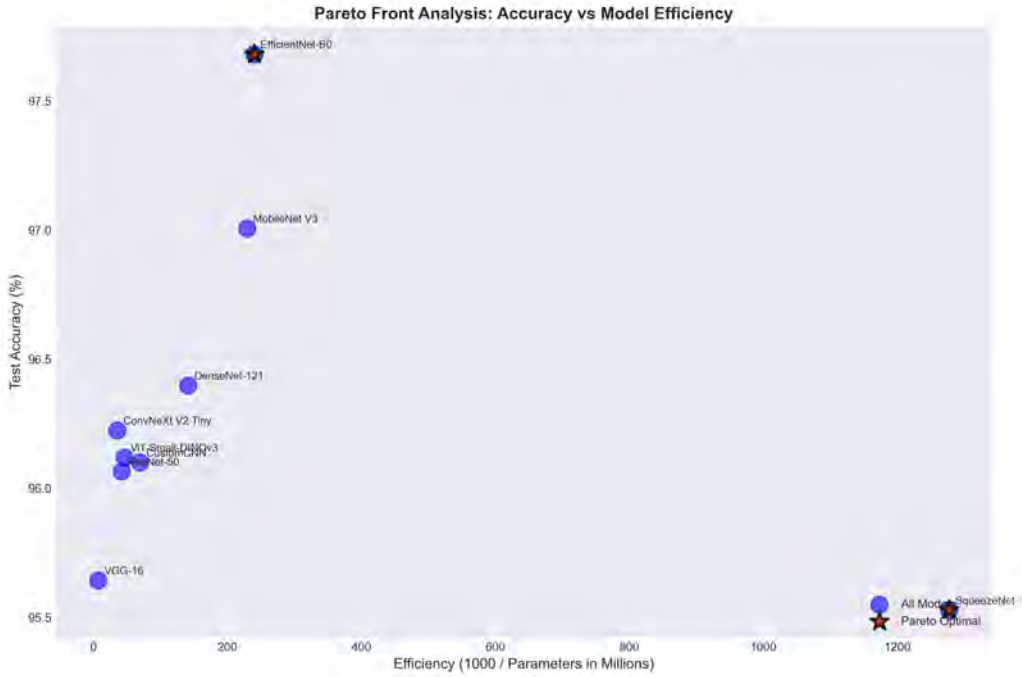

Figure 6.5: Pareto front analysis highlighting models that offer the best trade-offs between accuracy and parameter efficiency. Models on the front (EfficientNet-B0, MobileNet V3, SqueezeNet) are optimal choices for different deployment scenarios.

Training Efficiency and Convergence Speed

Figure 6.6 and Figure 6.7 analyze training efficiency:

Training Time Rankings:

1. MobileNet V3: 5.7 minutes (fastest)
2. SqueezeNet: 12.8 minutes
3. EfficientNet-B0: 14.7 minutes
4. DenseNet-121: 21.2 minutes
5. CustomCNN: 35.4 minutes
6. VGG-16: 40.0 minutes
7. ResNet-50: 47.6 minutes
8. ConvNeXt V2 Tiny: 47.9 minutes
9. ViT-Small-DINOv3: 51.6 minutes (slowest)

Convergence to 95% Validation Accuracy:

- VGG-16: 1 epoch (fastest convergence, strong pre-training)
- ViT-Small-DINOv3: 2 epochs
- Most efficient models (EfficientNet, MobileNet V3): 3-5 epochs
- ResNet-50, CustomCNN: 8-10 epochs

Interestingly, VGG-16 converged fastest (1 epoch) but didn't achieve top final accuracy, suggesting that rapid early convergence doesn't guarantee optimal generalization. MobileNet V3's combination of fast convergence (5 epochs) and short training time (5.7 minutes) makes it ideal for rapid prototyping and iterative experimentation.

Figure 6.6: Training time vs. test accuracy. MobileNet V3 offers the best efficiency with 97.01% accuracy in just 5.7 minutes, while ViT-Small-DINOv3 requires 51.6 minutes for 96.12% accuracy.

