Advanced Machine Learning

Final Project Report

Nida Murad

Bengali Handwritten Grapheme Classification

Github: `https://github.com/nida612/AML`

Heidelberg University

June 2, 2024

Contents

1 Introduction

There are approximately 1.3 billion people across regions of South East Asia especially across India, Bangladesh and Thailand that are speakers of languages from alpha-syllabary or Abugida family[1]. Historically, Latin has been at the forefront of optical character recognition (OCR) research for handwritten text. It is particularly difficult to adapt current methods from Latin to alpha-syllabary languages because of how drastically different their orthographies are from one another. Each word in the alpha-syllabary writing system is divided into segments by character units that are phonetically ordered. These units, known as graphemes, serve as the smallest written unit in alpha-syllabary languages[1].

In contrast to English, each character in alpha-syllabary languages like Bengali and Hindi may be composed of a sequence of symbols that do not correspond to the linear positioning of phonemes. Therefore, when creating an OCR for Bengali handwriting, non-linear positioning must be taken into account.

In Bengali, each grapheme can be decomposed into a root, which might be a conjunct of one or numerous characters, a vowel diactric, and/or a consonant diactric, both of which are demarcations which correspond to phonemic extensions.

Let us look at the following example for reference [2], the word ‘Proton’ in both Bengali and Hindi is equivalent to its transliteration

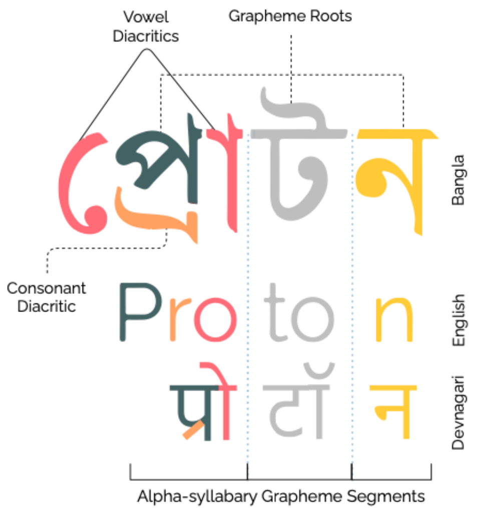


Figure 1: Orthographic components in a Bangla (Bengali) word compared to English and Devnagari (Hindi). Characters are color coded according to phonemic correspondence. Alphasyllabary grapheme segments and corresponding characters from the three languages are segregated with the markers

The grapheme based scheme proposed, pass the complexities of character segmentation inside handwritten alpha-syllabary words. Hence, this problem can be categorized as a multi-target classification problem with three targets (grapheme root, consonant diacritic, vowel diacritic).

2 Literature Review

Most of the available Bengali data sets [13], [14], [15], [16] were made following the contemporary approach used in creating databases of English characters. These contemporary English datasets label specific letters or words and work effectively for document recognition tasks, establishing the benchmark. The majority of character recognition datasets for other languages, such as the Arabic Printed Text Image Database [22],[23] and the Devanagari Character Dataset [19], [20], [21], were developed in accordance with their design. They do not, however, possess the same efficiency and versatility as their English counterparts. As a result, languages with varied writing systems need recognition pipelines that are tailored to them and a deeper understanding of how it affects performance.

Through various research based competitions and challenges, the Bangladesh-based non-profit BengaliAI creates and distributes crowdsourced and metadata-rich datasets. This project is a part of one such kaggle competition which focuses on the multi-target classification aspect of the problem. On the grander scheme of things, this non profit, through this competition, hopes to accelerate Bengali handwritten optical character recognition research and help enable the digitalization of educational resources. The participants of the competition are required to classify given images of Bengali graphemes into independent grapheme roots, vowel diacritics and consonant diacritics.

3 Challenges of Bengali Orthography OCR

Bengali consonant conjuncts are comparable to Latin ligatures in the sense that they join many consonants to create glyphs that may or may not retain the properties of the standalone consonant glyphs. Consonant conjuncts may contain two consonants (second order conjuncts) or three consonants (third order conjuncts). These conjunct glyphs can become very complicated. Allographs are cases where same grapheme can have multiple writing styles. For example in the figure below instead of using the orthodox form for the consonant conjunct as in (b) , a simplified and more explicit form is written in (a).

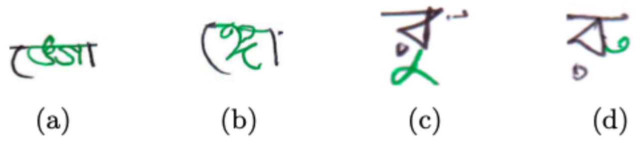


Figure 2: Examples of allograph pairs for the same consonant conjunct (3a and 3b) and the same vowel diacritic (3c and 3d) marked in green (3b and 3d follow an orthodox writing style)[2]

Another challenge is the large number of unique graphemes possible, So the approximate number of possible unique graphemes will be

$$n_v + 3 + \left((n_c - 3)^3 + (n_c - 3)^2 + (n_c - 3)\right) \cdot n_{vd} \cdot n_{cd} = 3883894$$

where (n_c) = 38 consonants including three special characters, (n_v) = 11 vowels and $(n_c^3 + n_c^2)$ possible consonant conjuncts (considering 2nd and 3rd order), there can be $\left((n_c - 3)^3 + (n_c - 3)^2 + (n_c - 3)\right)+3$ different grapheme roots possible in Bengali. Grapheme roots can have any of the 10 + 1 vowel diacritics (n_{vd}) and 7 + 1 consonant diacritics (n_{cd}). Not all of these combinations are viable or are used in practice. [2]

4 Solution Approach

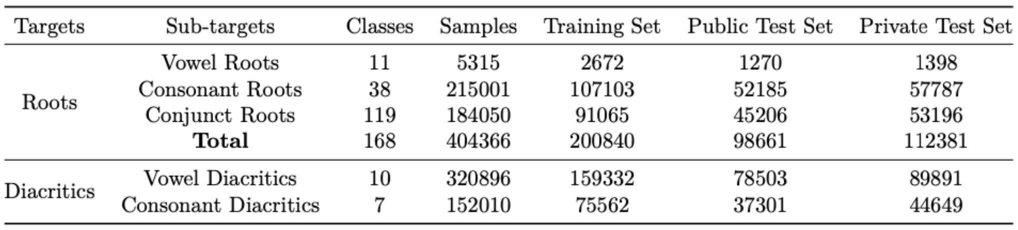
We need to develop a multi-target classifier which, given an image (flattened into an integer array), is capable of accurately predicting its corresponding components i.e. grapheme root, vowel diactric and consonant diactric. We do so by leveraging the publicly available pytorch classification models (Resnet, EffecientNet, DenseNet etc) , pretrained on ImageNet, as our base models which we further train over our specific dataset.

