# Confidence-Rated Predictions from Deep Learning Ensembles for Music Object Detection

Urs Gut

***Abstract*—The RealScore project is ScorePad's research partner for their sheet music scanning service. In recent years, multiple deep neural network architectures have been tested as part of this project. Although some models exhibited excellent predictive performance, the problem of overconfident predictions persists, at least for certain musical symbols.**

**In this thesis, we provide a summary of relevant methods for estimating model uncertainty in neural networks. Specifically, we focus on ensemble methods that are economic in terms of memory and computational costs and are compatible with the current network architecture. We present an implementation of the BatchEnsemble approach and a visualization tool that is based on post-processing detections using a combination of Non-Maximum Suppression and the Weighted Boxes Fusion Algorithm.**

**Our experimental results are preliminary in nature as a consequence of unanticipated technical issues. Despite the sub-optimal performance of our prototype, the proposed visualization method shows promising practical benefits for quickly identifying low confidence detections.**

***Index Terms*—Predictive Uncertainty, Ensemble Learning, Economic Ensembles, Confidence Visualization, Weighted Boxes Fusion, Optical Music Recognition, Single-shot Alignment Network**

## I. Introduction

The RealScore research project aims at improving the Optical Music Recognition (OMR) pipeline for ScorePad's sheet music scanning service [1]. In recent years, several deep neural network architectures have been evaluated in the context of the RealScore project [2], [3]. While some of these models have achieved state-of-the art predictive performance, there still exists the problem of overconfident predictions for certain classes of musical symbols [3].

Generally, deep neural networks (NNs) are poor at quantifying predictive uncertainty in the model parameters (i.e. epistemic uncertainty), and often produce overconfident predictions [4], [5]. Overconfident and incorrect predictions are problematic for many applications of deep learning [6], [7]. As a consequence, there exists a growing number of approaches for quantifying predictive uncertainty in NNs.

Broadly speaking, these approaches can be divided into two groups, referred to as 'Bayesian' and 'non-Bayesian'. Bayesian approaches are considered to offer a more consistent mathematical framework [8]. Yet, at the same time, they are usually more difficult to implement and often have a higher computational cost [4]. Moreover, since exact Bayesian inference is computationally intractable, the quality of the predictive uncertainty obtained using Bayesian approaches depends on the quality of the approximation to the true Bayesian posterior distribution [8]. This approximation is influenced by (1) the compromises resulting from computational constraints and (2) the adequacy of the chosen prior distribution [4], [5].

Because of these complexities, in recent years, non-Bayesian approaches, especially methods that take advantage of ensemble predictions, have regained attention [4], [9]. Aggregating over multiple predictions from ensembles of models is a well-known technique to improve the predictive performance in machine learning [9]. Good ensembles result when the predictions of the specific ensemble members are both accurate and have independent errors [9].

Typically, computational and memory costs of ensemble approaches increase linearly with ensemble size in both training and testing [9]. However, these higher costs seem acceptable because ensembles have been shown to outperform Monte-Carlo-Dropout (MC-Dropout) approaches, which are a popular Bayesian alternative to ensembles and have a relatively low computational cost [5], [10]. Moreover, the reduction of the computational and memory overhead of ensembles has been the object of many recent studies. Thus, several methods for decreasing memory and computational costs have already been developed under the label of 'economic ensembles' [9], [11]–[13].

Even more importantly, ensembles seem to offer more robust estimates of predictive uncertainty. [14] postulated that variational Bayesian methods only capture local uncertainty modes whereas ensembles explore different global modes in the loss landscape [9]. Hence, most state-of-the-art approaches to estimating predictive uncertainty, whether formulated within a Bayesian or a non-Bayesian framework, rely on ensembles [5], [7], [9], [12], [13], [15]–[18].

Consequently, our literature survey (section II) focuses on ensemble approaches that are economic and are potentially compatible with the current architecture of the RealScore project. Specifically, we will give a short introduction to the BatchEnsemble [9], HypernetEnsemble [12], BatchNormEnsemble [13], Masksembles [19], SnapshotEnsemble [20], and HatchEnsemble [18] approaches. Moreover, we briefly present the Late-Phase Weights method, which is potentially compatible with the ensemble approaches presented and could be used as a further methodological extension.

Following the survey, we detail the Single-shot Alignment Network architecture [21] and give a conclusion detailing which of the surveyed ensemble approaches seems to be the most readily implementable with respect to this architecture (section III). Next, we detail the prototype implemented and introduce the Weighted Boxes Fusion Algorithm [22] as a means to visualize predictions from ensembles of detectors (section IV). After presenting the results generated by our

Urs Gut was with the Centre for Artificial Intelligence, ZHAW School of Engineering, Technikumstrasse 71, 8400 Winterthur, e-mail: u.gut@posteo.de.

Submitted on 2022-01-31

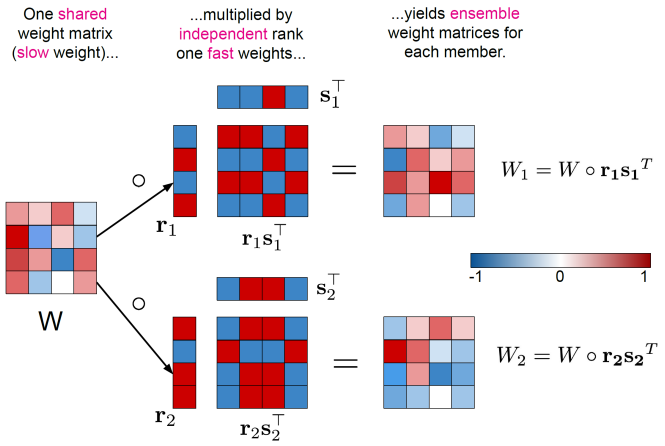


Fig. 1. The BatchEnsemble weight generation algorithm illustrated for the case of two ensemble members. (Copy of the original Figure 2 in [9]).