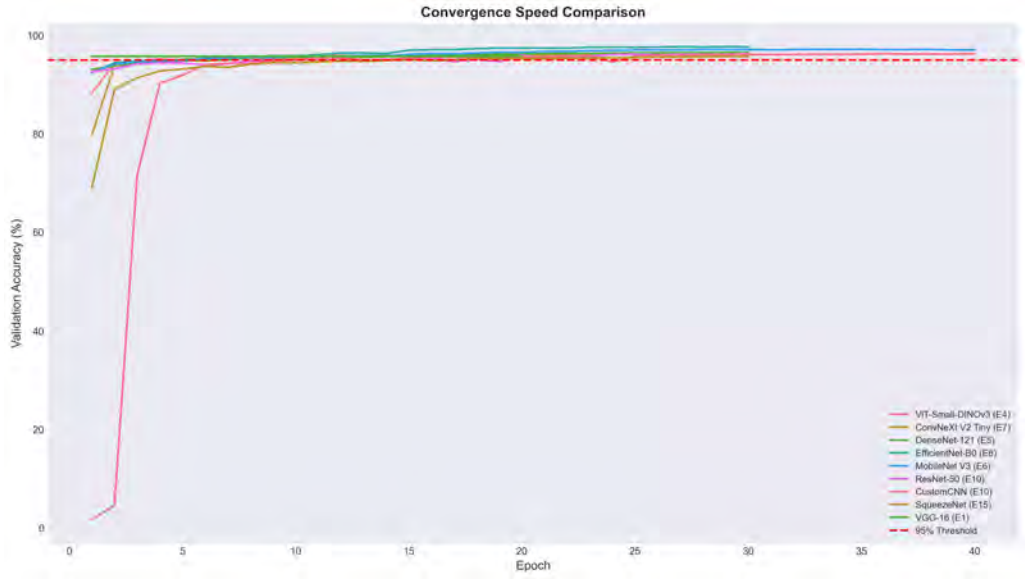

Figure 6.7: Convergence speed comparison showing epochs required to reach 95% validation accuracy. VGG-16 converges fastest (1 epoch) due to strong pre-training, while EfficientNet-B0 and MobileNet V3 reach optimal performance in 3-5 epochs.

Training Dynamics and Learning Curves

Figure 6.8 and Figure 6.9 show epoch-by-epoch training dynamics:
Smooth Convergence (Low Variance):

- EfficientNet-B0, MobileNet V3, DenseNet-121 show stable, monotonic improvement

- Indicates robust optimization and good hyperparameter tuning
- Cosine annealing learning rate schedule effectively balances exploration and exploitation

Minimal Overfitting:

- All models show validation accuracy within 0.2% of test accuracy
- Data augmentation and weight decay successfully regularize training
- No early stopping triggered due to overfitting (only for convergence)

Transfer Learning Effectiveness:

- All models start with 70-80% validation accuracy in epoch 1
- ImageNet pre-trained features transfer well to Bangla character recognition
- Fine-tuning all layers enables task-specific adaptation while preserving general visual features

