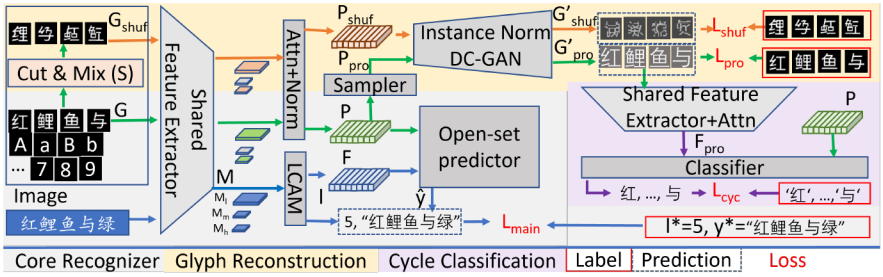


Fig. 7. The OpenSAVR framework.

Appendix A OpenSAVR: An Engineering Perspective

This appendix conveys an engineering perspective of the proposed OpenSAVR method shown in Fig. 7. The first part introduces the feature extractor, and the second part describes the LCAM module and the Attn module in question. [6]

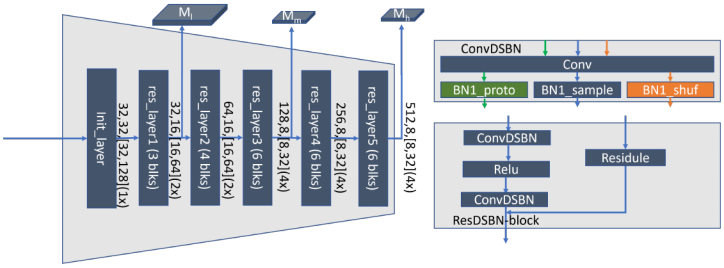


Fig. 8. The structure of the shared feature extractor.

A.1 Shared Feature Extractor

The shared feature extractor (shown in Fig. 8) is a Resnet45 [45] with all BatchNorm layers replaced with the Domain Specific Batch Norm [6] reimplemented by [27]. Structural-wise, the feature extractor contains one init layer and five Resnet layers. The network takes one 3-channeled image as its input, and produces 3 levels of feature maps as outputs, namely M_l, M_m, and M_h. For the large model, all channel numbers except those from the last layer of the feature extractor are multiplied by 1.5. Thus, except for the feature extractor itself

[6] The codes are mostly included in neko_sdk/encoders/chunked_resnet/bogo_nets.py and neko_sdk/chunked_resnet/res45.py

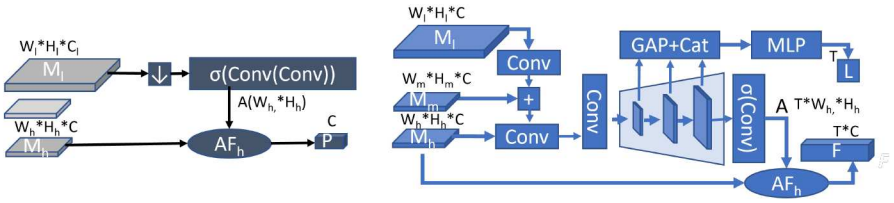


Fig. 9. The Attn module (left) and the L-CAM module (right)

and the attention modules (`Attn` and `LCAM`), the rest of the framework remains identical for the large and regular models.

The feature extractor is used to extract features for all three domains, namely the prototype domain (in green), the shuffled-prototype-domain (in orange), and the sample domain (in blue). To tackle domain biases and reduce their impacts, we introduce DSBN following [27]. The behavior of a DSBN layer is shown in Fig 8.

A.2 LCAM and Attn

The framework involves two attention modules adopted from [27], namely `Attn` and `LCAM`.

The `Attn` (shown in Fig. 9) is a module that aggregates character features extracted by the shared feature extractor to a feature vector. Our implementation involves two steps: First, the low-level feature map $M_l \in \mathbf{R}^{16\times16\times32}$ [7] is down-scaled to 8×8, followed by 2 convolutional layers which predict it into the foreground score map A. Second, the character feature vector is produced by a weighted average pooling over M_h with A as weight:

$$P = \frac{\sum_{i=1,j=1}^{8,8} A_{[i,j]} M_{h\,[i,j]}}{\sum_{i=1,j=1}^{8,8} A_{[i,j]} + eps}. \tag{16}$$

Here, eps is a small number to avoid divided-by-zero issues.

The L-CAM module (Shown in Fig. 9) is used to sample the time-stamp-aligned character feature from the input word image. The module first fuse feature maps from different levels, then upsample them and generate attention masks [45]. The feature maps are also used to generate length predictions [27] to break free from the end-of-speech tokens used in most attention-based methods [3, 26, 45].

