SUTD 50.035 Computer Vision

Final Project

Kaggle Bengali.AI Handwritten Grapheme Classification

Gary Ong (1002758)
Lu Jiankun (1002959)
Peng ShanShan (1002974)
Joseph Chan (1002948)
Nashita Abd Guntaguli (1003045)
Hong Pengfei (1002949)

Contents

1 Introduction

1.1 Problem Framing

Bengali is spoken by hundreds of millions of people, making it the 5^{th} most spoken language in the world. As such, achieving reliable optical handwritten character recognition algorithm can result in advancements in business and education that are extremely beneficial for this population. However, character recognition has proven to be very challenging as the language comprises of nearly 13000 possible graphemes, compared to the 250 graphemes of English. Further more, the Bengali graphemes can be broken down into its roots, vowel diacritics, and consonant diacritics, all of which are not segregated and arranged horizontally like that of the English language. We can refer to Figure 1 for a compilation of possible grapheme roots, vowel diacritics, and consonant diacritics.

Label	Class
Grapheme roots (168)	অ, আ, ই, ঈ, উ, ঊ, ঋ, এ, ঐ, ও, ঔ, ক, ক্ক, ক্ট, ক্ত, ক্ল, ক্ষ, ক্ষ, ক্ষ্ম, ক্স, খ, গ, গ্ধ, গ্ন, গ্ব, গ্ম, গ্ল, ঘ, ঘ্ন, ঙ, ঙ্ক,ঙ্ক্ত, ঙ্ক্ষ, ঙ্খ, ঙ্গ, ঙ্ঘ, চ, চ্ছ, চ্ছ্ব, চ্ছ্ব, জ, জ্ঞ, জ্ব, জ্ঝ, ঞ, ঞ্ছ, ঞ্জ, ঞ্ঝ, ট, ট্ট, ঠ, ড, ড়, ড্ড, ঢ, ঢ়, ণ, ণ্ট, ণ্ঠ, ত, ত্ত, ত্ত্ব, ত্থ, ত্ন, ত্ব, ত্ম, থ, দ, দ্ঘ, দ্দ, দ্ধ, দ্ব, দ্ভ, দ্ম, ধ, ধ্ব, ন, ন্জ, ন্ট, ন্ঠ, ন্ড, ন্ত, ন্ত্ব, ন্থ, ন্দ, ন্দ্ব, ন্ধ, ন্ন, ন্ব, ন্ম, ন্স, প, প্ট, প্ত, প্ন, প্প, প্ল, প্স, ফ, ফ্ট, ফ্ফ, ফ্ল, ব, ব্জ, ব্দ, ব্ধ, ব্ব, ব্ল, ভ, ভ্ন, ম, ম্ন, ম্প, ম্ফ, ম্ভ, ম্ম, ম্ল, য, য়, র, ল, ল্ড, শ, শ্ব, ষ, স, স্ট, স্ত, হ, হ্ণ, হ্ন, হ্ম, হ্ল, হ্ব, হ্ম, হ্ল, শ্চ, শ্ন, শ্ম, শ্ল, ষ্ক, ষ্ট, ষ্ঠ, ষ্ণ, ষ্প, ষ্ফ, ষ্ম, ষ্ক, স্ত, স্ন, স্প, স্ফ, স্ব, স্ম, স্ল, স্স, হ্ন, হ্ম, হ্ল, ৎ, ংং, ঃঃ
Vowel Diacritics (11)	Null, া, ি, ী, ু, ূ, ৃ, ে, ৈ, ো, ৌ
Consonant Diacritics (7)	Null, ্য (য-ফলা), ্র (র-ফলা), র্ (রেফ), ঁ, র্্য (রেফ য-ফলা), ্র্য (র-ফলা য-ফলা)

Figure 1: Grapheme Labels

1.2 Problem Statement

Thus, with this unique set of problems, we are given the following problem statement:
Based on a given image of a handwritten Bengali grapheme, to separately classify three constituent elements in the image: grapheme root, vowel diacritics, and consonant diacritics.

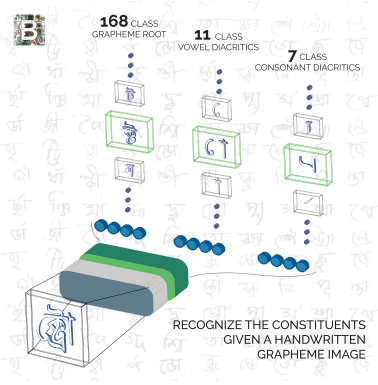


Figure 2: Constituents of a Bengali Grapheme Image

1.3 Dataset

Breakdown: Dataset Reports

Out of the 12936 possible graphemes (168*11*7), the provided training data set contains only 1295 unique graphemes, which were statistically determined as part of common everyday lexicon. The private undisclosed test set from Kaggle contains unique grapheme combinations that are not included in this training data set. Thus, our model approach should not only aim to achieve high accuracy on these 1295 graphemes, but also learn to generalize well and obtain good performance on unseen unique graphremes.