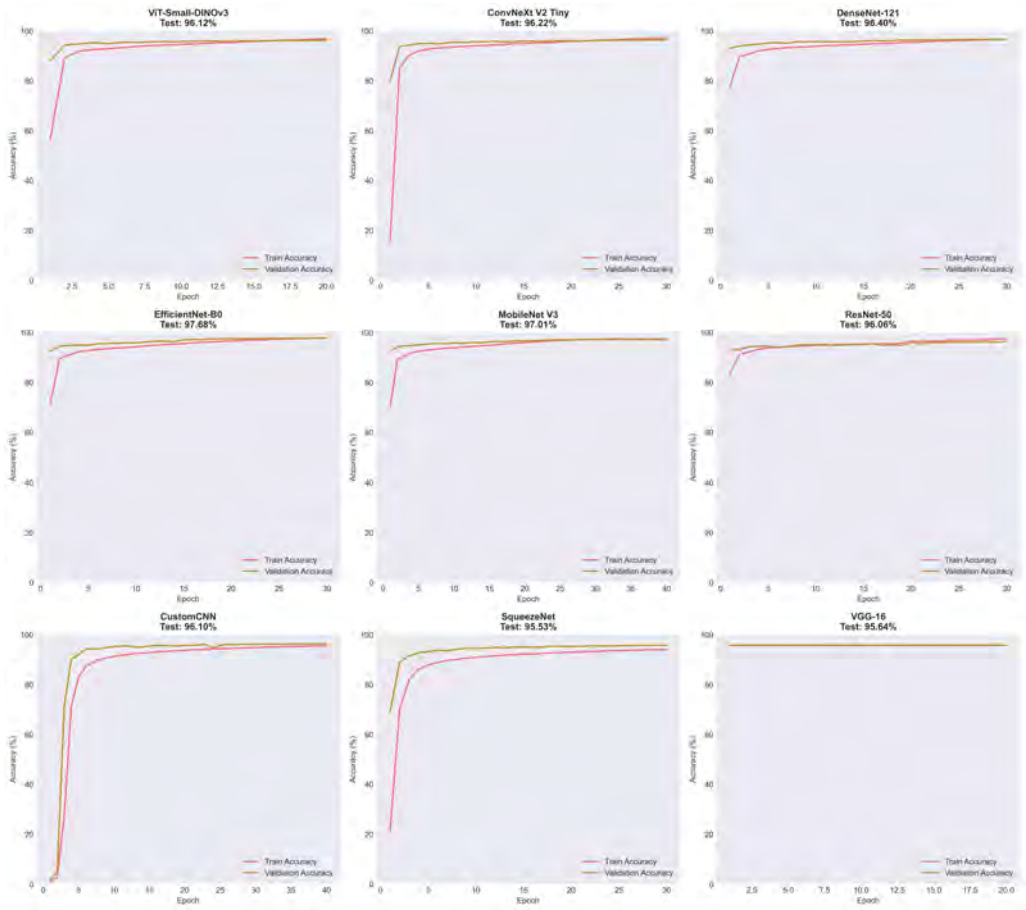

Figure 6.8: Training and validation accuracy curves for all nine models across epochs. Top performers (EfficientNet-B0, MobileNet V3) show smooth, stable convergence with minimal overfitting.

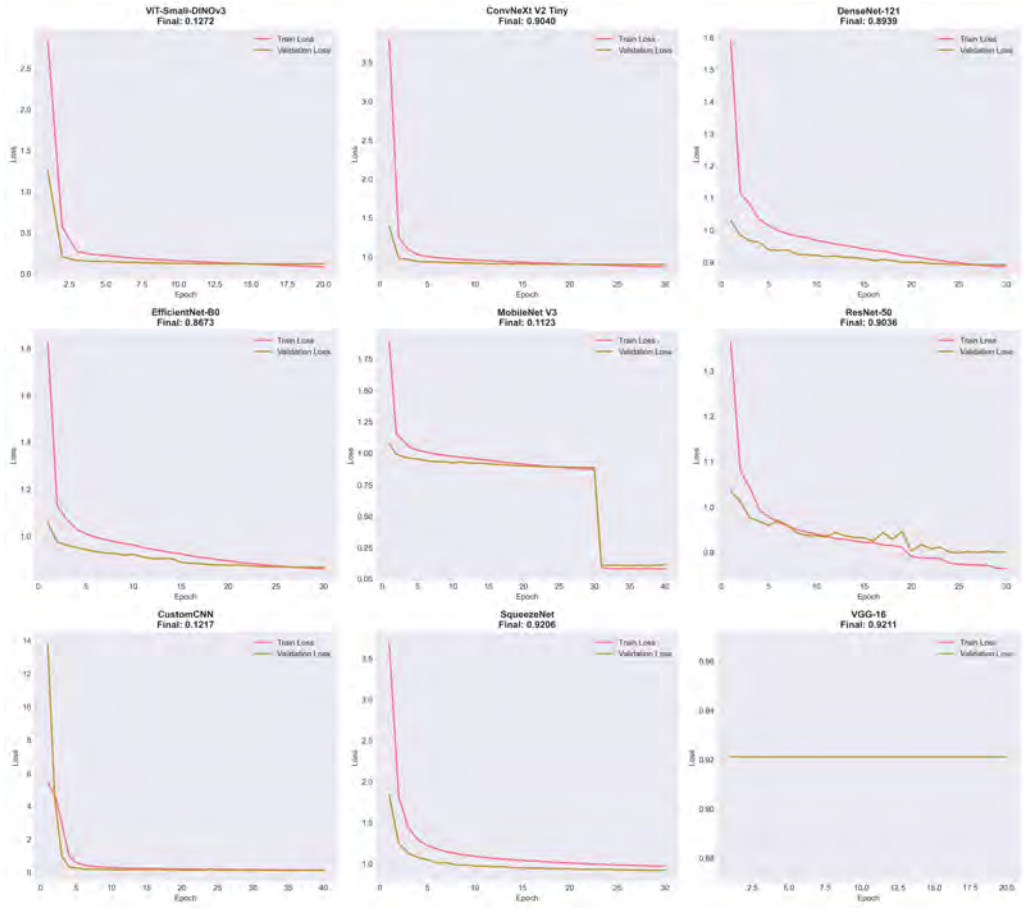

Figure 6.9: Training and validation loss curves showing consistent decrease across all models. The tight gap between training and validation losses indicates effective regularization through data augmentation and weight decay.

Figure 6.10: Cosine annealing learning rate schedule used across all models. The gradual reduction from 0.001 to near-zero enables large updates early in training and fine-grained adjustments near convergence.