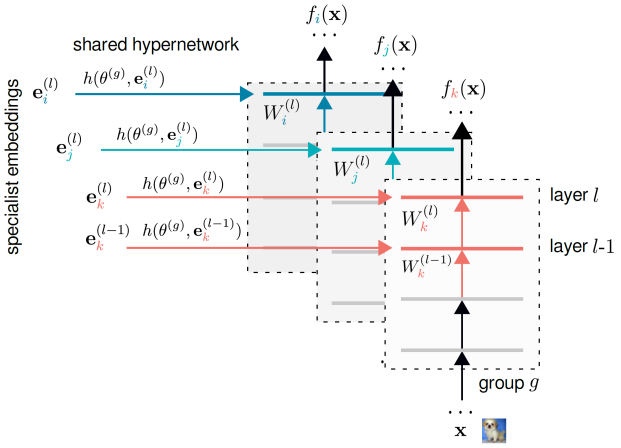


Fig. 2. The hypernetwork with parameters $\theta^{(g)}$ is shared across $N$ specialists and within a group of layers $g$ (dashed area). The specialist weights $W_i^{(l)}$ for some layer $l$ and specialist $i$ are generated using a low-dimensional weight embedding vector $\mathbf{e}_i^{(l)}$. Thus, for each input $\mathbf{x}$ results a set of predictions $\{f_i(\mathbf{x})_{i=1}^N\}$. Averaging over the $N$ specialists yields the ensemble prediction $f(\mathbf{x})$. (Copy of the original Figure S1 in [12]).

prototype (section V), we provide a final discussion (section VI) and a conclusion (section VII) where we outline paths for future research.

## II. Survey of Relevant Ensemble Approaches

### A. BatchEnsemble

BatchEnsemle has been proposed by [9] to remedy the large computational and memory overhead of conventional ensemble methods such as [4], [23]. The savings are achieved by a weight generation mechanism that generates the weight matrices for each ensemble member by the Hadamard product (i.e. element-wise product) between: (1) a single weight matrix that is shared among all members; and (2) multiple rank-one matrices (products of two vectors) that vary between ensemble members [9]. The shared weights are referred to as 'slow weights' and the rank-one matrices are referred to as 'fast weights' [9]. The latter are calculated from a tuple of trainable vectors, i.e., $r_i$ and $s_i$, which are specific to each ensemble member [9] (Fig. 1). The ensemble weights $\overline{W_i}$ result from the Hadamard product between slow and fast weights [9]:

$$\overline{W_i} = W_i \circ F_i, \text{where } F_i = r_i s_i^\top, \tag{1}$$

### B. HypernetEnsemble

A hypernetwork is a NN that generates the weights of an other NN (i.e., the main network) from an embedding input [7], [12], [24]–[26]. [7] investigated the ability of hypernetworks to produce arbitrarily complex, multi-modal weight posteriors for Bayesian variational inference. Thus, they quantified model uncertainty without relying on ensembles. [12] built upon the concept of parameter sharing using hypernetworks and leveraged this technique to generate economic ensembles. To build an ensemble of $N$ models, they input $N$ different embeddings to the same hypernetwork to generate $N$ different ensemble weight matrices [12]. The weight matrix $W$ of the main NN is partitioned or 'chunked' layer-wise to enable compression and allow a much smaller hypernetwork to learn the parameters of the primary NN [12]. Thus, they calculate the weights for a certain layer $l$ of the primary network as:

$$W^{(l)} = \theta \mathbf{e}^{(l)}, \tag{2}$$

where $\mathbf{e}^{(l)}$ are the embedding vectors of each layer $l$, which are ensemble member specific and are trained in parallel, and $\theta$ are the hypernetwork parameters, which are shared among all ensemble members [12] (Fig. 2).

### C. BatchNormEnsembles

[13] demonstrated how to use BatchNorm units (cf. [27]) to construct economic ensembles. BatchNorm units possess a learnable multiplicative (scale) parameter $\gamma$ and an additive (shift) parameter $\beta$, which are not only low-dimensional but have also a large expressive power [13]. Consequently, learning only $\gamma$ and $\beta$ while freezing the remaining weights can lead to significantly lower loss than learning random subsets of other weights [13], [28], [29]. Thus, ensembles can be constructed by training $\gamma$ and $\beta$ member-wise and sharing the rest of the weights across ensemble members [13].

### D. HyperBatchEnsembles

[17] combine the already introduced BatchEnsemble approach ([9]) with a procedure that involves a random search over different hyperparameter settings. Thereby, both ensemble member weights and ensemble member hyperparameters are learned end-to-end and in parallel during training resulting in HyperBatchEnsembles [17].

### E. Masksembles

Masksembles were proposed by [19] as an economical ensemble method. [19] conceptualize a spectrum of ensemble-like models of which MC-Dropout and conventional deep ensembles are extreme examples. While MC-Dropout relies

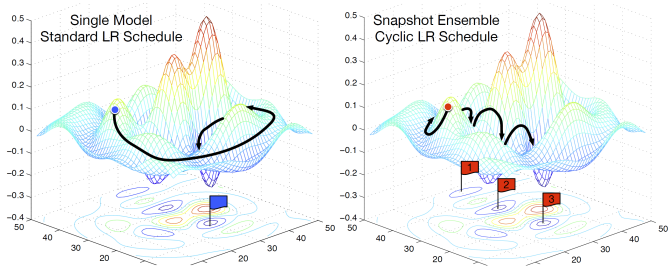


Fig. 3. Left: Stochastic Gradient Descent optimization with a typical learning rate schedule. Right: SnapshotEnsembling. The model runs through several learning rate annealing cycles, leading to the exploration of multiple local minima [20]. A snapshot is taken at each minimum and used during test-time to construct an ensemble. (Copy of the original Figure 1 in [20]).

on a potentially infinite number of highly correlated models, conventional ensembles are based on a relatively small number of independent models [19]. This independence arises from the fact that the individual models do not share any weights and are trained completely separately [4], [19]. In contrast, MC-Dropout models use shared weights and ensembles are obtained by applying random binary masks to drop a portion of network activations simultaneously [8], [19]. In practice, the randomly sampled masks often overlap significantly. Hence, predictions may be highly correlated, which potentially leads to underestimated uncertainty [19]. To remedy this issue, [19] propose to use a limited number of predefined binary masks whose overlap can be controlled. Consequently, these masks are used to drop corresponding network activations which results in sufficiently decorrelated models [19]. In their experiments they achieve a performance comparable to conventional ensembles at a fraction of the cost [19].

