TOWARD A MORE COMPLETE OMR SOLUTION

Guang Yang[1] **Muru Zhang**[1] **Lin Qiu**[1] **Yanming Wan**[1] **Noah A. Smith**[1,2]
[1]Paul G. Allen School of Computer Science & Engineering, University of Washington, United States
[2]Allen Institute for Artificial Intelligence, United States
{gyang1,nasmith}@cs.washington.edu

ABSTRACT

Optical music recognition (OMR) aims to convert music notation into digital formats. One approach to tackle OMR is through a multi-stage pipeline, where the system first detects visual music notation elements in the image (object detection) and then assembles them into a music notation (notation assembly). Most previous work on notation assembly unrealistically assumes perfect object detection. In this study, we focus on the MUSCIMA++ v2.0 dataset, which represents musical notation as a graph with pairwise relationships among detected music objects, and we consider both stages together. First, we introduce a music object detector based on YOLOv8, which improves detection performance. Second, we introduce a supervised training pipeline that completes the notation assembly stage based on detection output. We find that this model is able to outperform existing models trained on perfect detection output, showing the benefit of considering the detection and assembly stages in a more holistic way. These findings, together with our novel evaluation metric, are important steps toward a more complete OMR solution.

1. INTRODUCTION

Optical music recognition (OMR) focuses on converting music notation into digital formats amenable to playback and editing. OMR systems are generally divided into two categories: end-to-end systems (which directly convert the image into music notation) and multi-stage systems. Proposed and refined by [1–3], a standard multi-stage system consists of four stages: preprocessing, music object detection, notation assembly, and encoding. In this study, we focus on the object detection and notation assembly stages.

MUSCIMA++ [4] suggests representing music notation as a graph where each pair of musical symbols is linked by a binary relationship, allowing for clear notation reconstruction. The authors created a dataset of handwritten scores with a bounding box for each music object and a human-annotated graph of object relationships in each image. Notation assembly on MUSCIMA++ can be framed as a set of binary classification decisions to predict the pairwise relationships between music symbols. Most prior research has explored notation assembly with the assumption of perfect detection output [5], but such assumptions can introduce unwanted biases that deteriorate the performance of the notation assembly system when applied as part of a pipeline. Pacha et al. [6] evaluate a notation assembler on realistic detector output, finding some degradation relative to gold-standard objects, but they do not seek to mitigate the problem.

To improve notation assembly robustness, we propose a training method to complete notation assembly on top of (imperfect) object detection output directly. To have a strong detector to start with, we train YOLOv8 [7] and perform a set of preprocessing steps to adapt the model to the MUSCIMA++ v2.0 dataset. Our detector outperforms previous detectors on MUSCIMA++ v2.0 [8] by 2.4%, establishing a solid foundation for notation assembly.

Traditional evaluation methods, which perform notation assembly over all pairs of ground-truth objects and report an F1 score or a precision-recall curve, become inadequate when the input objects come from imperfect detection. We propose an end-to-end evaluation metric, called Match+AUC, that accounts for both detection errors and assembly errors by first matching detected objects with their ground-truth counterparts before assessing notation assembly accuracy. It complements metrics that evaluate pipeline components individually.

Our code for reproducing all of the experiments is publicly available at `https://github.com/guang-yng/completeOMR`.

2. MULTI-STAGE OMR

We focus on the MUSCIMA++ v2.0 dataset [4] and follow its multi-stage pipeline for the OMR system. This dataset includes 140 high-resolution annotated images out of 1000 images from the CVC-MUSCIMA dataset [9]. It contains 91,254 symbol-level annotations and 82,247 relationship annotations between symbol pairs by human annotators. These annotations span 163 distinct classes of music symbols. Figure 2 shows an example from this dataset.

As the MUSCIMA++ dataset provides symbol-level pairwise relationships, it allows study of two stages of the pipeline: (i) detection and (ii) assembly. In (i), given an image as input, an object detector is used to extract all music symbols in the image, denoted as the set $V = \{v_i\}_i$,