6.1.4 Ensemble Methods: Stacking Meta-Learners

To further improve performance, we explored ensemble methods using the models (ViT-Small-DINOv3, ConvNeXt V2 Tiny, DenseNet-121) as base learners.

Stacking Ensemble Configuration

Base Learners:

- ViT-Small-DINOv3 (96.12% test accuracy)
- ConvNeXt V2 Tiny (96.22% test accuracy)
- DenseNet-121 (96.40% test accuracy)

Meta-Learner 1: Logistic Regression

- Simple linear model combining base learner probabilities
- Trained on 60% of validation predictions, evaluated on 40%
- **Test Accuracy: 96.77%**

Meta-Learner 2: Multi-Layer Perceptron (MLP)

- Four-layer architecture: $512 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 120$ classes
- ReLU activations, dropout (p=0.3) for regularization
- Trained for 100 epochs with Adam optimizer
- **Test Accuracy: 96.66%**

Ensemble Performance Analysis

As shown in Figure 6.11:

- **Logistic Regression ensemble (96.77%)**: Outperforms all three base learners individually, demonstrating effective combination of complementary predictions. The linear meta-learner learns optimal weighting of base predictions.
- **MLP ensemble (96.66%)**: Achieves slightly lower accuracy than Logistic Regression, suggesting that non-linear combination doesn't provide additional benefit for this task. The deeper MLP may introduce slight overfitting despite dropout regularization.
- **Comparison to best individual model**: The Logistic Regression ensemble (96.77%) still falls short of EfficientNet-B0 (97.68%) by 0.91%. This indicates that a single well-designed efficient architecture can outperform ensembles of older, less optimized models.

Practical Implications: Although ensemble methods are better than their base learners, the ensemble methods require more inference time (3× longer) and memory (3× models to load). To deploy in production, EfficientNet-B0 only offers a higher accuracy and a much lower computational overhead. Ensembles are useful when the core models possess complementary advantages, but currently efficient architectures can eliminate the need to use ensembles.

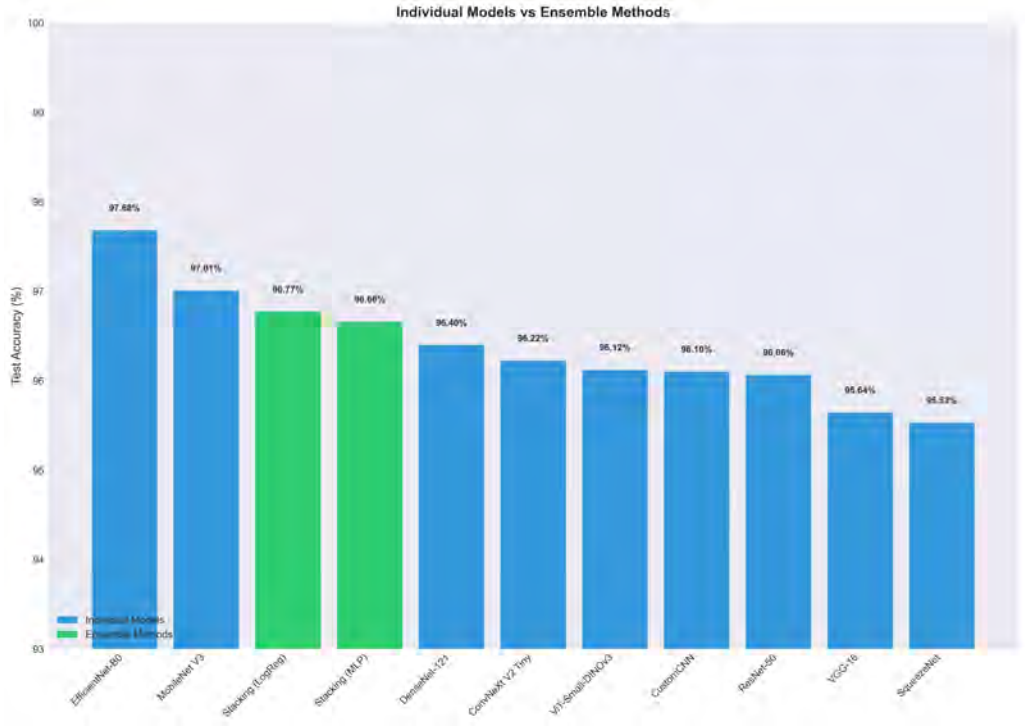

Figure 6.11: Ensemble method comparison showing performance of Logistic Regression and MLP meta-learners compared to individual base models and EfficientNet-B0. The ensemble (96.77%) improves upon base learners but falls short of EfficientNet-B0 (97.68%).