### F. SnapshotEnsemble

The SnapshotEnsemble approach of [20] tries to achieve the seemingly paradoxical goal of producing an ensemble at no additional training cost. Their method leverages work on cyclic learning rate schedules [30], [31]. [20] lower the learning rate at a very fast pace, thus encouraging the model to converge quickly to its first local minimum. Then the optimization is continued with a higher learning rate to dislodge the model from this local minimum again [20]. This procedure is repeated multiple times. At each local minimum, the model is saved (i.e. a snapshot is taken, Fig. 3). [20] report consistently lower error rates than single models, but do not quite match the performance of conventional ensembles.

### G. HatchEnsemble

HatchEnsemble uses a standard basic NN architecture as a so-called 'SeedNet' [18]. After the SeedNet has been trained to convergence, a series of function-preserving transformations is used to generate the so-called 'HatchNets' (function-preserving means these transformations do not compromise the function already learned during training). The HatchNets are wider then the SeedNet because additional parameters, i.e. neurons in the case of fully connected layers and channels in the case of convolutional layers, are inserted in the HatchNet's layers [18]. These additional parameters are randomly copied from existing parameters in the SeedNet. Moreover, Gaussian noise is added to the copied parameters to amplify diversity between the HatchNets (or ensemble members, respectively). [18] report faster training for HatchNets due to the fact that the SeedNet has already converged in its own parameter space. Thus, the HatchNets need to explore a small parameter subspace, only. In [18]'s experiments, the model diversity of HatchNets was higher than that of MC-Dropout models.

### H. Late-Phase Weights

The Late-Phase Weights Algorithm is not an actual ensemble method in its own right. Rather, it is a method that can be combined with many of the aforementioned ensemble approaches and potentially results in even more substantial savings regarding computational, memory and training-time costs. To apply Late-Phase Weights to a given NN the weights $W$ need to be specified in terms of two components, i.e., base and late-phase weights ($\theta$ and $\phi$, respectively), whereby the function $W = h(\theta, \phi)$ defines the interaction between the two weight components [13]. Both components are learned as a single model until time step $T_0$ (a hyperparameter of the algorithm), after which $K$ late-phase components $\phi = \{\phi_k\}_{k=1}^{K}$ are introduced [13]. These late-phase components (i.e., ensemble members) can be generated via an arbitrary economic ensemble method, provided the latter allows for training the late-phase components in parallel until convergence [13].

## III. The Current RealScore Architecture and its Compatibility with Economic Ensembles

### A. Single-shot Alignment Network

Currently, the RealScore Project uses a Single-shot Alignment Network ($S^2$A-Net) [21]. Single-shot detectors represent one of the two main frameworks that are implemented in modern object detectors [6]. The alternative framework, i.e., two-stage detectors, rely on a Region Proposal Network (RPN) and an R-CNN detection head [21]. Typically, in a first stage, the RPN is used to generate Regions of Interest (RoIs) from horizontal anchors. Via an RoI pooling operation, features are then extracted from the RoIs [21]. In the second stage, the R-CNN head regresses bounding boxes and classifies them into different categories [21]. This two-stage framework was used to build detectors that reached state-of-the-art performances regarding accuracy but that did not match the speed of single-shot detectors [21].

With their $S^2$A-Net, [21] were able to present a single shot detector that offers comparable accuracy as R-CNN based detectors without compromises regarding speed. They identified severe misalignment between heuristically defined anchors and axis-aligned convolutional features, as the main source of inconsistency between the classification score and localization accuracy in single-stage detectors [21]. Their solution to this issue are the two modules that make up the $S^2$A-Net detection head: the Feature Alignment Module (FAM) and the Oriented Detection Module (ODM) [21].

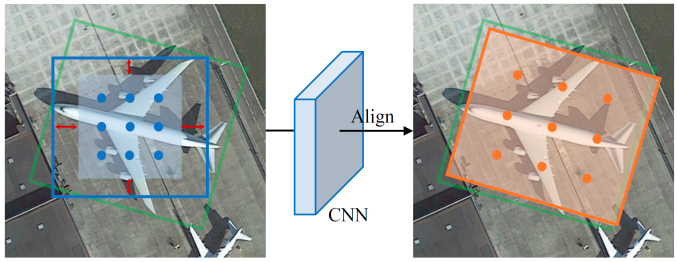


Fig. 4. Illustration of the misalignment problem (red arrows) between an anchor box (blue bounding box) and convolutional features (light blue rectangle). [21]'s solution is to refine the initial anchor into a rotated anchor (orange bounding box). Consequently, the feature sampling locations (orange points) are adjusted with the guide of the refined anchor box. This finally yields the aligned deep features [21]. Green box = ground truth. (Copy of the original Figure 1 a) in [21]).

The FAM generates anchors that accurately cover objects with an Anchor Refinement Network (ARN) and then adaptively aligns features according to their corresponding anchor boxes by applying an Alignment Convolution (ACL) (Fig. 4) [21].

Next, the ODM adopts active rotating filters (ARF) to encode orientation information, which in turn can be used to produce orientation sensitive features (adding orientation specific channels to the respective features). The bounding box regression task benefits from these orientation sensitive features, but the object classification task relies on orientation-invariant features [21]. Therefore, [21] pool the orientation-sensitive features by choosing the orientation channel with the strongest response. Finally, the orientation-sensitive and the orientation-invariant features are fed into two sub-networks to regress the bounding boxes and classify the categories [21] (Fig. 5).