Formally, we consider our dataset, D=$\{s_1, s_2, \ldots, s_N\}$ as composed of N data points $s_i = \{x_i, y_i^r, y_i^{vd}, y_i^{cd}\}$. Each datum s_i consists of an image, $x_i \in \mathbb{R}_{H\times W}$ and a subset of three target variables y_i^r, y_i^{vd} and y_i^{cd} denoting the ground truth for roots, vowel diacritics and consonant diacritics respectively, i.e., $y_i^r \in \mathscr{R}, y_i^{vd} \in \mathscr{V}$, and $y_i^{cd} \in \mathscr{C}$. Here, $\mathscr{R}, \mathscr{C}$, and $\mathscr{V}$ is the set of roots, vowel diacritics and consonant diacritics, respectively, where $|\mathscr{R}| = N_r, |\mathscr{C}| = N_{cd}$, and $|\mathscr{V}| = N_{vd}$. The multi-target classification task, thus will consist of generating a classifier h which, given an image x_i, is capable of accurately predicting its corresponding components, i.e., $h\,(x_i) = \{y_i^r, y_i^{vd}, y_i^{cd}\}$.[2]

4.1 Dataset

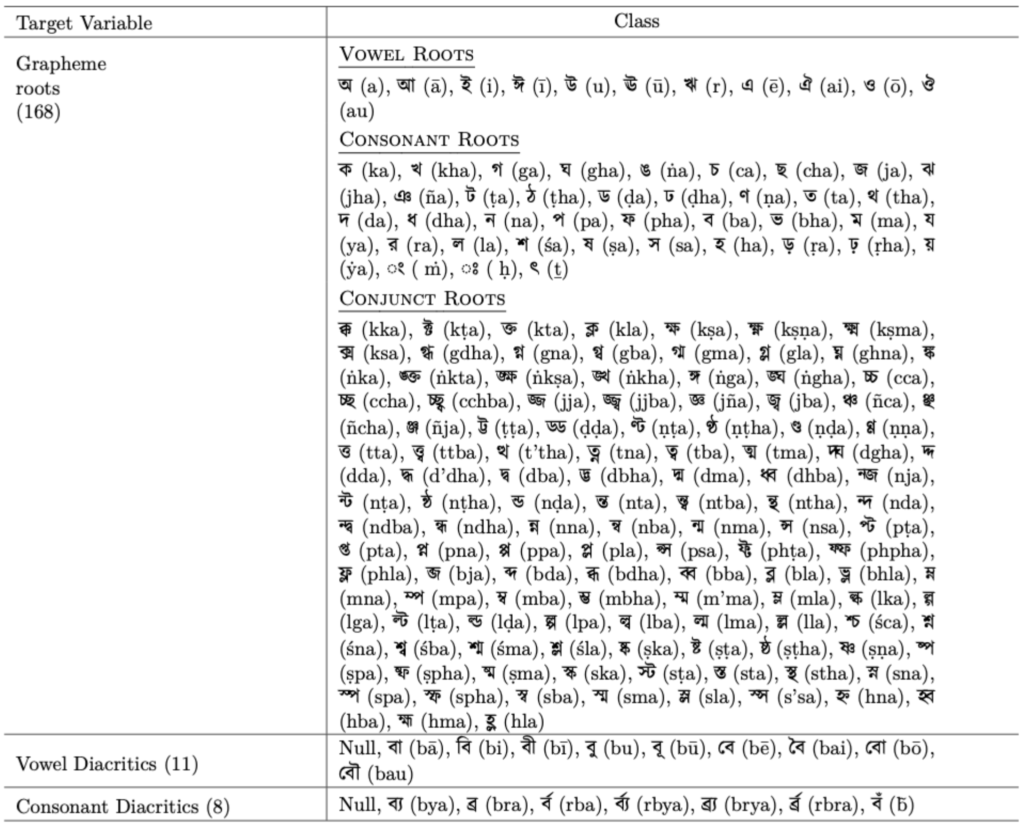
The dataset provided utilizes the Google Bengali ASR dataset [3] as a reference corpus, out of which only popular graphemes were chosen. The ASR dataset contains a large volume of transcribed Bengali speech data.

A breakdown of the composition of the training set, targets (that is roots into vowels, consonants and conjuncts and diacritics into vowel and consonant) along with the number of unique classes of the dataset is given in the table below. The final dataset contains handwritten graphemes of size 137 x 236 pixels.



Targets	Sub-targets	Classes	Samples	Training Set	Public Test Set	Private Test Set
Roots	Vowel Roots	11	5315	2672	1270	1398
	Consonant Roots	38	215001	107103	52185	57787
	Conjunct Roots	119	184050	91065	45206	53196
	Total	168	404366	200840	98661	112381
Diacritics	Vowel Diacritics	10	320896	159332	78503	89891
	Consonant Diacritics	7	152010	75562	37301	44649

Figure 3: Decomposition of the dataset used [2]