In order to gauge our performance of the model during training, we need to be able to assess it on both seen (part of the 1295 unique graphemes) and unseen (not part of the 1295 unique graphemes) graphemes. Thus, we decided to remove a small sample of combinations from the training set and use them to simulate unseen data. The remaining data is split into training and validation set.

The final split are as follows:

- Training set: 77%
- Validation seen set: 20%
- Validation unseen set 3%

To be perfectly clear, there exists multiple handwritten images for each Bengali grapheme. The graphemes that exist in the validation seen set also exist in training set, but they do not share the same handwritten images. On the other hand, the graphemes in the validation unseen set have been manually isolated by us, and none of its handwritten images exist in the training set, thereby simulating true unseen data.

Data Generation

In addition, we have also generated new training images that are utilized in some experiments. This has the potential to help the model learn features of graphemes that would otherwise be unseen. To approach this task, we first downloaded a dictionary of all possible Bengali characters. Based on those characters, we store all the graphemes that make up these characters, which expanded our data set to slightly more than 3000 unique graphemes, up from the initial 1295. For each unique grapheme, we generated text images of size (128,128) using 3 different fonts: Kalpurush, NikoshLight, and BenSenHandwriting. In order to improve generalisation on these generated images, we applied gaussian blur to reduce the sharpness of the edges. These images are then concatenated with training set so that the model is able to expose itself to unseen graphemes and learn to recognize overall topology of each grapheme. Below is an example of how our generated graphemes look like compared to actual handwritten graphemes.

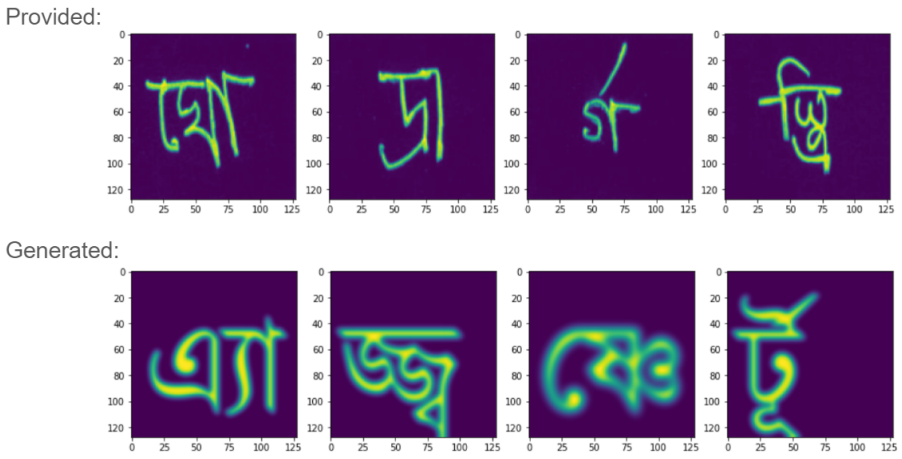


Figure 3: Handwritten and Generated Graphemes

2 Model Architecture

In this project, we explored the use of three main model architectures which are novel, implemented in PyTorch [Paszke et al., 2019]. Our baseline model by using a pretrained model Resnet18 [He et al., 2015] Inception net [Szegedy et al., 2015a].

Detailed setup and training instructions can be found on our GitHub repository.

2.1 Baseline Model

Overview

Our baseline models include Resnet18 [He et al., 2015], GoogleNet [Szegedy et al., 2015b], MobileNet v2 [Howard et al., 2017], VGG16 [Simonyan and Zisserman, 2014], and AlexNet [Krizhevsky et al., with their last fully connected layer slightly modified. The last fully connected layer has 186 neurons in total, which are splitted into three parts, 168 for grapheme root classification, 11 for vowel diacritics classification and the rest 7 for consonant diacritics classification.

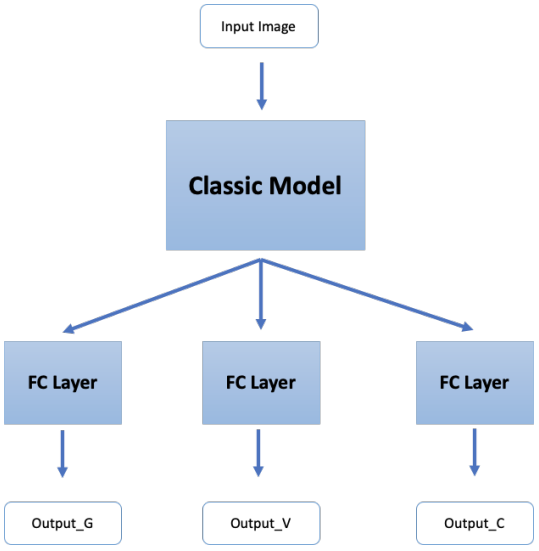


Figure 4: Architecture diagram for baseline models.

Experiment Details

Data Preprocessing and Augmentation Input images are resized to 128×128 pixels. During training, pixels of rectangles are randomly removed and shuffled in the training images.