Because of the high accuracy of the oriented bounding box detections and the computational efficiency of the fully convolutional single-shot detector architecture, the $S^2$A-Net is suited particularly well for large images (i.e. 4000 $\times$ 4000 pixels) where objects are densely clustered, as is the case with musical symbols in the DeepScores data set [3], [21].

The implementation of [21] was adopted for the RealScore project [32]. Like the original implementation of [21], it is based on the MMDetection toolbox [33]. Moreover, it uses project specific extensions of the code base, namely: (1) an Oriented Bounding Box Annotations module [34] and (2) an adapted version of the MMCV code base [35]. Regarding the architecture, the $S^2$A-Net of [21] was chosen as detection head, a ResNet-50 ([36]) as the network backbone and a Feature Pyramid [37] with five levels as the network neck. Like in [21], the loss of this $S^2$A-Net version is a multi-task loss (i.e., classification task and bounding box regression task, respectively) that consists of (1) a classification loss (Focal loss) and (2) a smooth L1 loss as the regression loss. During inference, the input image is passed to the backbone network to extract pyramid features [21]. Consequently, these are fed into the FAM to generate refined anchors and aligned features [21]. Next, the ODM encodes the orientation information and makes predictions [21]. In the end, the top-$k$ (e.g., $k$=2000) predictions with the highest confidence score are chosen and, after Non-Maximum Suppression (NMS) [38], the final detections are provided [21].

Contrarily to the old watershed detector architecture, which was previously used in the RealScore project, the network output of the $S^2$A-Net comprises of the confidence scores (of the classification) and the bounding boxes only. No feature maps are provided that could be compared to the energy, class and bounding box maps of the watershed detector [2]. Thus, no pixel-wise confidence scores can be retrieved using the $S^2$A-Net.

### *B. Conclusion Regarding the Different Economic Ensemble Methods*

Ideally, a useful compromise between computational costs and ensemble diversity can be found by choosing one of the economic ensemble approaches presented. Moreover, the $S^2$A-Net is a complex architecture, thus, changes to the NN need to be well considered regarding their costs and benefits.

SnapshotEnsemble seems to be the least complex approach and could possibly be implemented quickly. However, the member predictions coming from SnapshotEnsembles might be too similar and yield no accurate estimate of the predictive uncertainty. Maksembles are also relatively easy to implement. In addition, the predefined binary masks, which are used to drop activations within the dropout layers, might sufficiently decorrelate the ensemble members. However, they rely on MC-Dropout layers of which there are none in the current $S^2$A-Net architecture. Similarly, BatchNormEnsemble could be readily implemented if the mini batch size was $> 1$. Yet, the DeepScore data set consists of very large images that necessitate to process mini batches of size $= 1$. Thus, the BatchNorm Layers in the $S^2$A-Net are replaced by Layer Normalization [39]. It is unclear if the parameters of Layer Normalization could be used to generate economic ensembles analogous to the BatchNormEnsemble approach.

Despite [12] and [17]'s claims to the simplicity of their approaches, HypernetEnsemble and HyperBatchEnsemble seem to require fundamental changes to the implementation of the $S^2$A-Net, which are beyond the scope of this thesis. The same is true for the HatchEnsemble approach. Moreover, the random copying of weights involved using the latter approach might cause undesired behaviour within the relatively complex architecture and compromise the training of the $S^2$A-Net.

The BatchEnsemble approach appears to be the best first choice for a prototype. Besides its concise mathematical formulation, this approach might also be among the most cost efficient: According to the experiments of [9] the speedup at test time was 3X and memory reduction was 3X with an ensemble size of 4 compared to conventional ensembles. From an implemetational point of view, a slight setback only results from the fact that the original implementation uses the TensorFlow API, whereas the $S^2$A-Net code is written using PyTorch. Luckily, [40] presented BatchEnsemble Convolutional Layers implemented with the PyTorch API that can be readily modified and integrated in the current code base. Moreover, after a first prototype has been implemented and

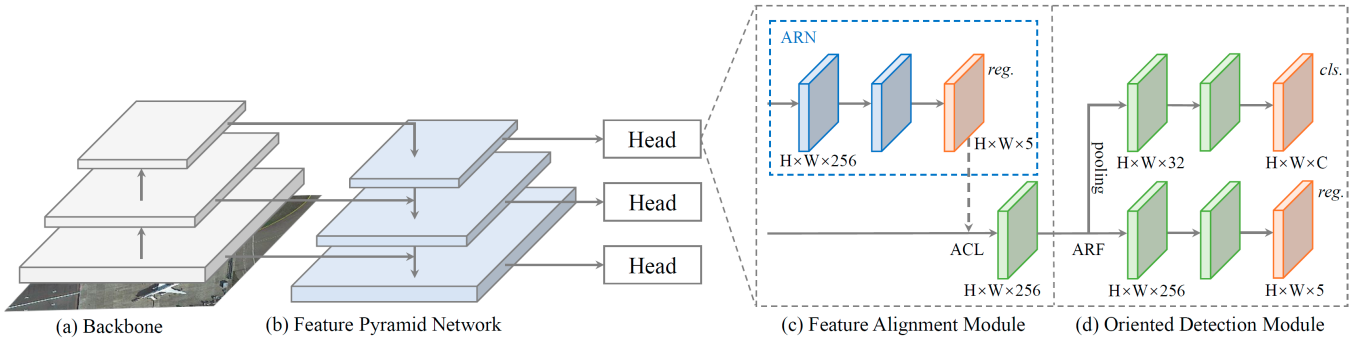


Fig. 5. Illustration of the S$^2$A-Net architecture with the detection head (right part) applied to all prediction levels of the Feature Pyramid Network. Abbreviations: ARN = Anchor Refinement Network; ACL = Alignment Concolution Layer; ARF = Active rotating filters; cls. = classification branch; reg. = regression branch (Copy of the original Figure 2 in [21]).

tested, one could consider a Late-Phase Weight version of this prototype. Initializing ensembles late in training would allow for an even more substantial cost saving.