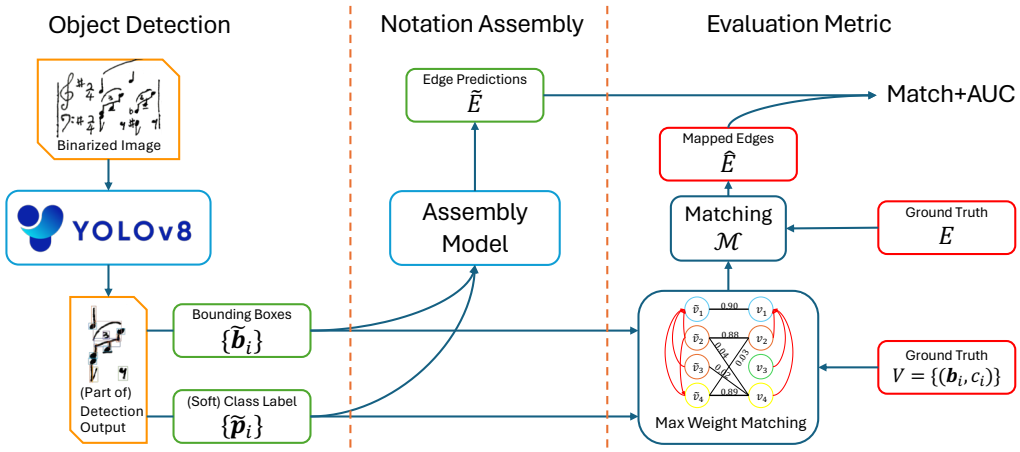

Figure 1. An overview of our OMR pipeline, highlighting key components: object detection, notation assembly, and evaluation metric. Detailed explanations of each component can be found in Subsections 3.1, 3.2, and 3.3 respectively.

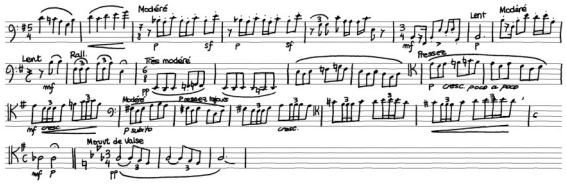

Figure 2. Example of a music image (binarized) extracted from the MUSCIMA++ dataset.

where $v_i = (\mathbf{b}_i, c_i)$ is a tuple of a bounding box and a class label. Each pair of music symbols (v_i, v_j) is then fed into (ii) the notation assembly model to predict whether or not there exists a relationship between them. The notation assembly stage can be framed as an edge prediction problem where the model needs to output a set of edges E to get a directed graph $G = (V, E)$. MUSCIMA++ defines a grammar over all possible music symbol classes so that the direction of an edge is uniquely determined by the class labels (c_i, c_j) of the vertices (v_i, v_j). Consequently, the edge prediction problem can be reduced to predicting an undirected graph. The authors of [4] argue that such a graph G enables straightforward reconstruction of the full symbolic music notation, so we do not consider the decoding process after (i) and (ii) in this work.

In previous works, the two stages are considered separately, either focusing on object detection, without fully analyzing its effect on downstream notation assembly [8, 10, 11]; or focusing on notation assembly and assuming perfect detection input during training [5, 6]. This raises the question of whether the best object detector is a good fit for the best notation assembly model. To investigate, we developed an end-to-end metric that evaluates the performance of the entire pipeline, as explained in Section 3.3. We found that, compared with our approach where both stages are considered together—specifically, where the notation assembly model is trained using the output of the object detector—treating the two stages separately leads to poorer results.

3. METHODOLOGY

We describe our method for each stage, and how we connect the two stages together and evaluate the entire pipeline. Figure 1 shows an overview of our methods.

3.1 Music Symbol Detection

A music object detection system analyzes an image to identify each music object it contains, providing both the bounding box and class label for every detected object [10]. Traditionally, this process would begin with an initial stage of image preprocessing, typically aimed at removing staff lines, followed by a second stage focusing on the segmentation and classification of symbols. Thanks to recent advances in computer vision, there are mature solutions for image preprocessing and staff line removal, allowing us to treat it as a largely solved problem [12–14]. In our case, MUSCIMA++ provides us with staff line removed images as input, so we directly build our detectors on top of these images.

Following the work of Zhang et al. [8], we adopted a convolutional neural network-based approach for page-level object detection of handwritten music notes, opting for this approach over segmentation-based methods, because segmentation-based methods often struggle with overlapping symbols. We choose YOLOv8 [7], which is the latest version of YOLO [15], due to its superior performance on traditional computer vision tasks. Compared to YOLOv4 [16], which is used by [8], YOLOv8 has a new loss function and a new anchor-free detection head, achieving higher performance on various detection tasks. YOLOv8 has not yet, to our knowledge, been applied to OMR. Furthermore, since the images of handwritten music notation in MUSCIMA++ have high resolution and music objects are drastically different from the objects considered in computer vision research, directly applying the training strategy of YOLOv8 doesn't work well. We follow [2, 8, 17] to crop images into small snippets during the training stage to alleviate this issue. Specifically, we randomly crop the images during training and compactly segment the image during inference. More details are pre-