Training

- Loss function: weighted sum of Cross Entropy loss for the three classification tasks, $0.5 \times graphemeLoss + 0.25 \times vowelLoss + 0.25 \times consonantLoss$.
- Optimizer: Adam Optimizer with 10^{-3} learning rate
- Scheduler: Reduce-learning-rate-on-plateau scheduler implementation in Pytorch, with patience of 3 and factor of 0.3.
- Batch size: 128.

For other not mentioned parameters, we left them as default values of Pytorch implementation.

Model	Seen Accuracy	Unseen Accuracy
ResNet18	0.9485	0.8391
GoogleNet	0.9442	0.8285
MobileNet_V2	0.934	0.812
VGG16	0.9327	0.8492
AlexNet	0.8284	0.7205
ResNet18 w/ Gen Data	0.9433	0.8247

Table 1: Baseline Performance

From the above table, the Seen Accuracy is based on the Validation Seen set, while the Unseen Accuracy is based on the Validation Unseen set. The best performing model is ResNet18 for seen accuracy and VGG16 for unseen accuracy. Resnet18 is a deeper model than VGG16, allowing it to learn the features of seen graphemes better, while VGG16's shallower model allows it to generalize better, and achieve a better score on unseen graphemes.
ResNet18 with Gen Data means that the generated data was concatenated with training data for this experiment. As we can see, it did not lead to an improvement in accuracy of either Seen or Unseen Graphemes. The generated style differs too much from handwritten images and the model is unable to learn any useful features from it.

2.2 Model 1 - Three Tailed Model

Overview

The three tailed model is an extension to the baseline model, where we implemented various methods to differentiate three constituents better. The core idea is to pass feature maps into three different classification blocks so that each block can be trained to specialise in one constituent of the three. The baseline model can be also viewed as a three tailed model with 1-layer tails.

Longer Tails

As mentioned above, the baseline model only splits the tails at the last layer. It may be too short to effectively learn how to separately classify the three constituents with only one layer of splited tails. Hence, we added more Dropout and Dense layers after the split. As another form of regularization all models in Model 1 uses random erasing augmentation with a probabilty of 0.5.

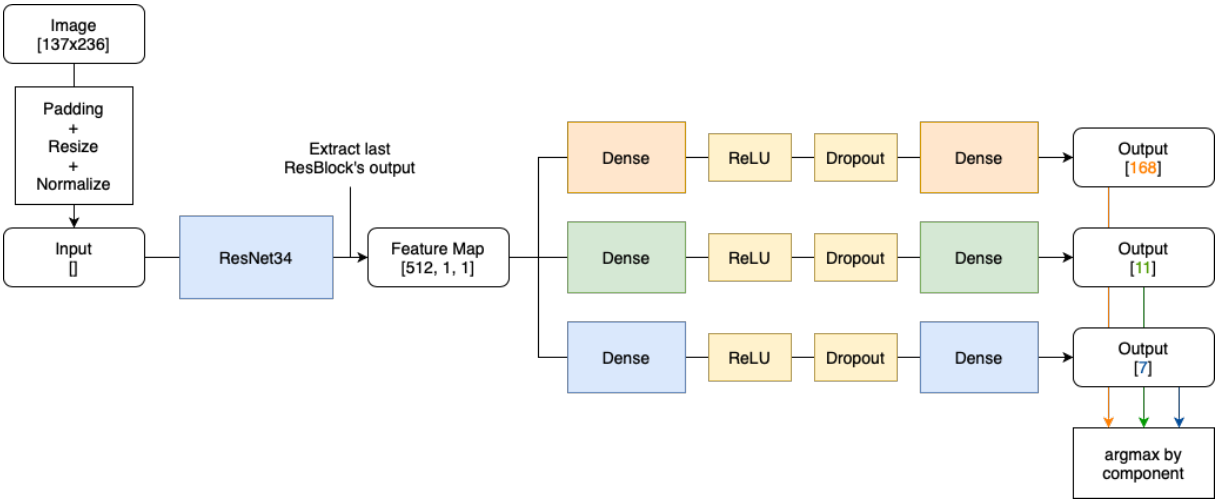


Figure 5: Longer Tails to Separately Classify the Constituents Better

Channel Squeeze and Spatial Excitation Pooling (sSE Blocks)

To separately classify the constituents better, it is important to separate not only the classification tails, but also the feature extraction part. Hence, before passing the CNN feature maps into classification tails, instead of doing global average pooling, we pass the last feature map ($2048 \times 4 \times 4$) to 3 sSE blocks.
An sSE block first takes an input feature map of shape $c \times h \times w$. It then uses a 2D convolution layer of kernel size 1×1 and output channel 1 to form a tensor with shape $h \times w$. It passes the tensor through a *sigmoid* function to get a weight tensor of shape $h \times w$, and multiplies the weight to the original feature map.
The 3 output feature maps are then passed into global average pooling layers which form 3 separate final features for each component. This allows the three tails to focus on different parts of the feature map.