Target Variable	Class
Grapheme roots (168)	VOWEL ROOTS অ (a), আ (ā), ই (i), ঈ (ī), উ (u), ঊ (ū), ঋ (r), এ (ē), ঐ (ai), ও (ō), ঔ (au) CONSONANT ROOTS ক (ka), খ (kha), গ (ga), ঘ (gha), ঙ (ṅa), চ (ca), ছ (cha), জ (ja), ঝ (jha), ঞ (ña), ট (ṭa), ঠ (ṭha), ড (ḍa), ঢ (ḍha), ণ (ṇa), ত (ta), থ (tha), দ (da), ধ (dha), ন (na), প (pa), ফ (pha), ব (ba), ভ (bha), ম (ma), য (ya), র (ra), ল (la), শ (śa), ষ (ṣa), স (sa), হ (ha), ড় (ṛa), ঢ় (ṛha), য় (ẏa), ং (ṁ), ঃ (ḥ), ৎ (t̲) CONJUNCT ROOTS ক্ক (kka), ক্ট (kṭa), ক্ত (kta), ক্ল (kla), ক্ষ (kṣa), ক্ষ্ণ (kṣṇa), ক্ষ্ম (kṣma), ক্স (ksa), গ্ধ (gdha), গ্ন (gna), গ্ব (gba), গ্ম (gma), গ্ল (gla), ঘ্ন (ghna), ঙ্ক (ṅka), ঙ্ক্ত (ṅkta), ঙ্ক্ষ (ṅkṣa), ঙ্খ (ṅkha), ঙ্গ (ṅga), ঙ্ঘ (ṅgha), চ্চ (cca), চ্ছ (ccha), চ্ছ্ব (cchba), জ্জ (jja), জ্জ্ব (jjba), জ্ঞ (jña), জ্ব (jba), ঞ্চ (ñca), ঞ্ছ (ñcha), ঞ্জ (ñja), ট্ট (ṭṭa), ড্ড (ḍḍa), ণ্ট (ṇṭa), ণ্ঠ (ṇṭha), ণ্ড (ṇḍa), ণ্ণ (ṇṇa), ত্ত (tta), ত্ত্ব (ttba), ত্থ (t’tha), ত্ন (tna), ত্ব (tba), ত্ম (tma), দ্ঘ (dgha), দ্দ (dda), দ্ধ (d’dha), দ্ব (dba), দ্ভ (dbha), দ্ম (dma), ধ্ব (dhba), ন্জ (nja), ন্ট (nṭa), ন্ঠ (nṭha), ন্ড (nḍa), ন্ত (nta), ন্ত্ব (ntba), ন্থ (ntha), ন্দ (nda), ন্দ্ব (ndba), ন্ধ (ndha), ন্ন (nna), ন্ব (nba), ন্ম (nma), ন্স (nsa), প্ট (pṭa), প্ত (pta), প্ন (pna), প্প (ppa), প্ল (pla), প্স (psa), ফ্ট (phṭa), ফ্ফ (phpha), ফ্ল (phla), ব্জ (bja), ব্দ (bda), ব্ধ (bdha), ব্ব (bba), ব্ল (bla), ভ্ল (bhla), ম্ন (mna), ম্প (mpa), ম্ব (mba), ম্ভ (mbha), ম্ম (m’ma), ম্ল (mla), ল্ক (lka), ল্গ (lga), ল্ট (lṭa), ল্ড (lḍa), ল্প (lpa), ল্ব (lba), ল্ম (lma), ল্ল (lla), শ্চ (śca), শ্ন (śna), শ্ব (śba), শ্ম (śma), শ্ল (śla), ষ্ক (ṣka), ষ্ট (ṣṭa), ষ্ঠ (ṣṭha), ষ্ণ (ṣṇa), ষ্প (ṣpa), ষ্ফ (ṣpha), ষ্ম (ṣma), স্ক (ska), স্ট (sṭa), স্ত (sta), স্থ (stha), স্ন (sna), স্প (spa), স্ফ (spha), স্ব (sba), স্ম (sma), স্ল (sla), স্স (s’sa), হ্ন (hna), হ্ব (hba), হ্ম (hma), হ্ল (hla)
Vowel Diacritics (11)	Null, বা (bā), বি (bi), বী (bī), বু (bu), বূ (bū), বে (bē), বৈ (bai), বো (bō), বৌ (bau)
Consonant Diacritics (8)	Null, ব্য (bya), ব্র (bra), র্ব (rba), র্ব্য (rbya), ব্র্য (brya), র্ব্র (rbra), বঁ (b̆)

Figure 4: Table of all target variables and classes present in the dataset

Files given for the challenge are:

a) **train.csv**

- image_id: the foreign key for the parquet files
- grapheme_root: the first of the three target classes
- vowel_diacritic: the second target class
- consonant_diacritic: the third target class
- grapheme: the complete character. Provided for informational purposes only, you should not need to use this.

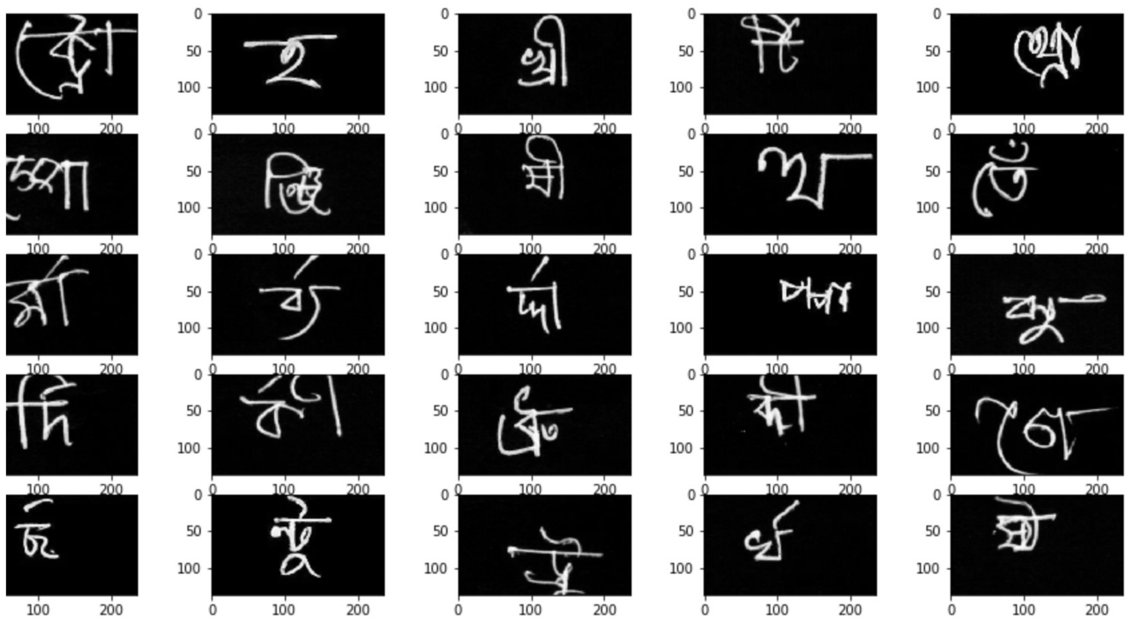


Figure 5: A glimpse of the first 25 images of the training data set rendered.

b) **test.csv**: Every image in the test set will require three rows of predictions, one for each component. This csv specifies the exact order for you to provide your labels.

- row_id: foreign key to the sample submission
- image_id: foreign key to the parquet file
- component: the required target class for the row (grapheme_root, vowel_diacritic, or consonant_diacritic)

c) **sample_submission.csv**

- row_id: foreign key to test.csv
- target: the target column

d) **(train/test).parquet**: Each parquet file contains tens of thousands of 137x236 grayscale images. The images have been provided in the parquet format for I/O and space efficiency. Each row in the parquet files contains an image_id column, and the flattened image.

e) **class_map.csv**

Maps the class labels to the actual Bengali grapheme components.