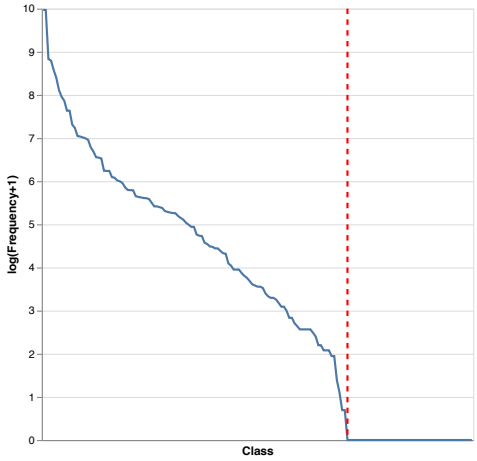

Figure 3. Frequencies of different classes in the dataset, from most- to least-frequent. A long-tailed distribution with 48 classes on the right of the red line that never appear. The y-axis shows the value of $\ln(\text{frequency} + 1)$. The top-5 classes are `stem`, `noteheadFull`, `ledgerLine`, `beam`, and `staffSpace`.

sented in Section 4.1.2.

The MUSCIMA++ v2.0 dataset includes 163 object classes in total, covering a large variety of notation. However, most of the classes scarcely appear and barely affect the replayability of the OMR output (e.g., the construction of a MIDI file encoding the score). The distribution of classes is shown in Figure 3; 48 classes never appear in the entire dataset. Given this, we manually remove these 48 classes along with some other rare classes, leading to a subset of 73 attested "essential" classes that are observed in the dataset. To get a direct comparison with previous methods, while also keeping a focus on essential classes, we report results using both the full class set and essential classes only. Meanwhile, we also report results on the 20 "primitive" classes selected for evaluation by [8].

3.2 Notation Assembly

The notation assembly model takes a pair of nodes as input, and gives a binary output indicating whether there is a relationship between them. An intuitive method is to first concatenate the features of two nodes, and then pass the pair as a single feature vector through a series of layers of a multi-layer perceptron (MLP). A sigmoid function σ is applied at the end to output the probability that there exists a relationship.

$$\hat{e}_{ij} = \sigma(\phi_{\text{MLP}}([v_i, v_j])) \tag{1}$$

As notation assembly is essentially binary classification, we use binary cross-entropy as our loss function:

$$\mathcal{L}_{\text{BCE}}(\hat{e}_{ij}) = -e_{ij}\log(\hat{e}_{ij}) - (1 - e_{ij})\log(1 - \hat{e}_{ij}).$$

We adopt the input feature design in [5], where each v_i is represented by its 4-dimensional bounding box and the class label. The class label is passed to an embedding layer with x dimensions. Therefore, the input to MLP will be a $(4 + x) \times 2$ dimensional vector.

Existing work assumes perfect detection output; therefore, the input bounding box and class label are the ground-truth information. While previous work has attempted to manually perturb the bounding box as a test of robustness, such perturbations don't reflect the kind of errors that might arise in a practical object detector.

To ensure our notation assembly system can adapt to errors introduced in the detection stage, we propose a supervised training pipeline that directly trains the assembly model on detection output $\tilde{V}$. Since most of the time $\tilde{V} \neq V$, we can't directly use the ground truth E as the supervision signal.

To deal with this issue, we construct a maximum weight matching M in the bipartite graph $G_M = (\tilde{V}, V)$ and build $\hat{E}$ for supervising our notation assembly model. We describe the detail of our matching procedure in Section 3.3, where it is also employed in evaluation. We adopt the edges from the ground truth according to our matching. Given a pair $(\tilde{v}_i, v_k) \in M$ and an edge $(v_k, v_h) \in E$, we add $(\tilde{v}_i, \tilde{v}_j)$ to $\hat{E}$ if $(\tilde{v}_j, v_h) \in M$. Our method essentially builds a training set for the detection output that is in the same format as the ground-truth, allowing seamless training and evaluation.

3.3 End-to-End Evaluation

The main challenge of OMR evaluation is finding the edit distance between two music scores under some particular representation (e.g., XML format [18]). Hajič [19] argued that intrinsic evaluation is needed to decouple research of OMR methods from individual downstream use-cases, since specific notation formats change much faster than music notation itself. Some works have taken steps to analyze the complexity of standard music notation [20] and propose common music representation formats [21].