A.3 Openset Predictor

In OpenSAVR, the prototypes are compared with the timestamp-aligned sample feature with an open-set predictor [27] illustrated in Fig. 10. Since some

[7] Note we are talking about side-information, which are $32 \times 32 \times 3$single character images extracted from font

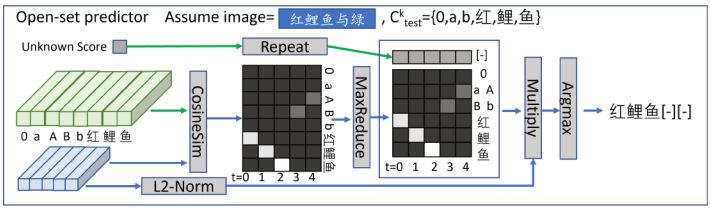
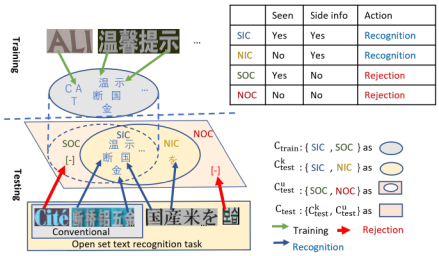


Fig. 10. A toy example of the open-set predictor.

labels may have more than one prototype, the prototype-wise score needs to be max-reduced into class-similarity. A learnable threshold is concatenated to the score to implement rejection. Specifically, sample features yielding smaller similarity scores for all prototypes would be classified as unknown. During training, the model leaves out some prototypes to create unseen labels to train the threshold [26].



	Seen	Side info	Action
SIC	Yes	Yes	Recognition
NIC	No	Yes	Recognition
SOC	Yes	No	Rejection
NOC	No	No	Rejection

	Training Label	Training Action	Evaluation Label	Evaluation Action
KKC	Specific label	Recognition	Specific Label	Recognition
SIC	Specific Label With side-information	Recognition	Specific Label With side-information	Recognition
KUC	Unknown label	Rejection	Unknown label	Rejection
SOC	Specific Label With side-information	Recognition	Side-information-absence	Rejection
UKC	Not available (inductive) Unpaired (transductive)	---	Specific Label With side-information	Recognition
NIC	Not available(inductive) Unpaired (transductive)	---	Specific Label With side-information	Recognition
UUC	Not available(inductive) Unpaired (transductive)	---	Side-information-absence	Rejection
NOC	Not available* Unpaired (transductive)	---	Side-information-absence	Rejection

Fig. 11. The illustration of the OSTR task [26] and a comparison to conventional OSR tasks [14].

Appendix B More on Open-set Text Recognition

This section first introduces the open-set text recognition task formulation, then adopts a showcase to illustrate how the task can be applied, and explains some non-intuitive settings of the OSTR task.

B.1 Openset Text Recognition Task

The OSOCR task is proposed by Liu et. al. [26] as illustrated in Fig 11, which aims to automatically spot novel characters from the data and incrementally adapt to them. To implement the goal, the task first splits all characters into four categories, namely SIC, SOC, NIC, and NOC. Specifically, the first alphabet describes whether or not the characters are included in the training samples, and the second describes whether corresponding side information is provided for the characters during testing. Conceptually, the four categories are correspondingly comparable to the KKC, KUC, UKC, and UUC concepts in the open-set recognition [14] tasks. However, there are substantial differences as highlighted in Fig 11 in purple color, hence different names are used to prevent confusion.

In OSTR, SICs and NICs form the "in-set" characters $\mathbf{C}^k_{test}$, whereas SOCs and NOCs form the "out-of-set" characters $\mathbf{C}^u_{test}$. The model needs to recognize samples composed of characters in $\mathbf{C}^k_{test}$, while rejecting the samples containing $\mathbf{C}^u_{test}$. In the OSTR task, "unknown label" is associated only with the presence of side information, rather than specific labels, nor is the label bonded to label novelty. This allows the user to exchange characters between "in-set" and "out-of-set" sets by providing or removing the corresponding side information according to data or real-life requirements.

B.2 Open-set Text Recognition: A Use-case Perspective

This part describes how OpenSAVR is designed to handle open environments, with a model trained as illustrated in Fig. 7.

Before putting it into production, one is advised to cache the prototypes for currently in-set characters ($\mathbf{C}^k_{test}$) beforehand, as illustrated in step 0. in Fig. 12.