4.2 Architecture and Methodology

In this section, we drawa detailed overview of the models used for the multi-target classification task.

4.2.1 ResNet

It makes sense, according to researchers, to say that convolutional neural networks are best when they are deeper, since they have a larger parameter space to explore, their ability to adapt to any space increases. When deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly.[7] Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error, as reported in[5][6] The infamous vanishing gradient is one of the issues that ResNets addresses.

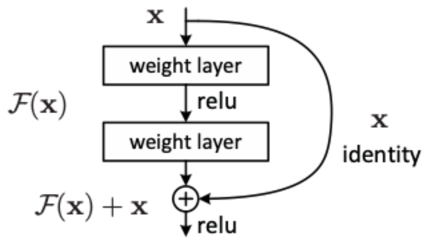

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 6: Residual learning: a building block

In the figure above, we see a building block which we define as,

$$\mathbf{y} = \mathscr{F}(x, \{W_i\}) + \mathbf{x}, \tag{1}$$

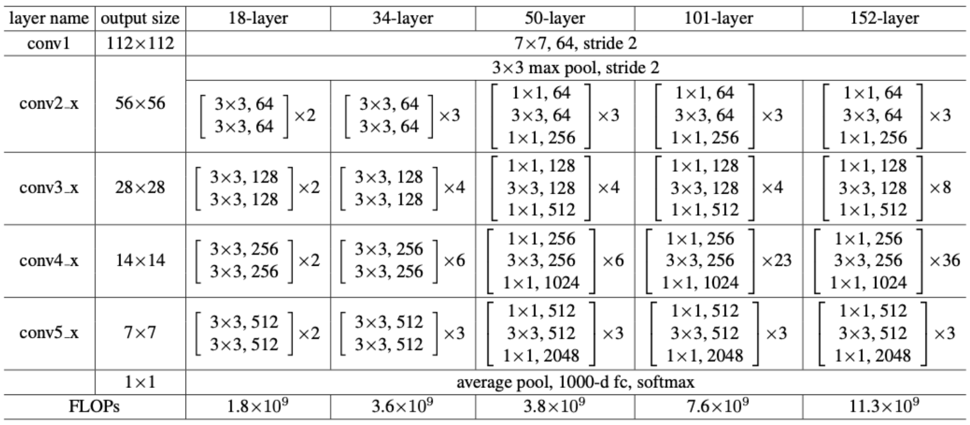
where $\mathbf{x}$ and $\mathbf{y}$ denote the input and output vectors of the layers considered. The residual mapping to be learned is given by $\mathscr{F}(\mathbf{x}, \{W_i\})$. We then adopt residual learning to every few stacked layers. For the two-layer example in Fig. 4, $\mathscr{F} = W_2\sigma(W_1\mathbf{x})$, where σ stands for ReLU and the biases are excluded to make the notation simpler. By using a shortcut connection and element-wise addition, the operation $\mathbf{F} + \mathbf{x}$ is accomplished. As seen in Figure 4, we adopt the second non-linearity after the addition that is $\sigma(y)$.

(paraphrase) The dimensions of $\mathbf{x}$ and $\mathscr{F}$ must be equal in Eqn.(1). If this is not the case (e.g., when changing the input/output channels), we can perform a linear projection W_s by the shortcut connections to match the dimensions:

$$\mathbf{y} = \mathscr{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}. \tag{2}$$

We can also use a square matrix W_s in Eqn.(1). But the identity mapping is sufficient for addressing the degradation problem and is economical, and thus W_s is only used when matching dimensions.

The form of the residual function $\mathscr{F}$ is flexible. If $\mathscr{F}$ has only a single layer, Eqn.(1) is similar to a linear layer: $\mathbf{y} = W_1\mathbf{x} + \mathbf{x}$, for which we have not observed advantages. We also note that although the above notations are about fully-connected layers for simplicity, they are applicable to convolutional layers. The function $\mathscr{F}(\mathbf{x}, \{W_i\})$ can represent multiple convolutional layers. The element-wise addition is performed on two feature maps, channel by channel.[7]



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 7: Sizes of outputs and convolutional kernels for ResNet 34

In the table, there is a summary of the output size at every layer and the dimension of the convolutional kernels at every point in the structure.[8]

4.2.2 Efficient Net

Convolution Networks are frequently scaled up in order to improve accuracy. It can be scaled up by their depth [9], width [10], or even image resolution [11]. Nevertheless the whole process has never been effectively understood. Previously, it was usual to scale one of the three dimensions, even though it is possible to scale two or three dimensions arbitrarily, doing so often results in subpar accuracy and efficiency and needs time-consuming manual adjustment. In previous work, it is common to scale only one of the three dimensions – depth, width, and image size. Though it is possible to scale two or three dimensions arbitrarily, arbitrary scaling requires tedious manual tuning and still often yields sub-optimal accuracy and efficiency. Intuitively, the compound scaling method makes sense be- cause if the input image is bigger, then the network needs more layers to increase the receptive field and more channels to capture more fine-grained patterns on the bigger image.[12]

(paraphrase) A ConvNet Layer i can be defined as a function: $Y_i = \mathcal{F}_i(X_i)$, where $\mathcal{F}_i$ is the operator, Y_i is output tensor, X_i is input tensor, with tensor shape $\langle H_i, W_i, C_i \rangle^1$, where H_i and W_i are spatial dimension and C_i is the channel dimension. A ConvNet $\mathcal{N}$ can be represented by a list of composed layers:

$$\mathcal{N} = \mathcal{F}_k \odot \ldots \odot \mathcal{F}_2 \odot \mathcal{F}_1(X_1) = \bigodot_{j=1\ldots k} \mathcal{F}_j(X_1).$$

Therefore, we can define a ConvNet as:

$$\mathcal{N} = \bigodot_{i=1\ldots s} \mathcal{F}_i^{L_i}\left(X_{\langle H_i, W_i, C_i \rangle}\right).$$

where $\mathcal{F}_i^{L_i}$ denotes layer F_i is repeated L_i times in stage i, $\langle H_i, W_i, C_i \rangle$ denotes the shape of input tensor X of layer i.