## IV. Assessing the Predictive Uncertainty of the S$^2$A-Net

### A. Implementational Details of the S$^2$A-Net BatchEnsemble Prototype

Theoretically, all fully connected layers and all convolutional layers could be replaced by a BatchEnsemble-version of the respective layer. However, in case debugging would become necessary, one would have to consider a multitude of possible error locations. Thus, we decided to first concentrate our efforts on implementing BatchEnsemble layers in the S$^2$A-Net detection head only. The backbone and neck of the NN remained unchanged.

Within the head, we chose four key locations where we replaced conventional convolution layers with a BatchEnsemble version (implementation as in: [41]). Specifically, we chose the last convolution layer for each, the FAM classification branch, the FAM regression branch, the ODM classification branch and the ODM regression branch, respectively (cf. Fig. 5). These layers seemed to have the biggest impact on the S$^2$A-Net's predictions.

The BatchEnsemble convolution layers internally reshape the layer-inputs by copying and concatenating the inputs $m$ times, where $m$ = size of the ensemble. Consequently, the outputs of these layers are $m$-times larger and contain the features of all ensemble members. For each of the $m$ members, the loss can be calculated during training. To enable parallel training in one forward-pass, we calculate the mean loss over all members before taking the next optimizer step. Because this is a differentiable mathematical operation, the weights of all members can be properly updated during back propagation and we can still log a single loss value.

The architecture of the S$^2$A-Net head posed one further difficulty: The ARN sub-network within the FAM uses an Alignment Convolution Layer (ACL, cf. [21], Fig. 5). This special convolutional layer would require a re-implementation from scratch in order to make it fully compatible with our BatchEnsemble approach. However, to start experimentation as soon a possible, we decided to run our first version of the prototype without implementing BatchEnsemble in the ARN. Instead, we directly passed the inputs of the first ensemble member to the ARN.

Our complete code base is available on GitHub: [42]. The key classes, i.e., S2ANetHeadBE and S2ANetDetectorBE, can be found within the mmdet models module (paths: mmdet/models/anchor_heads_rotated/s2anet_head_BE.py and mmdet/models/detectors/s2anet_BE.py, respectively). The paths to the training- and testing-script are tools/train_BE.py and tools/test_BE.py, respectively.

### B. Class-wise Average Precision

To assess the test performance of our prototype, we calculate class-wise average precision (AP) [6] for a test set of 200 images. Detections are non-maximum suppressed (score threshold = 0.3, IoU-threshold = 0.1). After this post-processing, detections are considered to be true positives (TP) if their labels match with the ground truth label and their intersection over union (IoU) with the ground truth annotations is $\geq 0.5$. Otherwise, they are regarded as false positives (FP). The IoU is determined as (cf. [6]):

$$IoU(b, b^g) = \frac{area(b \cap b^g)}{area(b \cup b^g)}, \tag{3}$$

where $b$ is the detected bounding box and $b^g$ is the respective ground truth bounding box. Using TP and FP, the average precision is calculated as the area under the precision-recall curve [6]. Class-wise AP values are calculated for each class of musical symbols in the test set. We process detections and calculate APs separately per ensemble member. Consequently, summary metrics can be calculated that aggregate the class-wise APs over all ensemble members. To provide information on the spread of the class-wise APs across ensemble members, not only the mean, but also the 0.25 percentile, the 0.75 percentile, the minimal and the maximal class-wise AP values are reported.

### C. Confidence Visualization

The class-wise APs provide overall summary metrics only. To be able to quickly assess problematic detections, a visualization becomes necessary. The standard method used

by single model detectors is to generate a specific number of detections per input image (e.g. 2000) and then apply NMS to generate the final predictions [38]. This first sorts all detections by their confidence scores. Next, the detection with the maximum score is chosen. All other detections that have an IoU larger than a predefined threshold with the chosen detection are discarded [22], [38]. In addition, often a threshold for the minimal confidence score is chosen. Finally, the detections thus post-processed are plotted on the respective input images.

With detection proposals coming from several ensemble members, rather than selecting the best detections per member, a method must be found that is able to aggregate the best detections across all ensemble members [22].

*1) Plotting Transparent Polygons:* As a first go-to approach, we separately applied NMS to the predictions of each ensemble member. Then we plotted the non-maximum suppressed detections of all members using transparent polygons with an alpha value of $1/m$ (where $m$ = ensemble size). Thus, areas with no transparency corresponded to detections that were present in all ensemble members. Increasing transparency correlated to detections that were less consistently detected across ensemble members. We could then have mapped the different transparency levels to a color map resulting in a heat map representation for the detections. However, the method's results were unsatisfactory (cf. section V-B; Appendix A, Fig. A1). Hence, we explored an alternative solution.

*2) Weighted Boxes Fusion Algorithm:* The Weighted Boxes Fusion Algorithm (WBF) was introduced by [22] to provide accurate detections for ensembles of detectors. The algorithm comprises six steps:

1) All predicted detections of all ensemble members are added to a single list **B**, which is sorted in decreasing order of the confidence scores **C** [22].
2) Empty lists **L** and **F** are declared for detection clusters and fused bounding boxes, respectively. In the list **L**, each position can represent a set of detections (or a single detection), which form a cluster. In **F**, each position contains only one bounding box, i.e. the fused bounding box from the corresponding cluster of detections in **L** (for the calculation of the fused box see equations below) [22].
3) Now, iterate through predicted detections in **B** to find a matching box in the list **F**. A valid match is found, when the IoU of a detection in **B** with a fused bounding box in **F** is larger than a predefined threshold (i.e., an IoU-threshold; we set it to IoU = 0.1 for our experiments). [22]. Moreover, thresholds can be defined for the fused score and the skip threshold. By applying the latter, detections with a score lower than the threshold are skipped.
4) If no match is found, the respective detection from list **B** is added to the end of lists **L** and **F** as new entries. We then proceed the search for matches, with the next detection listed in **B** [22].
5) If a match is found, the detection is added to list **L** at the position ($pos$) corresponding to the matching fused bounding box in list **F** [22].
6) Finally, the fused box coordinates and its confidence score need to be updated.