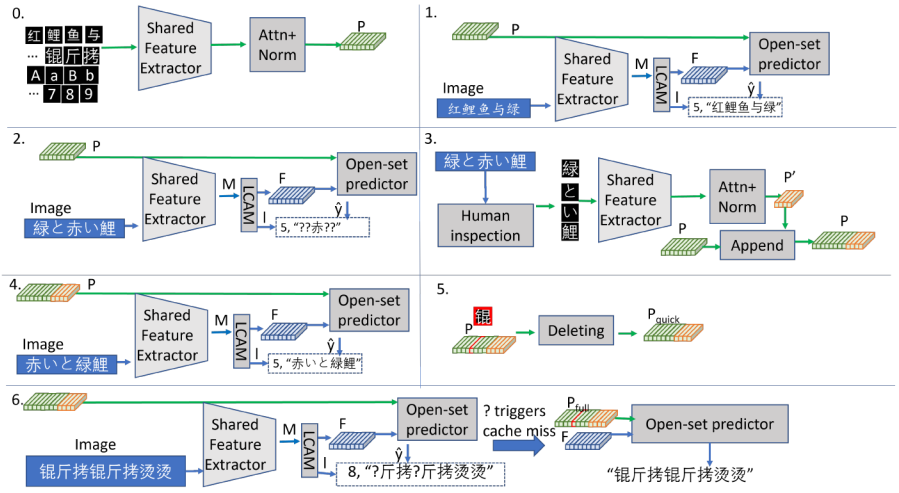


Fig. 12. Steps and typical actions of OpenSAVR to handle open environments.

Then the model would be ready to recognize text utilizing the cached prototypes and the core recognizer, as shown in step 1.

When a sample with an unknown label [8] occurs like in Step. 2, the model would produce "reject" labels for those novel characters that fail to match any characters in $\mathbf{C}^k_{test}$.

Such samples will raise alerts to the system admins, who will manually go through these samples as illustrated in Step 3. The admin needs to pick out the unseen characters, and decide whether they should be added for further recognition. Once the admin decides to do so, the person simply grabs the corresponding glyphs, maps them to prototypes with the feature extractor and `Attn` module, appends the generated prototypes to the cached prototypes, and the model is ready to recognize novel characters as shown in Step 4.

However, the admin may find some characters rarely occur in the data, and decide to remove or swap them out to speed up the inference process, as shown in Step. 5. This is also the reason for the OSTR task to include training characters as part of $\mathbf{C}^u_{test}$(excluded from $\mathbf{C}^k_{test}$ hence subject to rejection) in the full OSTR split. Finally, if the characters are swapped-out instead of removed, rejection will trigger a cache miss first, which invokes the model to rematch the swapped-out prototypes for such classes. The admins will be notified if and only if the rematch fails.

[8] in $\mathbf{C}^u_{test}$, could mostly be novel, or could be also explicitly swapped out characters. Note $\mathbf{C}^u_{test}$ describes anything NOT in $\mathbf{C}^k_{test}$, hence is possibly unknown to the user. $\mathbf{C}^k_{test}$ models characters with side information provided.

Table 4. The impact of L_{cyc} and L_{pro} over different type of characters.

Data	IIIT5k	CUTE	SVT	IC03	IC13	Shared Kanji	Unique Kanji	Kana
Seen	✓	✓	✓	✓	✓	✓	-	-
Base model	91.10	80.55	83.15	91.46	90.24	64.53	54.44	9.48
Ours	90.67	73.26	82.53	87.54	89.26	69.70	63.25	9.60
Ours-Large	92.33	84.38	85.16	91.58	90.64	72.91	64.37	10.72

Appendix C Discussion on Close-Set Performance Impacts

We witnessed a significant improvement after introducing L_{rec} and L_{cyc}. Here, we study how the losses potentially affect the performances of different types of characters.

The proposed losses may yield mild negative effects on seen text, potentially by making the model less context reliance. Specifically, on the seen part of the OSTR task(subset with samples containing only seen Chinese characters), the base model demonstrates a 64.53 line accuracy, lower than the full model (69.70). On the standard close-set-benchmark, the trained base-model yields 91.10 IIIT5k performance, pretty close to the full model (90.67). Most other datasets also show close performances, however, there is a gap in CUTE, partially caused by the less-context-reliance property, (CUTE has its dictionary completely covered by the training data.) Another reason would be the model experience difficulties modeling detailed visual shapes and drastic pose variations at the same time due to its lightweight nature, as Ours-large is much less affected.

The losses mostly yield improvements over novel characters with close shape to seen characters, specifically the Unique Kanjis that are not included in the training data. For characters that have strong shape differences like Kanas, the regularization terms are less effective, indicating the cut-and-mix approach may not be sufficient to cover the entire character space.