Unlike regular ConvNet designs that mostly focus on finding the best layer architecture $\mathcal{F}_i$, model scaling tries to expand the network length (L_i), width (C_i), and/or resolution (H_i, W_i) without changing $\mathcal{F}_i$ predefined in the baseline network. By fixing $\mathcal{F}_i$, model scaling simplifies the design problem for new resource constraints, but it still remains a large design space to explore different L_i, C_i, H_i, W_i for each layer. In order to further reduce the design space, we restrict that all layers must be scaled uniformly with constant ratio. Our target is to maximize the model accuracy for any given resource constraints, which can be formulated as an optimization problem:

$$
\begin{aligned}
&\max_{d,w,r} \ \text{Accuracy}(\mathcal{N}(d,w,r)) \\
&\text{s.t.} \ \ \mathcal{N}(d,w,r) = \bigodot_{i=1\ldots s} \hat{\mathcal{F}}_i^{d\cdot\hat{L}_i}\left(X_{\langle r\cdot\hat{H}_i, r\cdot\hat{W}_i, w\cdot\hat{C}_i\rangle}\right) \\
&\qquad \text{Memory}(\mathcal{N}) \leq targetmemory \\
&\qquad \text{FLOPS}(\mathcal{N}) \leq targetflops
\end{aligned}
$$

where w, d, r are coefficients for scaling network width, depth, and resolution; $\hat{\mathcal{F}}_i, \hat{L}_i, \hat{H}_i, \hat{W}_i, \hat{C}_i$ are predefined parameters in baseline network.

We propose a new compound scaling method, which use a compound coefficient ϕ to uniformly scales network width, depth, and resolution in a principled way:

$$
\begin{aligned}
&\text{depth: } d = \alpha^{\phi} \\
&\text{width: } w = \beta^{\phi} \\
&\text{resolution: } r = \gamma^{\phi}
\end{aligned}
$$

where α, β, γ are constants, with

$$
\begin{gathered}
\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\
\text{and} \\
\alpha \geq 1, \beta \geq 1, \gamma \geq 1.
\end{gathered}
$$

These constants can be determined by a small grid search. Intuitively, ϕ is a user-specified coefficient that controls how many more resources are available for model scaling, while α, β, γ specify how to assign these extra resources to network width, depth, and resolution respectively. Notably, the FLOPS of a regular convolution op is proportional to d, w^2, r^2, i.e., doubling network depth will double FLOPS, but doubling network width or resolution will increase FLOPS by four times. Since convolution ops usually dominate the computation cost in ConvNets, scaling a ConvNet with equation 3 will approximately increase total FLOPS by $\left(\alpha \cdot \beta^2 \cdot \gamma^2\right)^{\phi}$ [12].

Based on the baseline network created by the neural architecture search utilizing the AutoML MNAS framework, EfficientNet was created. The network is adjusted for optimum accuracy, but it is also penalized if the network is very computationally heavy. It is punished for slow inference time as well. Due to the increase in FLOPS, the architecture uses a mobile inverted bottleneck convolution that is larger than MobileNet V2. To create the EfficientNet family, this basic model is scaled up.

5 Implementation

To be able to efficiently train our models we first perform certain data-preprocessing steps. We then configure our models to align with our target classes. Finally, we also use an optimizer and a scheduler(EarlyStopping) to further improvise the training process.

5.1 Pre-Processing data

The images stored in parquet files are pickled to improve the efficiency of the image loading by image_id. We also segment the data into folds for cross validation which enhances the model evaluation giving better results.

5.1.1 Creating folds for Cross Validation(Mutli-level Classification)

Model evaluation is often performed with a hold-out split, where an often 80/20 split is made and where 80% of your dataset is used for training the model. and 20% for evaluating the model. While this is a simple approach, it is also very naïve, since it assumes that data is representative across the splits, that it's not a time series dataset and that there are no redundant samples within the datasets.

K-fold Cross Validation is a more robust evaluation technique. It splits the dataset in k-1 training batches and 1 testing batch across k folds, or situations. Using the training batches, you can then train your model, and subsequently evaluate it with the testing batch. This allows you to train the model for multiple times with different dataset configurations.

We use the package MultilabelStratifiedKFold() which uses an algorithm which was proposed in [17]. It is referenced in the paper [17] as Iterative Stratification that splits a multi-label dataset by considering each label separately, starting from the one with the fewest positive examples and working its way to the best represented one. Here, we have taken the number of folds k as 5, where every fold has 40168 samples, that is in total we have over 200000 samples for our training phase.

5.1.2 Pickling, Transformations and Augmentations

Pickle is the native format of python that is popular for object serialization. The advantage of pickle is that it allows the python code to implement any type of enhancements [27]. It is much faster when compared to parquet and reduces the file size significantly using its compression techniques. This significantly enhances the reading process.

For training, along with the transforms, we are also augmenting the images to prevent over-fitting. Without augmentations, the validation accuracy was still fairly low whereas the training accuracy quickly reached high levels. These additions are therefore mostly the result of experiments to see what improved the data set. These transformation and augemtnation steps are performed during the data loading (using pytorch dataset and dataloader APIs) while training, evaluating and inferring.

The mean and standard deviation parameters used, to perform the normalization augmentation, are the same as the ones that our pretrained models had been trained on, oover the ImageNet dataset

5.2 Model Set Up

Resnet34 is one of the very basic and widely used models for image classsification. It is a modern image classification model that consists of a 34 layer convolutional neural network. This model has already been pre-trained using the 100,000+ photos from 200 different classifications that make up the ImageNet dataset. For our use-case, we have changed the final layers to match with our target classes.

Additionally, we also implement Efficient Net B3 whose architectural structure is given below and follows the basic methodology as described in Section 4.2 except the number of parameters. Both model architectures have been generated and can be found in 7

5.3 Optimizer and Scheduler

Optimizer - We have used Adam Optimizer which computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.[25]. Adam combines the benefits of two further extensions of stochastic gradient descent which are AdaGrad or Adaptive Gradient and Root Mean Square Propagation (RMSProp) We use the package torch.optim.Adam to construct an optimizer object, that will hold the current state and will update the parameters based on the computed gradients. We pass the package the learning rate (0.0001) and the model paramaters.

Scheduler - EarlyStopping stops the training if validation loss doesn't improve after a given patience, where patience is nothing but how long one has to wait after the last time validation loss was improved.

5.4 Training

The training function that has been used is a standard image classification training function in PyTorch where we perform the following steps:

- Forward pass
- Calculate the losses - The loss function employed here is Cross Entropy,
- calculate the accuracies - the accuracy scores are calculated using macro recall,
- Backpropagate the gradients
- Update the optimizer parameters.