SE-ResneXt Blocks

Furthermore, we replaced the normal ResNet structure in the encoder part with SE-ResneXt50 structure. SE-ResneXt50 is similar to the conventional Resnet50, but it modifies the architecture in two ways. Firstly, each block in Resnet is changed to something similar to an Inception module. In this case, there are 32 different paths instead of 4 in Inception. Secondly, there is a Squeeze and Excitation (SE) Pooling in each block (See Figure 5). SE Pooling and sSE Pooling are two variations of the same idea, one applying weights over channel dimension, the other applying weights over spatial dimensions. The difference between them can be observed in Figure 9.

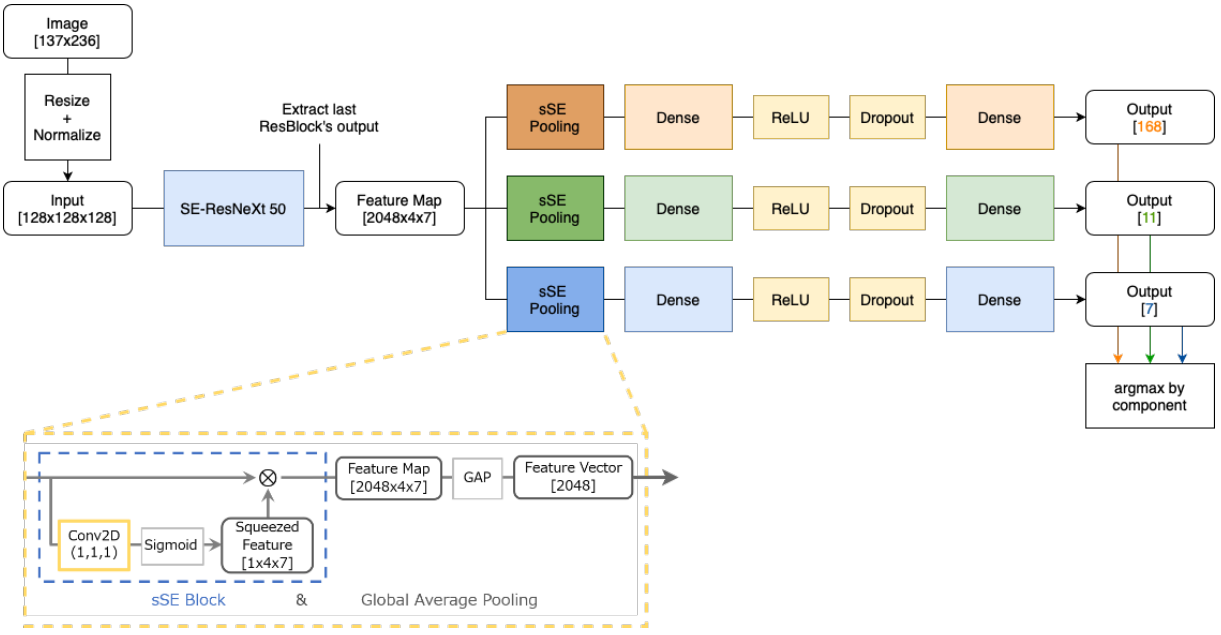
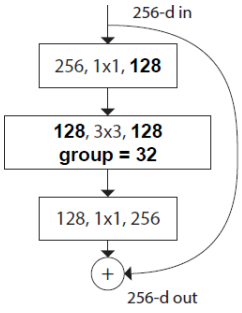
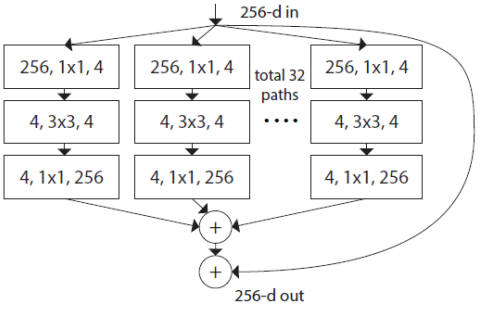


Figure 6: SE-ResNeXt50 with sSE Pooling and longer classification tails.