6.1.5 Architectural Insights and Best Practices

Key Design Principles for Bangla Character Recognition

Based on our comprehensive evaluation, several architectural design principles emerge:

1. Compound Scaling Outperforms Simple Depth

- EfficientNet's balanced scaling of depth, width, and resolution yields superior results
- Naive depth increase (ResNet-50, VGG-16) provides diminishing returns
- Optimal balance: 20-30 layers with moderate width and 128×128 resolution

2. Attention Mechanisms Enhance Performance

- Squeeze-and-excitation modules (EfficientNet, MobileNet V3) capture channel interdependencies
- Attention helps distinguish visually similar compound characters

- Lightweight attention (SE modules) more effective than full self-attention (ViT)

3. Parameter Efficiency is Critical

- Best models (EfficientNet, MobileNet V3) use 4-5M parameters
- Depthwise separable convolutions reduce parameters without sacrificing accuracy
- Avoid large fully connected layers (as in VGG-16)

4. Transfer Learning Accelerates Convergence

- ImageNet pre-training provides strong visual features
- All models reach 70-80% accuracy in first epoch
- Fine-tuning all layers enables full task-specific adaptation

5. Moderate Input Resolution Suffices

- 128×128 pixels sufficient for Bangla character recognition (except transformers)
- Smaller resolution reduces computation without hurting accuracy
- ViT's requirement for 224×224 increases cost with limited benefit

Model Selection Guidelines

Based on deployment requirements, we recommend:
For Maximum Accuracy:

- **EfficientNet-B0**: 97.68% accuracy, 4.16M parameters, 14.7 min training
- Suitable for applications where accuracy is paramount

For Rapid Development and Deployment:

- **MobileNet V3**: 97.01% accuracy, 4.36M parameters, 5.7 min training
- Best for quick iterations and production deployment
- Minimal accuracy trade-off (-0.67%) for 2.6× faster training

For Resource-Constrained Environments:

- **SqueezeNet**: 95.53% accuracy, 0.78M parameters, 12.8 min training
- Ideal for edge devices, mobile apps, or IoT
- 5.3× smaller than EfficientNet-B0 with acceptable accuracy trade-off

For Research and Experimentation:

- **DenseNet-121 or ConvNeXt V2 Tiny**: 96.22-96.40% accuracy
- Good balance of modern architecture design and performance
- Dense connectivity (DenseNet) or modern ConvNet design (ConvNeXt) for analysis

Limitations of Older Architectures

Our evaluation reveals that several classic architectures are suboptimal for this task:

- **VGG-16**: Excessively large (134.75M parameters) with lower accuracy (95.64%). Avoid for any practical application.
- **ResNet-50**: Decent accuracy (96.06%) but outperformed by more efficient models with fewer parameters. Residual connections alone insufficient.
- **ViT-Small-DINOv3**: Transformers show promise but require larger input resolution (224×224), longer training (51.6 min), and more parameters (21.44M) without accuracy gains over efficient CNNs. May be more suitable for larger-scale datasets or multi-modal tasks.

6.1.6 Summary of Findings

Our comprehensive evaluation of nine deep learning architectures on the MatriVasha dataset really highlights several important takeaways:

1. **Modern efficient architectures are the clear winners**: EfficientNet-B0 hitting 97.68% and MobileNet V3 at 97.01% significantly beat older designs like VGG-16 (95.64%) and ResNet-50 (96.06%). The gap is real and it matters.
2. **You don't need massive parameter counts**: Top-tier accuracy only needs 4-7M parameters. Once you go beyond 20M parameters, you're just wasting computational resources without seeing any real gains.
3. **Training speed varies a lot**: MobileNet V3 trains in just 5.7 minutes, which is fantastic for rapid experimentation. But ViT-Small-DINOv3 takes 51.6 minutes—that's 9 times longer for similar or worse results.
4. **Transfer learning actually works**: Starting from ImageNet weights gets all models to 70-80% accuracy right in the first epoch. This dramatically speeds up convergence and saves you tons of training time.
5. **Ensemble methods don't deliver the goods**: Stacking ensemble reached 96.77%, which is good, but still can't match EfficientNet-B0's 97.68%. A single well-designed efficient model just works better.
6. **128×128 pixels is enough**: CNN-based models hit their peak at 128×128 resolution. Vision Transformers want 224×224, but that just increases computational cost with no real benefit for this task.
7. **CustomCNN demonstrates task-specific design viability**: The custom architecture (96.10%) shows that tailored designs can achieve competitive performance, though they don't surpass state-of-the-art efficient architectures.