As a general system consisting several modules, we seek to also evaluate our OMR pipeline holistically, without a specific focus on what the downstream processing will be. We therefore propose a novel matching-based evaluation metric to assess predictions that include errors from the detection stage. For the same reason we had to adapt ground-truth edges to create training data for the notation assembly model ($\tilde{V} \neq V$), we cannot straightforwardly use the ground-truth graph to evaluate notation assembly. Our metric finds a matching between a test instance's predicted objects and those in the ground-truth object detection, and then uses this as a bridge to evaluate the edges returned by the notation assembly module.

The results reported by Pacha et al. [6] are the sole benchmark for assessing a notation assembly model using detected symbols. To address the matching issue between $\tilde{V}$ and V, Pacha et al. employ a rule-based method, considering two objects identical if they belong to the same class and their intersection over union is at least 50%. However, this greedy matching approach is inadequate, as inaccuracies in symbol detection cannot be compensated for by the notation assembly model. Furthermore, Pacha et al. use conventional precision/recall metrics with a hard decision boundary, which fails to capture the overall performance

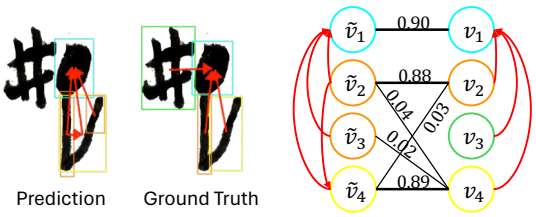

Figure 4. An example of detected objects and predicted graph, alongside ground truth. At the right is the constructed bipartite graph (zero-weight edges not shown). Thick edges represent the matching function $\mathcal{M}$ induced by the matching algorithm. In our notation, $E = \{(v_2, v_1), (v_3, v_1), (v_4, v_1)\}$ and the matching function maps v_1 to $\tilde{v}_1$, v_2 to $\tilde{v}_2$ and v_4 to $\tilde{v}_4$. Therefore, $\hat{E} = \{(\tilde{v}_2, \tilde{v}_1), (\tilde{v}_4, \tilde{v}_1)\}$. Because $\tilde{E} = \{(\tilde{v}_2, \tilde{v}_1), (\tilde{v}_2, \tilde{v}_4), (\tilde{v}_3, \tilde{v}_1), (\tilde{v}_4, \tilde{v}_1)\}$, we get a precision of 0.5 and recall of 1.0.

of the model comprehensively. To resolve these issues, we propose a complementary metric based on a global optimal matching and area under the precision-recall curve.

Formally, we denote $\tilde{V} = \{\tilde{v}_1, \tilde{v}_2, \cdots, \tilde{v}_{\tilde{n}}\}$ as the set of symbols obtained from an object detection model, where $\tilde{v}_i = (\tilde{\mathbf{b}}_i, \mathbf{p}_i)$ is a tuple of a bounding box $\tilde{\mathbf{b}}_i \in \mathbb{R}^4$ and a probability distribution vector $\mathbf{p}_i \in \mathbb{R}^C$ over all symbol classes. A notation assembly prediction on $\tilde{V}$ would be an edge set $\tilde{E} = \{\tilde{e}_1, \tilde{e}_2, \cdots, \tilde{e}_{\tilde{m}}\}$ where each edge $\tilde{e}_i$ is a tuple of two vertices. Similarly, we denote the ground truth notation graph as $G = (V, E)$ with $V = \{v_1, v_2, \cdots, v_n\}, v_i = (\mathbf{b}_i, c_i), E = \{e_1, e_2, \cdots, e_m\}$, where $\mathbf{b}_i \in \mathbb{R}^4$ is a bounding box and $c_i \in \{1, 2, \cdots, C\}$ is a symbol class label.

We first construct a complete weighted bipartite $(\tilde{V}, V)$ where the weight for edge $(\tilde{v}_i, v_j)$ is $w_{ij} = \mathrm{IoU}(\tilde{\mathbf{b}}_i, \mathbf{b}_j) \cdot \mathbf{p}_{i,c_j}$. Here, IoU is the intersection-over-union between the area occupied by the two boxes, defined as:

$$\mathrm{IoU}(\mathbf{b}_i, \mathbf{b}_j) = \frac{\mathrm{Area}(\mathbf{b}_i \cap \mathbf{b}_j)}{\mathrm{Area}(\mathbf{b}_i \cup \mathbf{b}_j)}.$$

Based on this bipartite graph, we find the maximum weighted matching M using the implementation described in [22] and filter the "weak" matching edges with weight w_{ij} less than a threshold T_{match} to get the matching function $\mathcal{M} : V \to \tilde{V} \cup \{\varnothing\}$:

$$\mathcal{M}(v_j) = \begin{cases} \tilde{v}_i, \text{ if } (\tilde{v}_i, v_j) \in M \text{ and } w_{ij} > T_{\mathrm{match}}, \\ \varnothing, \text{ otherwise.} \end{cases}$$

Here, T_{match} is a filtering threshold for matching and we set it to 0.05 without tuning.