(a) Resnet block



(b) ResneXt block

Figure 7: Comparison between resnet and resneXt

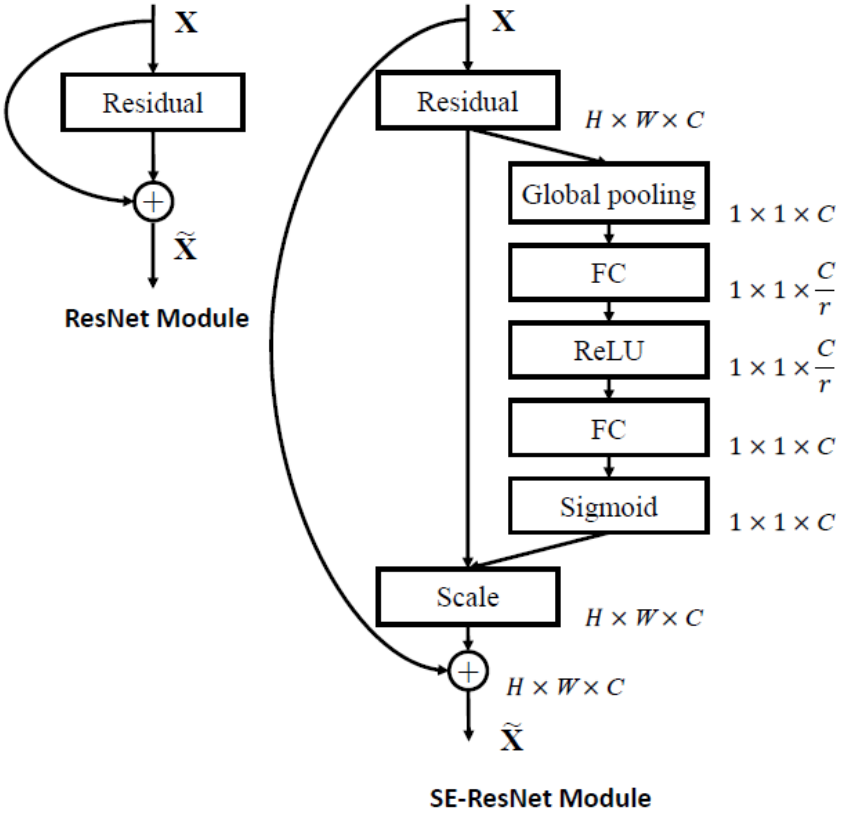


Figure 8: Comparison between resnet and SE-resneXt[Hu et al., 2017]

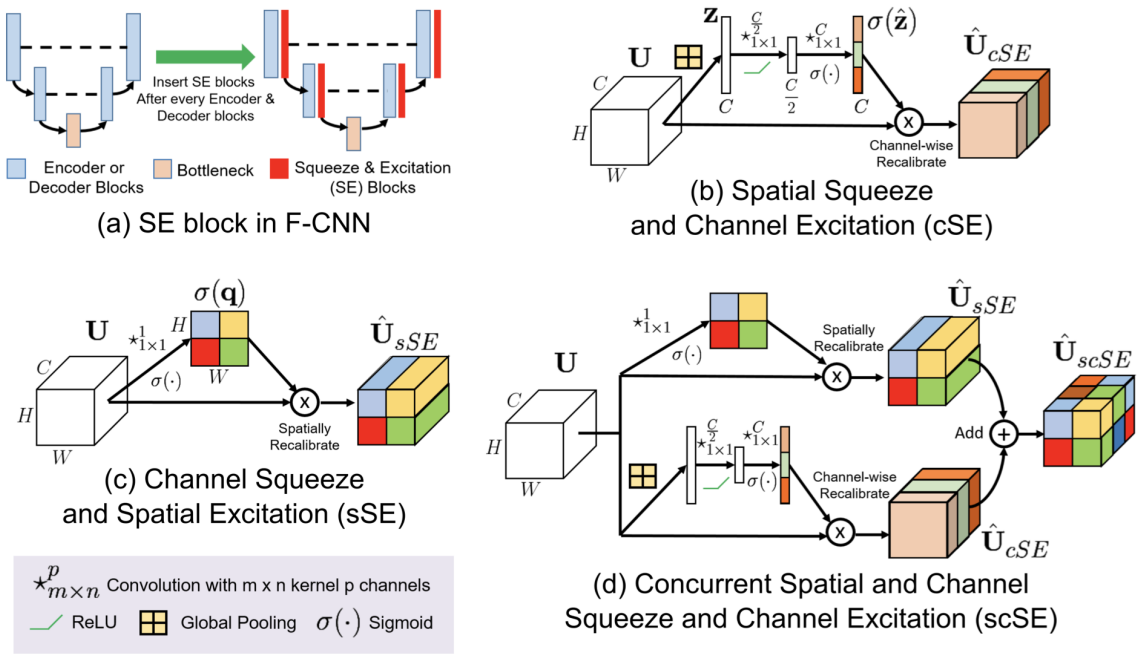


Figure 9: All kinds of variations of SE Blocks. (b) The SE block used in SE-ResneXt50; (c) The sSE block used before the 3 tails. [Roy et al., 2018]

Experiment Details

Data Preprocessing and Augmentation Input images are resized to 128×128 pixels. During training, pixels of rectangles are randomly removed in the training images.

Training

- Loss function: weighted sum of Cross Entropy loss for the three classification tasks, $0.5 \times graphemeLoss + 0.25 \times vowelLoss + 0.25 \times consonantLoss$.
- Optimizer: Nesterov SGD Pytorch implementation, with learning rate 1.5×10^{-2}, momentum 0.9 and weight decay 10^{-4}.
- Scheduler: Cosine Annealing Pytorch implementation with maximum number of iterations of 30.
- Batch size: 128.

For other not mentioned parameters, we left them as default values of Pytorch implementation.

Inference: Snapshot Ensemble To make the predictions more robust during inference, we combined the softmax output from the inference results from models saved at different training epochs (34, 69 and 104 epochs).