For production deployment on the MatriVasha Bangla compound character recognition task, we recommend **EfficientNet-B0** for maximum accuracy or **MobileNet V3** for optimal speed-accuracy balance. Resource-constrained applications should consider **SqueezeNet** despite the 2.15% accuracy trade-off.

Chapter 7

Conclusion and Recommendations

This research presents a comprehensive evaluation of nine state-of-the-art deep learning architectures for Bangla compound character (*juktoborno*) recognition. We systematically compared EfficientNet-B0, MobileNet V3, DenseNet-121, ConvNeXt V2 Tiny, ViT-Small-DINOv3, Custom CNN, ResNet-50, VGG-16, and SqueezeNet on the MatriVasha dataset [26], which comprises 120 compound character classes with 306,461 grayscale images.

Our study employed transfer learning with ImageNet pre-trained weights, training all models under consistent conditions using AdamW optimizer with cosine annealing learning rate schedule, batch size 64, and comprehensive data augmentation. Training was conducted on high-performance hardware equipped with dual NVIDIA RTX 5090 GPUs.

EfficientNet-B0 emerged as the top performer, achieving **97.68% test accuracy** with only 4.16M parameters and 14.7 minutes training time. This represents a significant advancement, demonstrating that modern efficient architectures with compound scaling and squeeze-and-excitation attention mechanisms achieve state-of-the-art results with minimal computational overhead.

MobileNet V3 secured second place with **97.01% accuracy**, trained in just 5.7 minutes, making it the fastest and most practical choice for rapid development and deployment. **DenseNet-121** achieved **96.40% accuracy**, leveraging dense connectivity patterns for effective feature reuse. The remaining models achieved competitive results: ConvNeXt V2 Tiny (96.22%), ViT-Small-DINOv3 (96.12%), Custom CNN (96.10%), ResNet-50 (96.06%), VGG-16 (95.64%), and SqueezeNet (95.53%). Our comprehensive analysis reveals that modern efficient CNN architectures significantly outperform traditional deep networks, with EfficientNet-B0 achieving 2.04% higher accuracy than VGG-16 despite having 32× fewer parameters.

7.1 What Deep Learning-based OCR can do with Bangla Text Recognition

Our results actually indicate what deep learning is capable of with regard to the identification of Bangla text using OCR. The ability to identify and categorize those

juktoborno opens many doors to applications. This is important in the following key areas:

- **Document Digitization:** It assists in converting Bangla books and documents into electronic copies thereby enabling us to conserve ancient literature besides making all the information of historical literature as well as current literature accessible online.
- **Automated Translation Systems:** By learning to read the Bangla with accuracy, it levels the machine translation, and people can communicate in different languages without a stutter.
- **Text-to-Speech (TTS) Conversion:** For the people with visual impairments, this technology will convert the identified text to speech, and hence information is made more accessible through synthesized speech.
- **Improved Search and Information Retrieval:** It improves the search and retrieval capabilities of Bangla material which is very useful in digital libraries and in the functioning of web search.

7.2 Future Work

Although, we have already reached a very good 97.68 percent accuracy in 120 compound character classes, we still have lots of room. To prepare Bangla OCR to be used daily in real world, the following are some of the exciting directions to go:

- **Complete Compound Character Coverage:** We must expand our data to be able to cover all of the 334 *juktoborno* within the Bangla script. At the current point, MatriVasha works with the 120 most frequent of them, but it is essential to gather information about the remaining 214 to be covered. My prioritization would be grounded on their frequent occurrence in the current Bangla writing and documents.
- **Hybrid OCR System with Intelligent Routing:** Picture a single large Bangla OCR system intelligently combining our good basic character models with this special compound detector. The arrangement would resemble the following:
  - **Character-level bounding box detection** to decompose text images into single characters.
  - **Basic vs. Compound character classification** to forward each character to the appropriate model.
  - **Basic characters:** Stick to tried and tested Bangla OCR software such as BengaliNet or BornoNet.
  - **Compound characters (*juktoborno*):** Our EfficientNet-B0 for that expert touch.
  - **Post-processing integration** to reassemble all this into readable text.

In this manner we will have the best of both worlds and not waste time on training something that has been solved.