The S$^2$A-Net uses rotated bounding boxes. Hence, we need to adjust the calculation of the IoU to be compatible with rotated boxes (adapting code from the *polyiou* module in [34] and the *ensemble-boxes* module [43]). Moreover, the coordinates calculation requires expansion to include all four vertex points (i.e., $x_1, y_1, x_2, y_2, x_3, y_3, x_4$ and $y_4$) of a rotated bounding box. In contrast to [22], we first apply NMS to the detections proposed by each ensemble member (score threshold = 0.3, IoU-threshold = 0.1). Thus, we reduce the number of imprecise detections per detectable object and speed up the calculation of the fused bounding boxes.

Following formulas were used to calculate (1) the overall confidence score and (2) the coordinates of the respective fused bounding box over all **T** bounding box detections that belong to a specific cluster of detections at **L**[$pos$] (adapted from [22]):

$$\mathbf{C}_{\text{fused}}[pos] = \frac{\sum_{i=1}^{\mathbf{T}} \mathbf{C}_i}{T}, \tag{4}$$

$$\mathbf{F}_{xy}[pos] = \frac{\sum_{i=1}^{\mathbf{T}} (\mathbf{C}_i * \mathbf{V}[i])}{\sum_{i=1}^{\mathbf{T}} \mathbf{C}_i}, \tag{5}$$

where **V** holds the vertex point coordinates of all detections at position $pos$ in list **L**. Hence, **V** holds the coordinates of the detections that form a cluster, which we want to fuse into one box (i.e., $\mathbf{F}_{xy}[pos]$). Finally, this fused box and its confidence score $\mathbf{C}_{\text{fused}}[pos]$ are added to the list **F** at position $pos$, i.e., **F**[$pos$].

As a last step, the fused bounding boxes and the respective input images are visualized. The confidence score (within the interval [0,1]) can be mapped to a diverging color map (red-yellow-green) and is plotted below each bounding box detection. Bounding boxes receive the same color as the scores, but are only plotted if the fused score is $> 0.5$. For detections with lower confidence, only a polygon shape with transparent fill (red) is drawn.

## V. Experiments

During our project, we encountered unexpected technical issues with the current setup and code base of the S$^2$A-Net used in the RealScore project. Extensive debugging was unavoidable. Thus, the hyper parameter tuning of our prototype remained incomplete. Currently, we observe a training loss plateau after epoch 12. Despite this, we present our experiments to give a glimpse of the results that could be obtained using a properly trained model. All experiments were conducted with an ensemble size = 30.

### A. Class-wise Average Precision

As expected, most classes yielded low APs (i.e., 85% of the mean APs were $< 0.5$; cf. Table I). However, there were notable exceptions such as *noteheadBlackOnLine*, *noteheadBlackInSpace*, *restWhole*, *clefG* or *clefF*. These classes had a mean AP $> 0.8$ across ensemble members. Moreover, often

the minimal AP calculated for one of these classes was high too (i.e. $> 0.7$). In total, 14 classes exhibited a mean AP $> 0.6$ (Table I, highlighted in green).

In general, we observed relatively high ensemble diversity: There existed a considerable mean range of 0.33 between minimum and maximum class-wise APs across ensemble members (excluding classes for which min and max were both 0).

### B. Visualization

The visualization using transparent polygons (cf. section IV-C1) did not allow for a quick identification of low confidence predictions (Appendix A, Fig. A1). It became obvious that this issue would persist even if the transparency levels were mapped to a heat map with different colors.

In contrast, on the visualizations produced by the combination of NMS and WBF (cf. section IV-C2) low confidence predictions stand out immediately. Due to the insufficiently trained model, high confidence predictions are scarce and sometimes difficult to spot. Hence, we report filtered and unfiltered visualization results for (1) an easier example (Fig. 6), where the model seems to provide decent predictions and (2) a more difficult example (Fig. 7), where the model's predictions are less accurate.

The easier example (Fig. 6) shows that note heads and clefs are detected reasonably well. Line shaped objects, such as stems, are much more challenging for our model to detect. The same observations are true for the more difficult example (Fig. 7). Multiple low confidence detections are shown e.g. for the stem class because the IoU of these line shaped detections is extremely sensitive to the rotation angle, i.e. a high IoU results for perfectly aligned detections only (consequently, these bounding boxes are not fused). Moreover, note that very low confidence scores are possible for the unfiltered results (i.e., Fig. 6 a) and Fig. 7 a)). This is a result of the calculation of the fused confidence score. If e.g. only one member predicts a detection for a certain object with a score = 0.3, this results in a fused score of 0.3/30 = 0.01 with an ensemble of size = 30 as was used here.

## VI. Discussion

First, we discuss our experimental results and how our visualization method could be further developed. Afterwards, we dedicate a separate subsection to the discussion of potential improvements to our current prototype.

### A. Experimental Results and Confidence Visualization

The class-wise APs offer only a basic assessment of NN predictive confidence. However, in our case of a sub-optimally trained model, these APs provide a quick sanity check and are an important tool for overall model evaluation during hyperparameter tuning. Once a fully trained model is available, also the calibration of the predictions should be examined and the diversity of ensemble member predictions should be investigated more comprehensively [5].