2.3 Model 2 - Seen and Unseen Model

Overview

Like 10, This model adapts a 2-stage prediction process using two models, one model is used to predict seen images and one model is used to predict unseen images. Seen model will output the result for the grapheme as well as its ArcFace Loss which can be used to classify whether the grapheme has been seen before. If the Arcface loss classify the grapheme as unseen before by the model, we will feed it to the unseen model and use its prediction as final prediction result.
The intuition behind this model is that due to a lot of the graphemes in the test dataset have not appeared in the training dataset, therefore we would like to have a model that can recognize graphemes as a whole if it has been appeared in the train dataset that is to say to overfit on the training dataset, and also at the same time be able to predict three constitutive component of the grapheme (root, consonant and vowel) independently if it has not seen the grapheme in the training dataset before. We followed the inspiration from: [1]

[1] kaggle: https://www.kaggle.com/c/bengaliai-cv19/discussion/135990

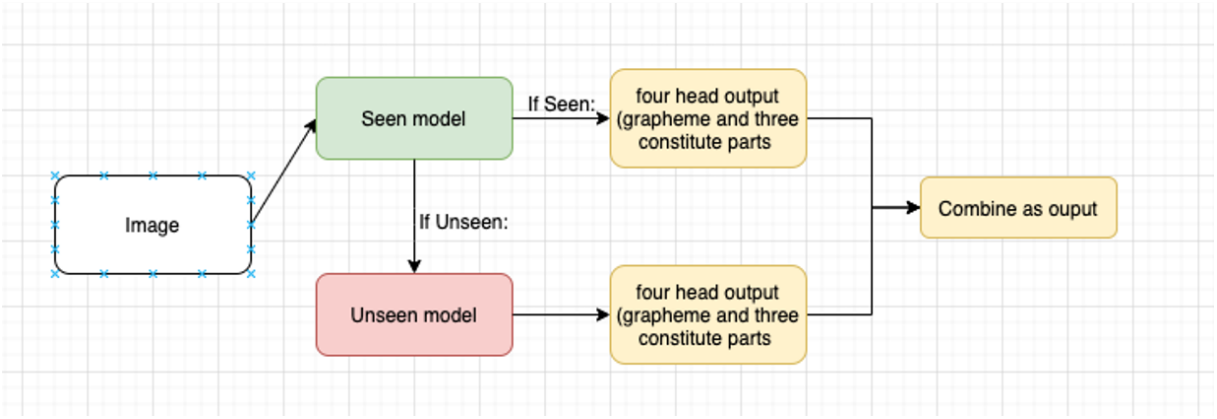


Figure 10: Seen and Unseen model pipeline

Seen Model

The Goal of the Seen Model is to make sure it can overfit to the graphemes in the training dataset.
Its output consists of Five parts:

- Grapheme id
- Grapheme Root id
- Consonant id
- Vowel id
- ArcFace loss

we used efficient-net [Tan and Le, 2019] [2]as our backbone for the model, and used Swish activation [Ramachandran et al., 2017] for more versatile activation after the last linear layer. Then we ensembled the results of dropout [Srivastava et al., 2014] by taking the average of the results each time after the dropout then adding another activation and dropout. We put more layers in the head than unseen models because this way will allow us to enable the model to balance between different kinds of features. and although grapheme id is not needed as a final result, since we have the label, we used it to co-train the model hoping that it can train better feature extractor. the detailed architecture can be refered to in 11.

[2]Pytorch implementation for efficientnet: https://github.com/lukemelas/EfficientNet-PyTorch