After getting the matching function, the ground truth assembly edges are naturally mapped back to edges between predicted vertices. The mapped edge set $\hat{E} = \{(\mathcal{M}(v_i), \mathcal{M}(v_j)) \mid (v_i, v_j) \in E, \mathcal{M}(v_i) \neq \varnothing, \mathcal{M}(v_j) \neq \varnothing\}$ represents a ground truth edge set on detected vertices, which can be used to evaluate predictions $\tilde{E}$ to get a precision and recall. An example is shown in Figure 4.

Most notation assembly models predict a probability of the existence of an edge (v_i, v_j), and the probability is further compared with a threshold T_{predict} to determine whether (v_i, v_j) belongs to the prediction set $\tilde{E}$. By adjusting the model prediction threshold T_{predict}, we can get a series of predictions $\{\tilde{E}_1, \tilde{E}_2, \cdots\}$ and therefore derive a series of precision-recall pairs, which are used to estimate the area-under-the-curve (AUC) score. We refer to the full evaluation metric as "Match+AUC."

"Match+AUC" is an end-to-end evaluation metric for the OMR pipeline with following advantages:

- "Match+AUC" accounts for model performance in both the object detection and notation assembly stages. To be specific, given an object detector's output, a notation assembly model will achieve a higher score if it predicts no edges among redundant objects, since connecting redundant nodes into the assembly graph would greatly affect the final output music score. Also, for the same assembly model, a worse object detector would generate a large amount of redundant and inaccurate objects, making it very hard for the assembly model to distinguish them.

- Instead of a hard rule-based matching used in past methods, "Match+AUC" creates a comprehensive matching among detected symbols and ground truth symbols, making the final score more accurate and sensitive.

- "Match+AUC" evaluates the model using the area under the precision-recall curve, which summarizes performance across a range of threshold choices that could be made by a downstream module or a system user.

We believe that our novel "Match+AUC" is a compelling tool for analyzing OMR pipelines that is complementary to existing approaches.

4. IMPLEMENTATION DETAILS

4.1 Music Symbol Detection

4.1.1 Model Details

We finetune the "large" version of YOLOv8 (YOLOv8l), an object detection model pre-trained on the COCO dataset [23], on MUSCIMA++ v2.0 for music object detection. The model consists of 43.7M parameters and is capable of detecting object bounding boxes and generating corresponding class distributions. The input image size of our model is set to 640.

4.1.2 Training

We used the MUSCIMA++ v2.0 dataset to train and evaluate the music symbol detection model [4]. The images are binarized (pixels are 0/1-valued) and in a size of approximately 3500 × 2000 pixels. For simplicity, we use images with staff lines removed. Additionally, following the exact method described in [6], we split the dataset into 60% training data, 20% validation data, and 20% test data. To effectively train YOLOv8 on these dense images involving many small annotations, which include augmentation dots and piano pedal markings, we have to reduce the image size. Therefore, following the methods used by [8],

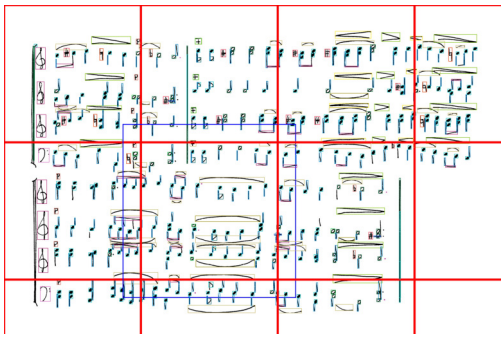

Figure 5. Example of music symbol detection segments for inference. The thick red line indicates the primary cropped area, while the thick blue line represents an extended cropped section designed to include partial symbols that may extend beyond the main cropped area. For better visualization, we only show the extended area of one image crop. Image crops on the right and bottom border of the page are padded to fit into YOLOv8.

given a large music score image, we randomly sample 14 1216×1216 crops and further resize them in to 640×640 to fit the YOLOv8 input requirement.

We fine-tune the YOLOv8 model for 500 epochs with a batch size of 8. We use the AdamW optimizer with a learning rate of 5.5×10^{-5} and a momentum of 0.9, which are automatically set by the YOLOv8 codebase [7]. During training, we use the early stopping strategy with a patience of 100 epochs. We keep the checkpoint with the highest validation performance as our final model.

