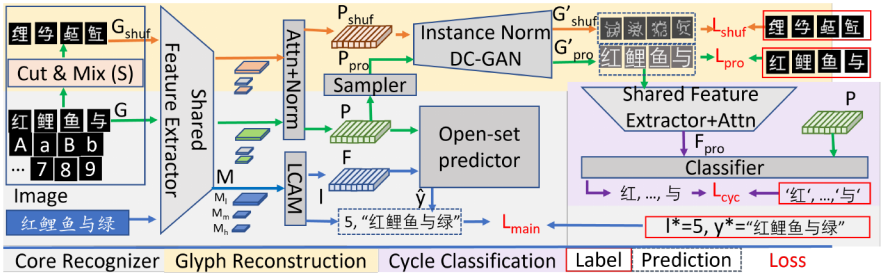

Fig. 7. The OpenSAVR framework.

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. [3]

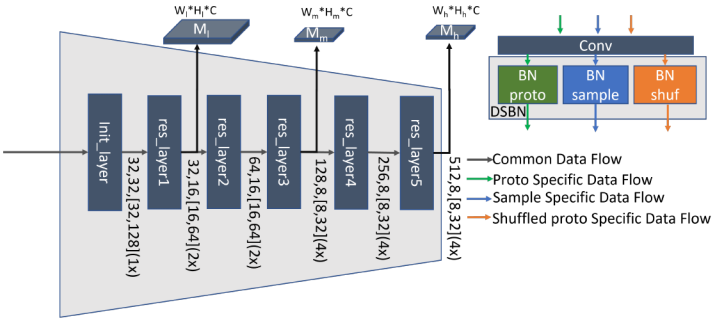

Fig. 8. The structure of the shared feature extractor.

A.1 Shared Feature Extractor

The shared feature extractor (shown in Fig. 8) is a Resnet45 [40] with all BatchNorm layers replaced with the Domain Specific Batch Norm [6] reimplemented by [25]. 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

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

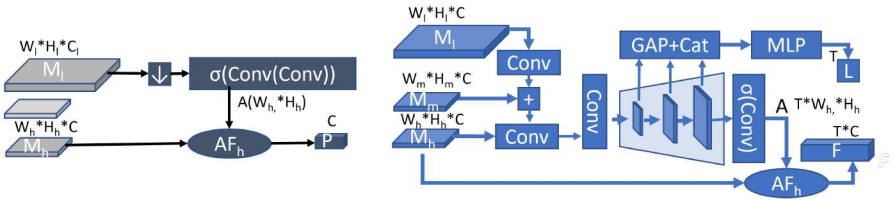

Fig. 9. The Attn module (left) and the L-CAM module (right)

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 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 [25]. The behavior of a DSBN layer is shown in Fig 8.

A.2 LCAM and Attn

The framework involves two attention modules adopted from [25], 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 $M_l \in \mathbf{R}^{16\times16\times32}$ [4] 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:

$$P = \frac{\sum_{i=1,j=1}^{8,8} A_{[i,j]} M_{h\,[i,j]}}{\sum_{i=1,j=1}^{8,8} A_{[i,i]} + eps}. \tag{14}$$

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 [40]. The feature maps are also used to generate length predictions [25] to break free from the end-of-speech tokens used in most attention-based methods [24,3,40].

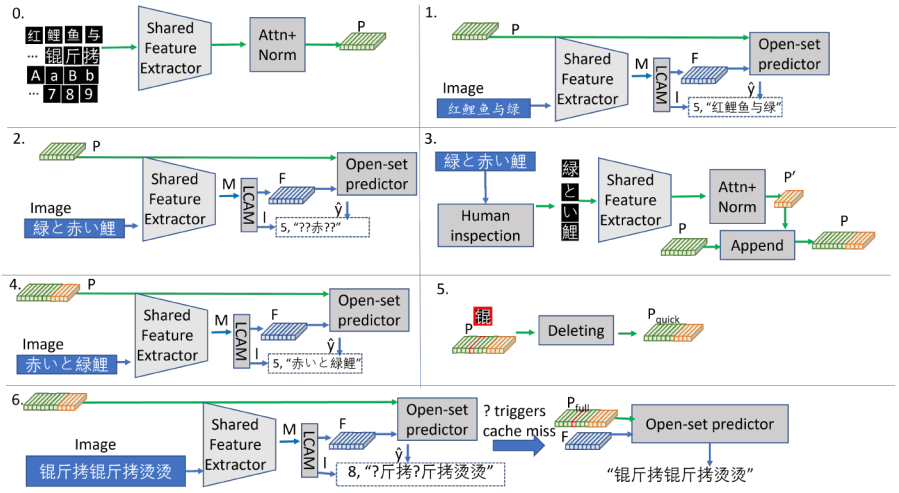

Fig. 10. Steps and typical actions of OpenSAVR to handle open environments.

A.3 Recognizing Novel Characters in Action

In simple words, the model learns to compare samples and glyphs, thus it adapts to novel characters once corresponding template(s) is/are present, and rejects characters with no present templates. 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 known characters[5] beforehand, as illustrated in step 0. in Fig. 10.

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 new label [6] 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.

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

[5] Which we denote as the current $\mathbf{C}^k_{test}$.

[6] such characters are denoted as $\mathbf{C}^u_{test}$, which describes anything in $\mathbf{C}^k_{test}$, which is also possibly unknown to the user

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 us 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.