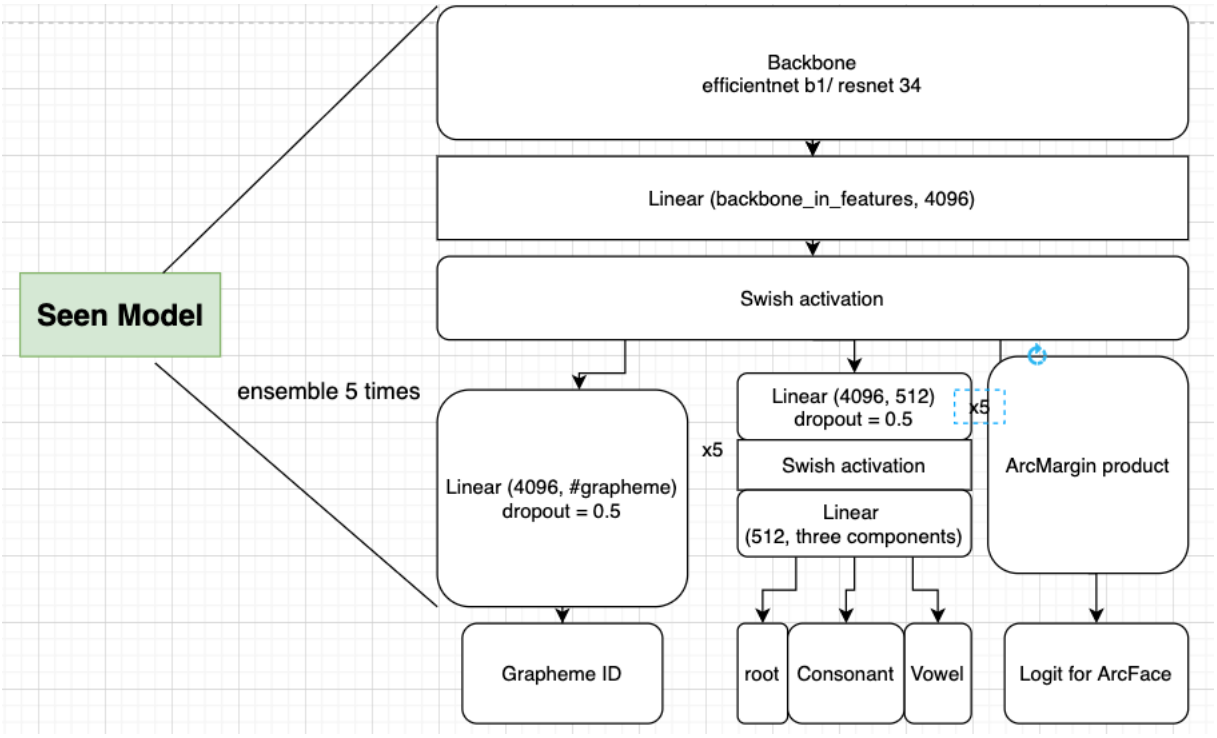


Figure 11: Seen model architecture

Unseen Model

The Goal of the UnSeen Model is to let it independently predict the three parts of the grapheme without associates them with the grapheme.
Its output consists of Four parts:

- Grapheme id
- Grapheme Root id
- Consonant id
- Vowel id

The whole of the Unseen model is quite similar to Seen Model mentioned above, except the following differences:

1. In order to predict separately for three components of the grapheme, we used cutmix with a certain probability [Yun et al., 2019] to regularize the model to train stronger classifiers, whereas in Seen model there is no cutmix used. The advantage of cutmix over the grid dropout is that it have a more efficient use of training pixels.
2. It have less linear layer as head after the efficientnet, this can prevent the model does not learn too complex interaction between the features output by the body.
3. grapheme is output as a combination of its three components rather than previously in Seen model we consider a complex interaction by the output features of the 'body' efficientnet in our case. The intuition behind this is that we would like to put our focus on predicting the correct component of the grapheme rather than grapheme itself, therefore unlike the seen model, we don't model grapheme separately using its own MultiLayer perceptrons.

the detailed architecture can be refered to in 12.

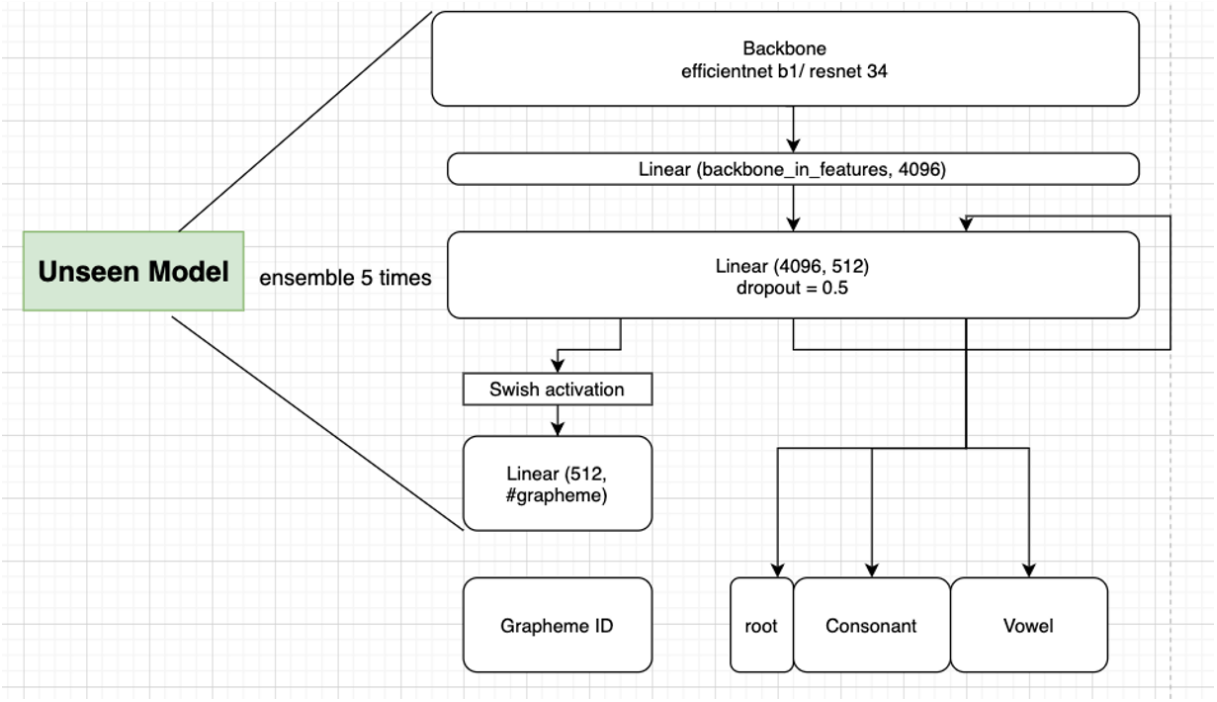


Figure 12: Unseen Model architecture

Loss function

Due to drawbacks of softmax Loss, it does not explicitly optimize the feature embedding to enforce higher similarity for intra-class samples and diversity for interclass samples. Therefore we used ArcFace Loss [Deng et al., 2019] to impose an angular margin between each classes. By doing this, not only we can achieve a compactness of intra-class representations and more discrepancy of inter-class representations. Unseen images tends to give a higher loss to each of it compact clusters. Therefore, by using a threshold to filter out the images of high ArcFace loss, we can have the models that is seen before.