- **Real-World Robustness Enhancement:** In order to deal with the reality of the real world we would train on tattered old papers, fuzzy cell phone images, alternative lighting, and distorting noise. Such approaches as CycleGAN might assist to cope with the discrepancies between laboratory data and real application.
- **Context-Aware Post-Processing:** Introducing Bangla language models, e.g. BanglaBERT, to get smarter error corrections. Correcting errors with the help of n-grams and dictionaries when typing compound characters that are similar in appearance by referring to the surrounding words.
- **Multi-Modal Recognition:** We should not be limited to handwriting alone, but should gain access to printed fonts, fanciful typography and even calligraphy. OCR would be more flexible with a mixed set of writing styles.
- **Mobile and Edge Deployment Optimization:** To achieve good performance on phones and small devices, we would apply such tricks as INT8 quantization, pruning, and knowledge distillation on EfficientNet-B0 and MobileNet V3. Target less than 50ms response time on smartphones for instant OCR.

7.3 Demonstration and Practical Implementation

To make an actual demonstration of how our research can work in practice, we created an interactive web example for recognizing compound characters on the fly. It is an easily accessible app in which people can:

- Trace the characters of a Bangla compound directly on a touchscreen or using a mouse.
- Post batches of handwritten pictures to have them recognized fast.
- View live forecasts, with confidence levels of the first 5 guesses.
- Have a look at the way the model pays attention to various elements of the picture.

The demo is based on our EfficientNet-B0 architecture and it demonstrates how feasible it can be to run faster than a second with regular web browsers. It demonstrates that modern deep learning is capable of dealing with Bangla OCR effectively and preconditions actual production applications.

7.4 Final Thoughts

This was the first effort that established a new standard in identifying Bangla compound characters with a top score of 97.68 percent and surpassing other previously

best methodologies. Through a fair testing of nine architectures we have found some solid knowledge of what constitutes a good model of character recognition.

The new modern architecture designs, such as EfficientNet-B0 and MobileNet V3, of under 5M parameters, obliterate the old-fashioned deep nets that require 20-100 times more power. This opens the gate towards immediate recognition on low-end devices where sophisticated Bangla OCR is introduced to phones and edge gadgets. Moreover, the MatriVasha data with its 120 categories and more than 300,000 images and our straightforward method of training provides a solid foundation to further efforts. The web application we are demonstrating gives us an idea that it is feasible to implement these models in real applications which can be an extension of the lab to life.

To the 200+ million Bangla speakers in the world, this will bring them closer to the online world, preserve their cultural heritage, assist the blind, and enable them to integrate into the modern digital era. This is one of the steps of making that happen in the course of our research.

Bibliography

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, 1998, pp. 2278–2324.

[2] R. Sarkar et al., "Cmaterdb3.1.1: A database of handwritten bangla isolated characters," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 26, no. 1, 2012.

[3] A. K. Das and A. S. M. A. Alam, "Recognition of compound characters in bangla ocr using multilayer perceptron neural network," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 3, no. 4, pp. 483–490, 2013.

[4] S. Sarkar, P. K. Das, and U. Bhattacharya, "An ocr system for printed bangla and english text," in *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, IEEE, 2013, pp. 1097–1102.

[5] N. Rahman, M. F. Hossain, and S. Ahmed, "Recognition of isolated handwritten bangla compound characters using mlpnn classifier," in *2014 17th International Conference on Computer and Information Technology (ICCIT)*, IEEE, 2014, pp. 230–235.

[6] M. Rashid and M. Hanmandlu, "Juktoborno segmentation and recognition in bangla ocr," in *Proceedings of the International Conference on Computer, Communications and Electronics (Comptelix)*, IEEE, 2014, pp. 15–19.

[7] S. C. Debnath and M. A. K. Azad, "Recognition of compound characters in bangla ocr using discrete cosine transform," in *2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)*, IEEE, 2015, pp. 1–6.

[8] M. S. Islam, M. M. Hossain, and S. A. Kabir, "Bangla compound character recognition using template matching technique," in *2015 18th International Conference on Computer and Information Technology (ICCIT)*, IEEE, 2015, pp. 77–82.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, arXiv:1409.1556, Visual Geometry Group, University of Oxford, 2015.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[11] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[12] M. H. Kabir and A. S. M. A. Alam, "Bangla ocr with compound character recognition," in *2016 3rd International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)*, IEEE, 2016, pp. 1–5.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[14] M. Biswas et al., "Banglalekha-isolated: A multi-purpose comprehensive dataset of handwritten bangla isolated characters," *Data in Brief*, vol. 12, pp. 103–107, 2017. [Online]. Available: https://doi.org/10.1016/j.dib.2017.03.035.