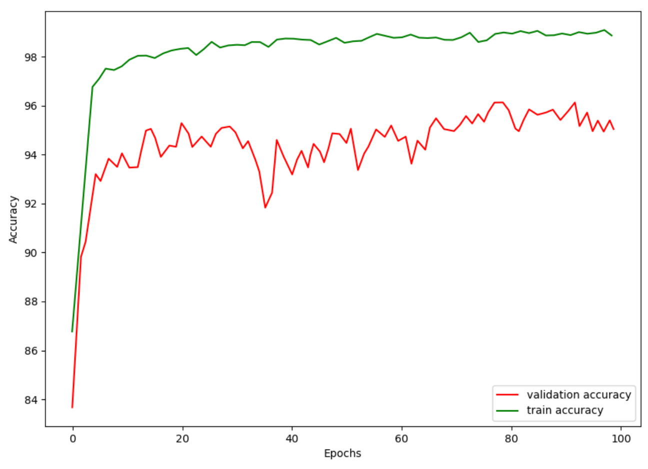


Figure 8: ResNet34 Accuracy for fold 0

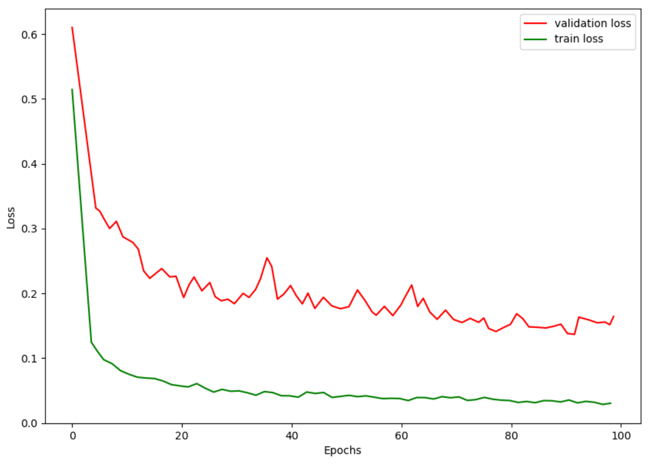


Figure 9: ResNet34 Loss for fold 0

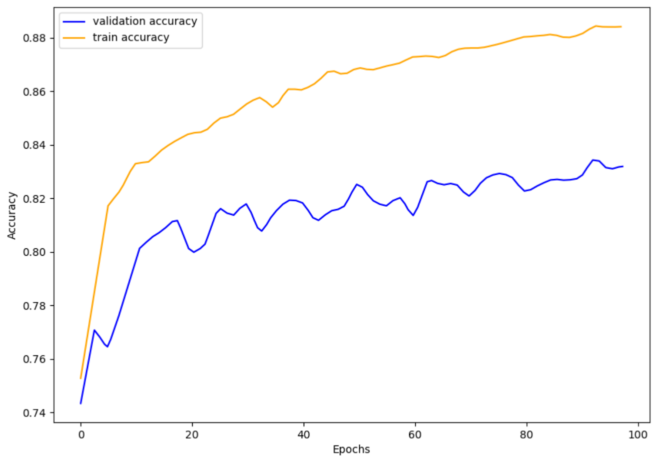


Figure 10: EfficientNetB3 Accuracy for fold 0

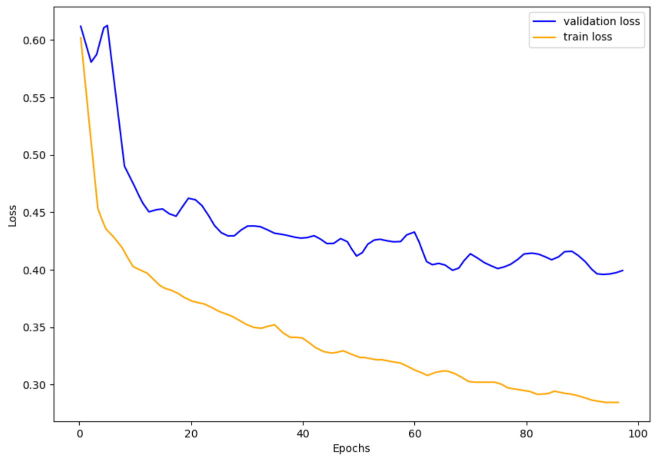


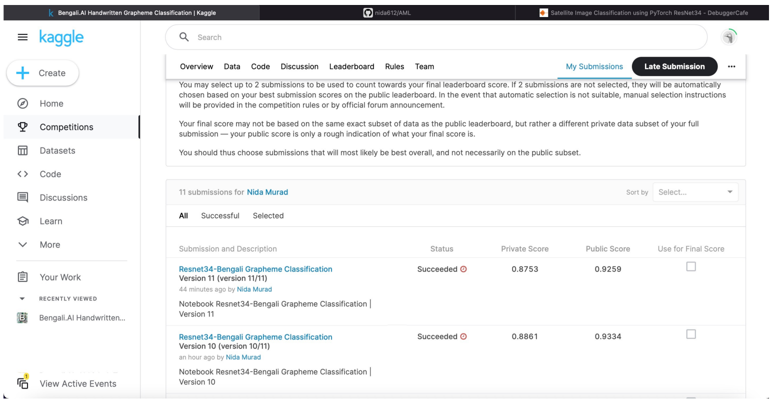
Figure 11: EfficientNetB3 loss for fold 0

6 Summary and Outlook

6.1 Competition Metric

The metric for the challenge is a hierarchical macro- averaged recall. First, a standard macro-averaged re-call is calculated for each component. Let the macroaveraged recall for grapheme root, vowel diacritic, and consonant diacritic be denoted by R_r, R_{vd}, and R_{cd} respectively. The final score R is the weighted average

$$R = \frac{1}{4}\left(2R_r + R_{vd} + R_{cd}\right)$$



You may select up to 2 submissions to be used to count towards your final leaderboard score. If 2 submissions are not selected, they will be automatically chosen based on your best submission scores on the public leaderboard. In the event that automatic selection is not suitable, manual selection instructions will be provided in the competition rules or by official forum announcement.

Your final score may not be based on the same exact subset of data as the public leaderboard, but rather a different private data subset of your full submission — your public score is only a rough indication of what your final score is.

You should thus choose submissions that will most likely be best overall, and not necessarily on the public subset.