Experimental Details

We resized our images to 137x128 for all the images just like other model, and we used a cutmix of 0.85 of the original image size. We trained with a batch size of 128, on Nvidia GTX 2080 Ti for 60 epochs, around 5 hours. During training we used Cosine annealing to warm up the training [Loshchilov and Hutter, 2016] for better convergence, the initial training rate is 0.001 and we decrease it to 1/10 of the original learning rate while training the model.
The loss for both the seen model and the unseen model has a weight of 4 for the grapheme, 2 for the grapheme root since root has more weightage during evaluation 3.1 for other two component - consonant and vowel we assigned a weightage of 1, this is the loss using softmax loss, and we assigned a weightage of 2 for the arcFace loss for more compact loss for the seen model, and used a threshold of 0.825 to determine whether the model is seen or unseen. We used 5-fold evaluation to get our best results.

3 Evaluation

3.1 Model Performance

We followed the evaluation metric proposed by Kaggle[12]:
Averaged Macro recall of "root", "consonant" and "vowel", the score is combined with a weight of 2:1:1. (this is also reflected in our loss function)

Model	Private Score	Public Score
ResNet18 Baseline	0.8926	0.9529
ResNet34LongerTail 34 epoch	0.9043	0.9605
ResNet34LongerTail 69 epoch	0.9027	0.9636
ResNet34LongerTail 104 epoch	0.9023	0.9641
ResNet34LongerTail SnapshotEnsemble	0.9084	0.9661
se-resnextLongerTail 30 epoch	0.9176	0.9743
se-resnextLongerTail SnapshotEnsemble	0.9247	0.9737
Seen and Unseen Model	0.9350	0.9814

Table 2: Kaggle Test Set Results

Table 2 are results from the test sets on Kaggle's servers. The private test set contains unique combinations of graphemes that do not exist in the provided training set. ResNet18 has performed the best amongst all our baseline models for seen graphemes, and as such, we've put its performance here to compare with other improved models. From the table above, we see the effectiveness of Snapshot Ensemble, that its overall performance is better than the network at each epoch. Motivated by this performance, we applied the same Snapshot Ensemble technique onto the ResNet with sSE blocks, which gave us even better performance. This demonstrates that the feature maps of the sSE blocks plays a significant role, allowing the model to parameterize itself according to grapheme roots, vowel diacritics, and consonant diacritics.
Unsurprisingly, the Seen and Unseen Model performed the best on both Public and Private score. The importance of having an unseen model is emphasized as current models have a tendency to learn from the whole grapheme rather than its components. By isolating a model for unseen graphemes, we are able to use heavier data augmentation such as cutmix to force the model to focus on each individual component of the graphemes. The separate seen model is then free to overfit to seen graphemes by extracting features from the whole grapheme. Putting them in tandem allows accurate prediction on both seen and unseen graphemes.

4 Conclusion

Handwritten Bengali graphemes are indeed very challenging for optical character recognition, and we have yet to explore many more techniques that will result in a more reliable recognition algorithm. Nevertheless, there are many things we can learn from the experiments we have run. Simple generation of Bengali graphemes does not help the model learn more meaningful features as the style is too different. Future improvements on this include implementing a CycleGAN to generate images that resemble handwritten graphemes so that the model can learn more useful features when predicting on handwritten images. Additionally, when splitting the tails to predict on each component, instead of using non-parameterizable modules such as global averaging, using parameterizable blocks such as sSE blocks enables the model to learn the nuances for the components individually. Lastly, current models have a tendency to overfit to

seen grapheme combinations, resulting in less than optimal performance for graphemes that do not exist in the training data set. Thus, it is important for a model to learn on individual grapheme components, which can be done through heavy augmentations such as cutmix.

Bibliography

[Deng et al., 2019] Deng, J., Guo, J., Xue, N., and Zafeiriou, S. (2019). Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4690–4699.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

[Howard et al., 2017] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications.

[Hu et al., 2017] Hu, J., Shen, L., Albanie, S., Sun, G., and Wu, E. (2017). Squeeze-and-excitation networks.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[Loshchilov and Hutter, 2016] Loshchilov, I. and Hutter, F. (2016). Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

[Ramachandran et al., 2017] Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

[Roy et al., 2018] Roy, A. G., Navab, N., and Wachinger, C. (2018). Concurrent spatial and channel squeeze excitation in fully convolutional networks.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

[Szegedy et al., 2015a] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015a). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[Szegedy et al., 2015b] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015b). Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*.

[Tan and Le, 2019] Tan, M. and Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*.

[Yun et al., 2019] Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., and Yoo, Y. (2019). Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6023–6032.