TABLE I
Class-wise Average Precision (AP)

| class_label | mean | 0.25 | 0.75 | min | max | nr_occur |
|---|---|---|---|---|---|---|
| stem | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 36650.0 |
| noteheadBlackOnLine | 0.87 | 0.86 | 0.87 | 0.83 | 0.9 | 19482.0 |
| noteheadBlackInSpace | 0.85 | 0.84 | 0.86 | 0.8 | 0.88 | 18769.0 |
| ledgerLine | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 12741.0 |
| beam | 0.14 | 0.13 | 0.16 | 0.07 | 0.19 | 10637.0 |
| augmentationDot | 0.14 | 0.12 | 0.16 | 0.07 | 0.19 | 3261.0 |
| staff | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2165.0 |
| keySharp | 0.32 | 0.27 | 0.36 | 0.18 | 0.48 | 2018.0 |
| restWhole | 0.88 | 0.88 | 0.9 | 0.78 | 0.91 | 1814.0 |
| noteheadHalfOnLine | 0.62 | 0.55 | 0.7 | 0.26 | 0.85 | 1690.0 |
| tie | 0.08 | 0.08 | 0.1 | 0.0 | 0.11 | 1636.0 |
| keyFlat | 0.66 | 0.63 | 0.74 | 0.0 | 0.81 | 1626.0 |
| noteheadHalfInSpace | 0.35 | 0.27 | 0.45 | 0.07 | 0.64 | 1612.0 |
| slur | 0.23 | 0.22 | 0.25 | 0.16 | 0.28 | 1528.0 |
| rest8th | 0.43 | 0.39 | 0.49 | 0.14 | 0.55 | 1437.0 |
| flag8thDown | 0.08 | 0.07 | 0.09 | 0.06 | 0.13 | 1352.0 |
| restQuarter | 0.24 | 0.18 | 0.31 | 0.1 | 0.4 | 1313.0 |
| clefG | 0.94 | 0.96 | 0.98 | 0.06 | 0.99 | 1256.0 |
| accidentalSharp | 0.26 | 0.24 | 0.27 | 0.19 | 0.31 | 1249.0 |
| accidentalNatural | 0.07 | 0.05 | 0.08 | 0.0 | 0.11 | 1187.0 |
| flag8thUp | 0.11 | 0.1 | 0.13 | 0.07 | 0.16 | 1163.0 |
| clefF | 0.98 | 0.99 | 0.99 | 0.86 | 0.99 | 830.0 |
| timeSig4 | 0.85 | 0.9 | 0.93 | 0.0 | 0.95 | 805.0 |
| dynamicF | 0.16 | 0.14 | 0.18 | 0.11 | 0.24 | 802.0 |
| articStaccatoAbove | 0.05 | 0.04 | 0.06 | 0.0 | 0.1 | 750.0 |
| accidentalFlat | 0.23 | 0.2 | 0.28 | 0.0 | 0.39 | 728.0 |
| dynamicP | 0.81 | 0.8 | 0.83 | 0.76 | 0.87 | 646.0 |
| repeatDot | 0.01 | 0.01 | 0.02 | 0.0 | 0.05 | 568.0 |
| noteheadWholeInSpace | 0.3 | 0.21 | 0.36 | 0.05 | 0.63 | 544.0 |
| noteheadWholeOnLine | 0.43 | 0.4 | 0.49 | 0.17 | 0.56 | 470.0 |
| rest16th | 0.19 | 0.1 | 0.26 | 0.0 | 0.6 | 428.0 |
| restHalf | 0.31 | 0.23 | 0.4 | 0.0 | 0.52 | 398.0 |
| dynamicM | 0.47 | 0.44 | 0.52 | 0.0 | 0.64 | 367.0 |
| articStaccatoBelow | 0.01 | 0.0 | 0.01 | 0.0 | 0.04 | 271.0 |
| articAccentAbove | 0.56 | 0.51 | 0.66 | 0.06 | 0.74 | 222.0 |
| tuplet6 | 0.0 | 0.0 | 0.01 | 0.0 | 0.01 | 214.0 |
| clef8 | 0.09 | 0.03 | 0.13 | 0.0 | 0.23 | 202.0 |
| timeSig3 | 0.01 | 0.0 | 0.01 | 0.0 | 0.12 | 194.0 |
| timeSig8 | 0.11 | 0.05 | 0.14 | 0.0 | 0.32 | 187.0 |
| dynamicCrescendoHairpin | 0.24 | 0.2 | 0.29 | 0.1 | 0.43 | 162.0 |
| flag16thUp | 0.14 | 0.13 | 0.18 | 0.0 | 0.24 | 145.0 |
| tuplet3 | 0.33 | 0.3 | 0.35 | 0.02 | 0.57 | 136.0 |
| clefCAlto | 0.18 | 0.17 | 0.22 | 0.0 | 0.27 | 133.0 |
| fingering1 | 0.14 | 0.06 | 0.2 | 0.0 | 0.33 | 130.0 |
| flag16thDown | 0.02 | 0.01 | 0.03 | 0.0 | 0.04 | 116.0 |
| dynamicDiminuendoHairpin | 0.06 | 0.0 | 0.13 | 0.0 | 0.26 | 111.0 |
| articAccentBelow | 0.64 | 0.55 | 0.85 | 0.06 | 0.94 | 107.0 |
| fermataAbove | 0.88 | 0.88 | 0.9 | 0.71 | 0.92 | 107.0 |
| clefCTenor | 0.48 | 0.41 | 0.53 | 0.34 | 0.63 | 102.0 |
| timeSig1 | 0.02 | 0.0 | 0.02 | 0.0 | 0.09 | 95.0 |
| caesura | 0.1 | 0.0 | 0.11 | 0.0 | 0.61 | 86.0 |
| articStaccatissimoBelow | 0.0 | 0.0 | 0.0 | 0.0 | 0.04 | 85.0 |
| fingering2 | 0.01 | 0.0 | 0.0 | 0.0 | 0.09 | 85.0 |
| fingering3 | 0.02 | 0.0 | 0.03 | 0.0 | 0.15 | 83.0 |
| accidentalDoubleSharp | 0.04 | 0.0 | 0.08 | 0.0 | 0.17 | 80.0 |
| fingering4 | 0.03 | 0.0 | 0.03 | 0.0 | 0.21 | 74.0 |
| keyNatural | 0.13 | 0.08 | 0.17 | 0.0 | 0.27 | 73.0 |
| timeSig5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.06 | 72.0 |
| timeSig6 | 0.15 | 0.01 | 0.27 | 0.0 | 0.56 | 71.0 |
| timeSig2 | 0.04 | 0.01 | 0.06 | 0.0 | 0.09 | 70.0 |
| fingering0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.01 | 70.0 |
| stringsDownBow | 0.04 | 0.01 | 0.04 | 0.0 | 0.26 | 63.0 |
| keyboardPedalUp | 0.25 | 0.05 | 0.41 | 0.0 | 0.51 | 58.0 |
| stringsUpBow | 0.74 | 0.67 | 0.9 | 0.15 | 0.98 | 55.0 |
| timeSigCommon | 0.07 | 0.04 | 0.11 | 0.0 | 0.18 | 55.0 |
| dynamicS | 0.02 | 0.0 | 0.03 | 0.0 | 0.08 | 54.0 |
| noteheadDoubleWholeOnLine | 0.15 | 0.09 | 0.22 | 0.0 | 0.3 | 54.0 |
| articStaccatissimoAbove | 0.0 | 0.0 | 0.0 | 0.0 | 0.02 | 53.0 |
| ornamentMordent | 0.08 | 0.02 | 0.12 | 0.0 | 0.25 | 52.0 |
| articTenutoAbove | 0.01 | 0.0 | 0.01 | 0.0 | 0.06 | 48.0 |
| articMarcatoBelow | 0.27 | 0.11 | 0.44 | 0.0 | 0.62 | 46.0 |
| timeSig9 | 0.31 | 0.22 | 0.36 | 0.0 | 0.7 | 46.0 |
| keyboardPedalPed | 0.11 | 0.06 | 0.16 | 0.0 | 0.2 | 45.0 |
| ornamentTurn | 0.42 | 0.37 | 0.51 | 0.05 | 0.76 | 43.0 |
| articMarcatoAbove | 0.25 | 0.12 | 0.37 | 0.0 | 0.63 | 42.0 |
| restDoubleWhole | 0.27 | 0.16 | 0.38 | 0.01 | 0.48 | 40.0 |
| rest32nd | 0.01 | 0.0 | 0.03 | 0.0 | 0.04 | 39.0 |
| timeSig7 | 0.08 | 0.01 | 0.13 | 0.0 | 0.21 | 38.0 |
| segno | 0.88 | 0.93 | 0.97 | 0.0 | 0.97 | 36.0 |
| arpeggiato | 0.01 | 0.0 | 0.02 | 0.0 | 0.06 | 36.0 |
| fermataBelow | 0.28 | 0.19 | 0.4 | 0.0 | 0.51 | 29.0 |
| timeSig0 | 0.35 | 0.25 | 0.43 | 0.0 | 0.97 | 28.0 |
| coda | 0.95 | 0.97 | 1.0 | 0.0 | 1.0 | 27.0 |
| clef15 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 25.0 |
| ornamentTrill | 0.1 | 0.05 | 0.15 | 0.0 | 0.25 | 20.0 |
| dynamicZ | 0.33 | 0.15 | 0.53 | 0.0 | 0.85 | 20.0 |
| noteheadDoubleWholeInSpace | 0.02 | 0.0 | 0.01 | 0.0 | 0.22 | 18.0 |
| flag32ndUp | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 17.0 |
| ornamentTurnInverted | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 14.0 |
| timeSigCutCommon | 0.49 | 0.43 | 0.68 | 0.0 | 0.78 | 14.0 |
| tupletBracket | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 13.0 |
| rest64th | 0.04 | 0.0 | 0.08 | 0.0 | 0.18 | 10.0 |
| flag32ndDown | 0.01 | 0.0 | 0.0 | 0.0 | 0.16 | 8.0 |
| rest128th | 0.08 | 0.0 | 0.04 | 0.0 | 0.57 | 7.0 |
| flag128thUp | 0.04 | 0.0 | 0.08 | 0.0 | 0.22 | 6.0 |
| dynamicR | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.0 |
| flag64thUp | 0.0 | 0.0 | 0.0 | 0.0 | 0.03 | 4.0 |
| flag64thDown | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 |