11 submissions for Nida Murad

Submission and Description	Status	Private Score	Public Score	Use for Final Score
Resnet34-Bengali Grapheme Classification Version 11 (version 11/11) 44 minutes ago by Nida Murad Notebook Resnet34-Bengali Grapheme Classification | Version 11	Succeeded	0.8753	0.9259	☐
Resnet34-Bengali Grapheme Classification Version 10 (version 10/11) an hour ago by Nida Murad Notebook Resnet34-Bengali Grapheme Classification | Version 10	Succeeded	0.8861	0.9334	☐

Figure 12: Competition Results

6.2 Conclusion

The Kaggle competition resulted in 31,002 submissions from 2,059 teams consisting of 2,623 competitors. For the competition we were suppposed to submit a notebook sample_submission.csv The submission notebook we formulated was in such a way that the base model can be supplied as a script parameter, making the code versatile and re-usable for any model. Now we notice from Figure 12 that EfficientNet performed better than the ResNet34 if we compare the public and private scores. In lieu of component recognition, the winner who had a public score of 0.995 and a private score of 0.976 and bagged the first place in the competition used grapheme classification. The input visuals were divided into 14784 classes (168 x 11 x 8), or all conceivable graphemes that could be made using the given grapheme roots and diacritics. Initial classification of the graphemes was performed using an EfficientNet[12] model. The sample, on the other hand, is regarded as an OOD grapheme and is sent to the pipeline for OOD grapheme classification if the network is not confident in its prediction. A CycleGan [26] is used in the pipeline to translate handwritten graphemes into font produced graphemes. An EfficientNet classifier trained on a 14784 class synthetic grapheme dataset is used as an OOD grapheme classifier.

If there was more time, other models like ResNet50 or EfficientNetB7 could have been further explored.

References

[1] Liudmila Fedorova. The development of graphic representation in abugida writing: The akshara's grammar. Lingua Posnaniensis, 55(2):49–66, Dec 2013

[2] https://arxiv.org/abs/2010.00170

[3] Oddur Kjartansson, Supheakmungkol Sarin, Knot Pi- patsrisawat, Martin Jansche, and Linne Ha. Crowd- sourced speech corpora for javanese, sundanese, sin- hala, nepali, and bangladeshi bengali. In Proc. 6th Intl. Workshop SLTU, pages 52–55, 08 2018.

[4] https://www.run.ai/guides/deep-learning-for-computer-vision/pytorch-resnet

[5] K.HeandJ.Sun.Convolutionalneuralnetworksatconstrainedtime cost. In CVPR, 2015.

[6] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. arXiv:1505.00387, 2015.

[7] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in CVPR, 2016.

[8] https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8

[9] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. CVPR, pp. 770–778, 2016.

[10] Zagoruyko, S. and Komodakis, N. Wide residual networks. BMVC, 2016.

[11] Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J., Le, Q. V., and Chen, Z. Gpipe: Efficient training of giant neural networks using pipeline parallelism. arXiv preprint arXiv:1808.07233, 2018.

[12] EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks https://arxiv.org/pdf/1905.11946.pdf

[13] Samiul Alam, Tahsin Reasat, Rashed Mohammad Doha, and Ahmed Imtiaz Humayun. Numtadb - as- sembled bengali handwritten digits. arXiv preprint arXiv:1806.02452, 2018

[14] Mithun Biswas, Rafiqul Islam, Gautam Kumar Shom, Md Shopon, Nabeel Mohammed, Sifat Momen, and Md Anowarul Abedin. Banglalekha-isolated: A comprehensive bangla handwritten character dataset. arXiv preprint arXiv:1703.10661, 2017.

[15] AKM Shahariar Azad Rabby, Sadeka Haque, Md. Sanzidul Islam, Sheikh Abujar, and Syed Akhter Hossain. Ekush: A multipurpose and multitype com- prehensive database for online off-line bangla hand- written characters. In RTIP2R, pages 149–158, 2019.

[16] Ram Sarkar, Nibaran Das, Subhadip Basu, Mahanta- pas Kundu, Mita Nasipuri, and Dipak Kumar Basu. Cmaterdb1: a database of unconstrained handwritten bangla and bangla–english mixed script document im- age. IJDAR, 15(1):71–83, 2012.

[17] On the Stratification of Multi-Label Data-Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas http://lpis.csd.auth.gr/publications/sechidis-ecmlpkdd-2011.pdf

[18] https://www.analyticsvidhya.com/blog/2021/09/building-resnet-34-model-using-pytorch-a-guide-for-beginners/

[19] S. Acharya, A. K. Pant, and P. K. Gyawali. Deep learning based large scale handwritten devanagari character recognition. In Proc. 9th Int. Conf. SKIMA), pages 1–6, Dec 2015.

[20] Sk Md Obaidullah, Chayan Halder, Nibaran Das, and Kaushik Roy. A new dataset of word-level offline hand- written numeral images from four official indic scripts and its benchmarking using image transform fusion. Int. J. Intell. Eng. Inform., 4(1):1–20, Feb. 2016

[21] K. Dutta, P. Krishnan, M. Mathew, and C. V. Jawa- har. Offline handwriting recognition on devanagari us- ing a new benchmark dataset. In Proc. 13th IAPR Int. Workshop DAS, pages 25–30, April 2018

[22] Reza Farrahi Moghaddam, Mohamed Cheriet, Math- ias M. Adankon, Kostyantyn Filonenko, and Robert Wisnovsky. Ibn sina: A database for research on pro- cessing and understanding of arabic manuscripts im- ages. In Proc. 9th IAPR Int. Workshop DAS, page 11–18, 2010. 2

[23] Jawad H. AlKhateeb. A database for arabic handwrit- ten character recognition. Procedia Computer Science, 65:556 – 561, 2015. 2, 4

[24] https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

[25] https://arxiv.org/pdf/1412.6980.pdf

[26] J. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent ad- versarial networks. In Proc. ICCV, 2017, pages 2242– 2251, Oct 2017. 7

[27] https://analyticsindiamag.com/complete-guide-to-different-persisting-methods-in-pandas/