4.1.3 Inference

Since our detector is trained on cropped data, during the inference stage, we also need to segment the large images into smaller segments. However, partial objects at the edges of these crops would be hard to detect since the model can't see the full object. To resolve this issue, we extend every crop with a margin, which serves as a context for each image. The cropping is visualized in an example in Figure 5. We then perform symbol detection on each extended crop and consolidate the detection results. To make sure the objects on the edges are only detected once, overlapping bounding boxes are filtered based on their Intersection over Union (IoU) overlap rate.

4.2 Notation Assembly

4.2.1 Model Details

We use a 4-layer MLP for ϕ_{MLP}, where the two hidden layers both have hidden dimension 32. The embedding dimension for the symbol class is also set to be 32. We use ReLU [24] as the activation function.

4.2.2 Training

Again we used the MUSCIMA++ v2.0 dataset to train and evaluate the notation assembly model [4]. Following previous work [5, 6], we balance the positive and negative pairs in the training set by filtering out the pairs of nodes that are too distant from each other since they are unlikely to be connected. Before feeding the bounding box coordinates to the model, we normalize them by the image width while keeping the aspect ratio fixed, so that all of the x-coordinate values fit in the range of $[-1, 1]$.

We train our models for 200 epochs with batch size 256, and use Adam optimizer with a learning rate of 0.0001. We evaluate our model every 20 epochs and pick the checkpoint with highest validation Match+AUC as our final model. All of the experiments are conducted with three different random seeds.

In our experiments, we consider three methods for training the notation assembly model:

- A **baseline**, which uses the ground-truth object lists provided in the MUSCIMA++ dataset to train the notation assembly model. This is the setup used in [5].
- A **pipeline**, which runs the music object detection model on the images to construct the training set for the notation assembly model, as discussed in Section 3.2.
- A "soft" variant of the pipeline, where we replace the embedding layer for the symbol class with a linear layer that maps the symbol class probabilities outputted from the music object detection model to a 32-dimensional vector. Note that this linear layer will have the same parameter count (number of classes multiplied by the hidden dimension) as the replaced embedding layer.

4.2.3 Inference

Since we consider both stages together, the input to the notation assembly stage should correspond to the output of the object detection stage. As described in Section 3.2, the detection output is converted into (V', E'). We then pass each pair of nodes to the notation assembly model, and feed the result into our evaluation function. We hypothesize that this realistic setup introduces a distribution shift to the model that was trained on the ground-truth objects and we will make the comparison in Section 5.

5. EXPERIMENTS

In this section, we first report the performance of our music symbol detection model. Then, we compare the performance of different notation assembly training pipelines using the evaluation metric described in Section 3.3.

5.1 Music Symbol Detection

Following the evaluation protocols of the Pascal VOC challenge [25], which is used by previous methods [8, 10, 11], we present both the mean average precision (mAP) and the weighted mean average precision, as detailed in Table 1. To elaborate, a predicted bounding box $\tilde{\mathbf{b}}_i$ is thought to be a true positive only if $\text{IoU}(\tilde{\mathbf{b}}_i, \mathbf{b}_j) > 0.5$ for some ground truth box $\mathbf{b}_j$. Then, average precision (AP) computes the area under the precision-recall curve, providing a single value that encapsulates the model's precision and recall performance. The weighted/unweighted mean Average Precision (mAP) extends the concept of AP by calculating the average AP values across multiple object classes,

Models	# Classes		mAP (%)	Weighted mAP (%)
YOLOv8 + cropping (ours)	163	(all)	84.79	92.67
YOLOv8 + cropping (ours)	73	(essential)	85.67	89.96
YOLOv8 + cropping (ours)	20		**94.22**	**95.72**
YOLOv4 + CBAM [8]	20		91.8	94.56†
PP-YOLO-V2 [8]	20		91.1	–
YOLO-X [8]	20		90.4	–
YOLOv4 [8]	20		89.1	–
Faster R-CNN [8]	20		86.2	–

Table 1. Object detection results on test set. "mAP" is mean average precision. We compared it with results reported by [8]. The lower block is included for comparability with the 20-class setting from past work. †: Value computed from average precision per class reported in [8].