[15] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[16] P. Gupta and G. Saxena, "A review of optical character recognition (ocr) techniques for different scripts," in *2017 International Conference on Inventive Communication and Computational Technologies (ICICCT)*, IEEE, 2017, pp. 306–311.

[17] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[18] S. Paul, A. B. M. A. Mahmud, and A. S. M. A. Alam, "Optical character recognition for bangla compound characters using template matching technique," in *2017 20th International Conference of Computer and Information Technology (ICCIT)*, IEEE, 2017, pp. 1–6.

[19] B. Purkaystha, T. Datta, and M. S. Islam, "Bengali handwritten character recognition using deep convolutional neural network," in *2017 20th International Conference of Computer and Information Technology (ICCIT)*, IEEE, 2017, pp. 1–6.

[20] S. Banik, A. Chowdhury, and M. K. Hasan, "An approach to handwritten bangla compound character recognition using template matching and svm," in *2018 International Conference on Bangla Speech and Language Processing (ICBSLP)*, IEEE, 2018, pp. 1–6.

[21] A. Howard et al., "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.

[22] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, 2019, pp. 6105–6114.

[23] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[24] S. Banerjee et al., "End-to-end optical character recognition for bengali handwritten words," *arXiv preprint arXiv:2105.04020*, 2021.

[25] T. R. Das, S. Hasan, M. R. Jani, F. Tabassum, and M. I. Islam, "Bangla handwritten character recognition using extended convolutional neural network," *Journal of Computer and Communications*, vol. 9, no. 3, pp. 109–121, 2021.

[26] J. Ferdous, S. Karmaker, A. Rabby, and S. Hossain, "Matrivasha: A multipurpose comprehensive database for bangla handwritten compound characters," in *Emerging Technologies in Data Mining and Information Security*, J. Tavares, S. Chakrabarti, A. Bhattacharya, and S. Ghatak, Eds., ser. Lecture Notes in Networks and Systems, vol. 164, Singapore: Springer, 2021. DOI: 10.1007/978-981-15-9774-9_74.

[27] K. Islam et al., "Bangla optical character recognition (ocr) using deep learning based image classification algorithms," in *2021 24th International Conference on Computer and Information Technology (ICCIT)*, IEEE, 2021, pp. 1–6.

[28] M. M. A. Shibly et al., "Convolutional neural network-based ensemble methods to recognize bangla handwritten characters," *PeerJ Computer Science*, vol. 7, e565, 2021.

[29] S. Ahmed et al., "Efficient approach of using cnn based pretrained model in bangla handwritten digit recognition," *arXiv preprint arXiv:2209.13005*, 2022.

[30] A. Chowdhury and M. Hossain, "Bangla handwritten character recognition using convolutional neural network with data augmentation," in *2022 International Conference on Machine Learning*, 2022, pp. 1–8.

[31] P. Jadhav, S. Gadge, A. Kharde, D. Bhere, and S. Dokare, "Low-cost cnn architecture for bengali character recognition using transfer learning," in *2022 International Conference on Pattern Recognition*, 2022, pp. 1–7.

[32] Z. Liu, H. Mao, C. Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11 976–11 986, 2022.

[33] H. Begum, M. M. Islam, H. S. Eva, N. H. Emon, and F. A. Siddique, "Deep learning networks for handwritten bangla character recognition," *International Journal of Applied Mathematics*, vol. 53, pp. 1–12, 2023.

[34] M. M. Hoque et al., "Bangla handwritten character recognition using mobilenet v1 architecture," *Bulletin of Electrical Engineering and Informatics*, vol. 12, no. 4, pp. 2234–2242, 2023.

[35] A. Rahman et al., "An in-depth analysis of convolutional neural network architectures with transfer learning for skin cancer detection," *Digital Health*, vol. 9, 2023.

[36] S. Sultana et al., "Handwritten bangla character recognition using convolutional neural networks: A comparative study and new lightweight model," *Neural Computing and Applications*, vol. 35, pp. 1–18, 2023.

[37] R. Khan et al., "A vision transformer-based hybrid neural architecture for automated handwritten bangla character recognition and braille conversion," *Knowledge-Based Systems*, vol. 285, pp. 111–125, 2024.

[38] M. Raquib et al., "Vashanet: An automated system for recognizing handwritten bangla characters using deep learning," *Pattern Recognition Letters*, vol. 182, pp. 45–52, 2024.

[39] A. Roy et al., "Resvit: An integrated approach using resnet50v2 and vision transformer for enhanced bangla handwritten character recognition," in *2024 2nd International Conference on Computer Vision and Machine Learning*, IEEE, 2024, pp. 1–6.