7 Appendix

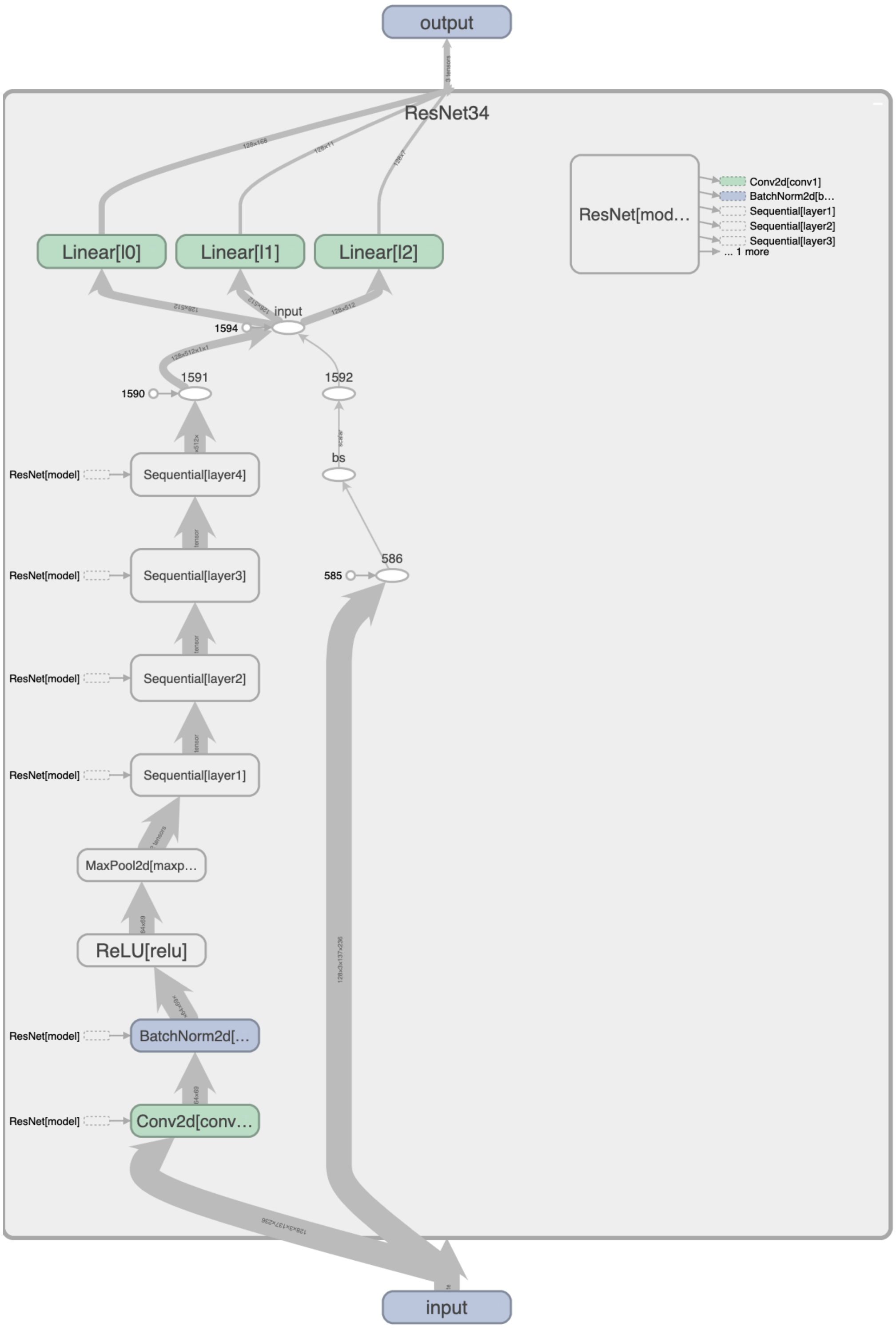


Figure 13: Curated model architecture of the ResNet34

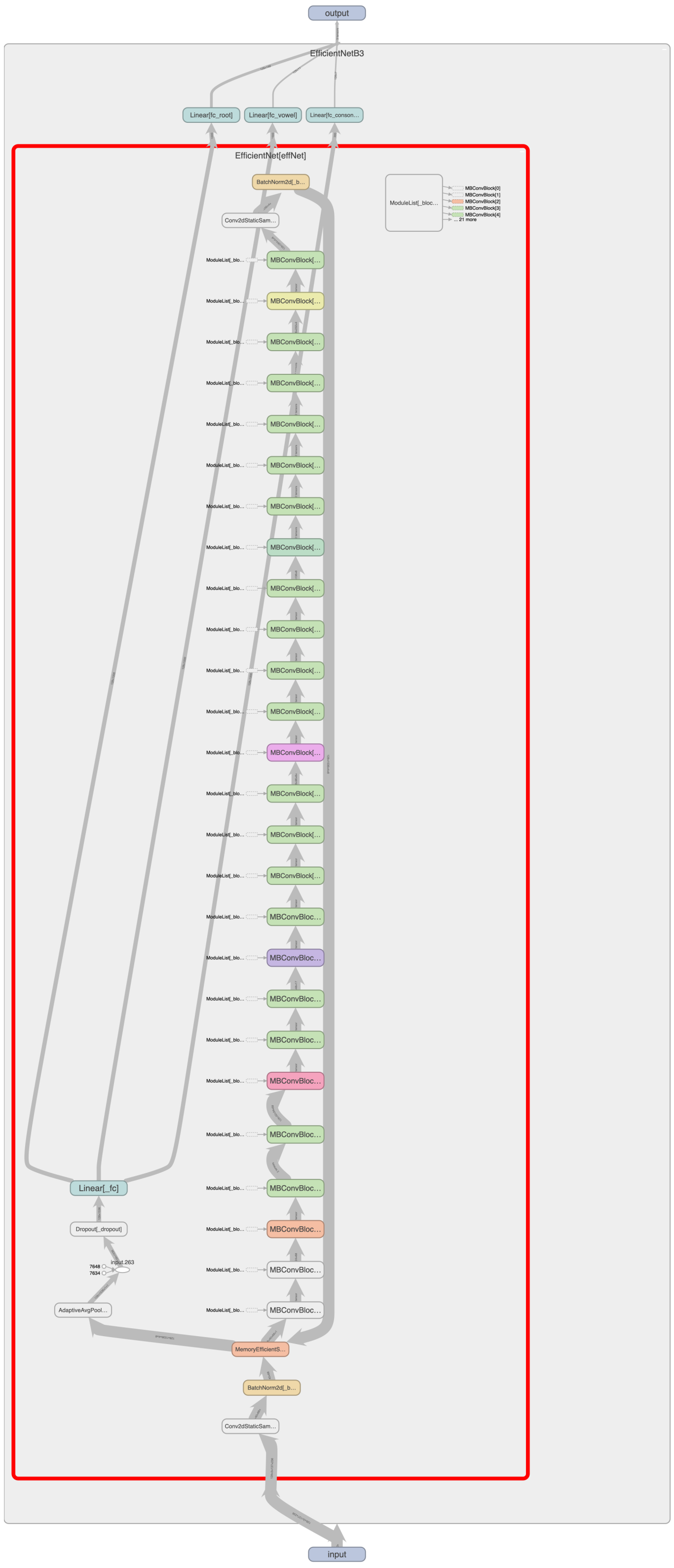


Figure 14: Curated model architecture of the EfficientNetB3