Models	# Classes	Match+AUC Average	Match+AUC S.D.
MLP baseline (train on ground truth objects)	73	92.44 ± 0.24	
+ pipelined training (ours)	73	93.09 ± 0.16	
+ pipelined training + soft label (ours)	73	$\mathbf{95.00} \pm 0.18$	
MLP baseline (train on ground truth objects)	163	83.97 ± 3.04	
+ pipelined training (ours)	163	85.76 ± 0.42	
+ pipelined training + soft label (ours)	163	$\mathbf{87.10} \pm 1.19$	

Table 2. Multi-stage system results (test set) using our Match+AUC metric.

taking into account the number of occurrences of each class in a weighted or unweighted manner. Our experiments are conducted with the MUSCIMA++ v2.0 dataset, while the authors of most previous methods [10, 11] have only tested their models on MUSCIMA++ v1.0. This introduces a misalignment between our results. Thanks to Zhang et al. [8], who provided reproduced results of most previous methods on MUSCIMA++ v2.0, we directly report their reproduced results in the table.

Our model outperforms Zhang et al.'s method on their selected 20 classes by 2.4% (mAP, absolute), likely due to the improvements in YOLOv8 compared to v4.

5.2 Notation Assembly

In this section, we complete the multi-stage OMR system by chaining different notation assembly models to the best music object detection model we trained in Section 5.1. We use the metric we designed in Section 3.3 to report the end-to-end performance of the OMR system.

In Table 2, we compare the notation assembly systems trained with baseline training, pipelined training, and soft pipelined training as described in Section 4.2. We found that pipelined training improves the Match+AUC score by 0.65% (essential) and 1.79% (all), absolute, and incorporating the soft class label further increases the performance by 1.91% (essential) and 1.34% (all), absolute. Training the notation assembly model on the detection model output and using the soft label probability to represent the class information, we are able to improve the Match+AUC of the OMR system by 3.13%. We hypothesize that pipelined training helps the assembly model adapt to any inaccuracies our object detector has, and incorporating the soft class labels enables the assembly model to consider alternative class labels, not just those chosen by the object detector.

6. CONCLUSION AND FUTURE WORK

In our study, we reconsider a multi-stage OMR pipeline built and evaluated using the MUSCIMA++ dataset. We first propose a state-of-the-art music symbol detector, serving as a strong preprocessor for the notation assembly stage. We then propose a training pipeline in which notation assembly is learned from imperfect object detection outputs (rather than ground-truth objects), which leads to higher performance. Finally, we introduce an evaluation score, Match+AUC, which can jointly consider the error in both detection and assembly stages, allowing evaluation of the two stages together.

Match+AUC is not restricted to being an evaluation metric. Future research could explore the application of Match+AUC within a joint training objective function for both the object detection and notation assembly stages. This approach would enable the entire model to be optimized for retrieving a globally optimal music notation graph.

In this study, we focused on the object detection and notation assembly stages in the OMR pipeline. Progress on the encoding stage is also required for a complete OMR solution; while the music notation graph arguably contains the essential information for recovering a score [4], conversion of such graphs into standard formats remains unsolved.

7. ACKNOWLEDGMENTS

The authors wish to express our deepest gratitude to all creators of the public OMR datasets for their dedication

and generosity in collecting and sharing these invaluable resources. We extend our sincere thanks to Carlos Peñarrubia for his assistance in clarifying questions regarding the reproduction of their method. We are also grateful to Tim Althoff for his insightful comments on our evaluation metric. Special thanks go to Victoria Ebert and Teerapat Jenrungrot for providing us with essential materials in the OMR field. Finally, we sincerely appreciate the constructive reviews, which have significantly enhanced the rigor and completeness of this paper.

8. REFERENCES

[1] D. Bainbridge and T. Bell, "The challenge of optical music recognition," *Computers and the Humanities*, vol. 35, pp. 95–121, 05 2001. [Online]. Available: https://doi.org/10.1023/A:1002485918032

[2] A. Rebelo, I. Fujinaga, F. Paszkiewicz, A. R. S. Marcal, C. Guedes, and J. S. Cardoso, "Optical music recognition: state-of-the-art and open issues," *International Journal of Multimedia Information Retrieval*, vol. 1, no. 3, pp. 173–190, Oct. 2012. [Online]. Available: https://doi.org/10.1007/s13735-012-0004-6

[3] J. Calvo-Zaragoza, J. Hajič, Jr., and A. Pacha, "Understanding optical music recognition," *ACM Comput. Surv.*, vol. 53, no. 4, jul 2020. [Online]. Available: https://doi.org/10.1145/3397499

[4] J. Hajič and P. Pecina, "The MUSCIMA++ dataset for handwritten optical music recognition," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 01, 2017, pp. 39–46. [Online]. Available: https://doi.org/10.1109/ICDAR.2017.16

[5] C. Peñarrubia, C. Garrido-Munoz, J. J. Valero-Mas, and J. Calvo-Zaragoza, "Efficient notation assembly in optical music recognition," in *Proceedings of the 24th International Society for Music Information Retrieval Conference*. ISMIR, Dec. 2023, pp. 182–189. [Online]. Available: https://doi.org/10.5281/zenodo.10265253

[6] A. Pacha, J. Calvo-Zaragoza, and J. Hajič, Jr., "Learning notation graph construction for full- pipeline optical music recognition," in *Proceedings of the 20th International Society for Music Information Retrieval Conference*. ISMIR, Nov. 2019, pp. 75–82. [Online]. Available: https://doi.org/10.5281/zenodo.3527744

[7] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[8] Y. Zhang, Z. Huang, Y. Zhang, and K. Ren, "A detector for page-level handwritten music object recognition based on deep learning," *Neural Computing and Applications*, vol. 35, no. 13, pp. 9773–9787, May 2023. [Online]. Available: https://doi.org/10.1007/s00521-023-08216-6

[9] A. Fornés, A. Dutta, A. Gordo, and J. Lladós, "CVC-MUSCIMA: a ground truth of handwritten music score images for writer identification and staff removal," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 15, no. 3, pp. 243–251, Sep. 2012. [Online]. Available: https://doi.org/10.1007/s10032-011-0168-2

[10] A. Pacha, K.-Y. Choi, B. Coüasnon, Y. Ricquebourg, R. Zanibbi, and H. Eidenberger, "Handwritten music object detection: Open issues and baseline results," in *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, 2018, pp. 163–168. [Online]. Available: https://doi.org/10.1109/DAS.2018.51

[11] A. Pacha, J. Hajič, Jr., and J. Calvo-Zaragoza, "A baseline for general music object detection with deep learning," *Applied Sciences*, vol. 8, 2018. [Online]. Available: https://doi.org/10.3390/app8091488

[12] A. Konwer, A. K. Bhunia, A. Bhowmick, A. K. Bhunia, P. Banerjee, P. P. Roy, and U. Pal, "Staff line removal using generative adversarial networks," in *2018 24th International Conference on Pattern Recognition (ICPR)*, 2018, pp. 1103–1108. [Online]. Available: https://doi.org/10.1109/ICPR.2018.8546105

[13] J. Calvo-Zaragoza, A. Pertusa, and J. Oncina, "Staff-line detection and removal using a convolutional neural network," *Machine Vision and Applications*, vol. 28, no. 5, pp. 665–674, Aug. 2017. [Online]. Available: https://doi.org/10.1007/s00138-017-0844-4

[14] A.-J. Gallego and J. Calvo-Zaragoza, "Staff-line removal with selectional auto-encoders," *Expert Systems with Applications*, vol. 89, pp. 138–148, 2017. [Online]. Available: https://doi.org/10.1016/j.eswa.2017.07.002

[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788. [Online]. Available: https://doi.org/10.1109/CVPR.2016.91

[16] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2004.10934

[17] A. Baró, P. Riba, J. Calvo-Zaragoza, and A. Fornés, "From optical music recognition to handwritten music recognition: A baseline," *Pattern Recognition Letters*, vol. 123, pp. 1–8, 2019. [Online]. Available: https://doi.org/10.1016/j.patrec.2019.02.029

[18] F. Foscarin, F. Jacquemard, and R. Fournier-S'niehotta, "A diff procedure for music score files," in *Proceedings of the 6th International Conference on Digital Libraries for Musicology*, ser. DLfM '19, 2019, p.

58–64. [Online]. Available: https://doi.org/10.1145/3358664.3358671

[19] J. Hajič, Jr., "A case for intrinsic evaluation of optical music recognition," *International Workshop on Reading Music Systems*, 2018.

[20] D. Byrd and J. Simonsen, "Towards a standard testbed for optical music recognition: Definitions, metrics, and page images," *Journal of New Music Research*, vol. 44, 07 2015. [Online]. Available: https://doi.org/10.1080/09298215.2015.1045424

[21] P. Torras, S. Biswas, and A. Fornés, "The common optical music recognition evaluation framework," *arXiv preprint arXiv:2312.12908*, 2023.

[22] D. F. Crouse, "On implementing 2D rectangular assignment algorithms," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 4, pp. 1679–1696, 2016. [Online]. Available: https://doi.org/10.1109/TAES.2016.140952

[23] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 740–755. [Online]. Available: https://doi.org/10.1007/978-3-319-10602-1_48

[24] A. F. Agarap, "Deep learning using rectified linear units (ReLU)," 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1803.08375

[25] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015. [Online]. Available: https://doi.org/10.1007/s11263-014-0733-5