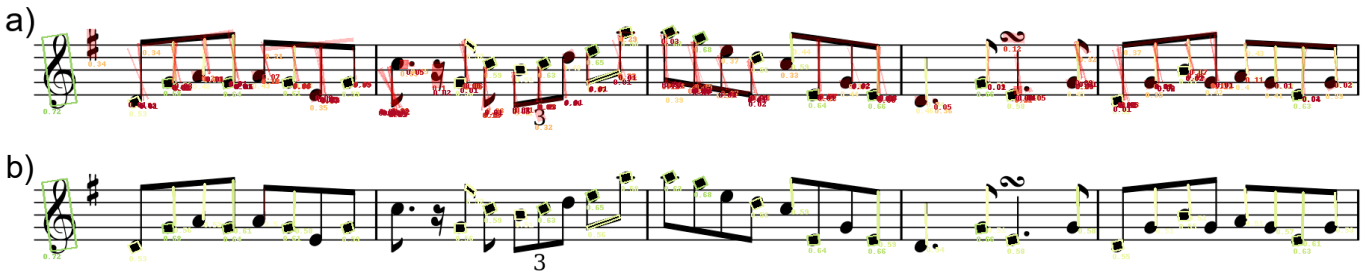


Fig. 6. An easier example where the prototype manages to propose quite decent detections. a) IoU-threshold = 0.1, fused score threshold = 0.00001, skip threshold = 0.00001; b) IoU-threshold = 0.1, fused score threshold = 0.1, skip threshold = 0.1. Image-ID: lg-110143839-aug-gutenberg1939-.

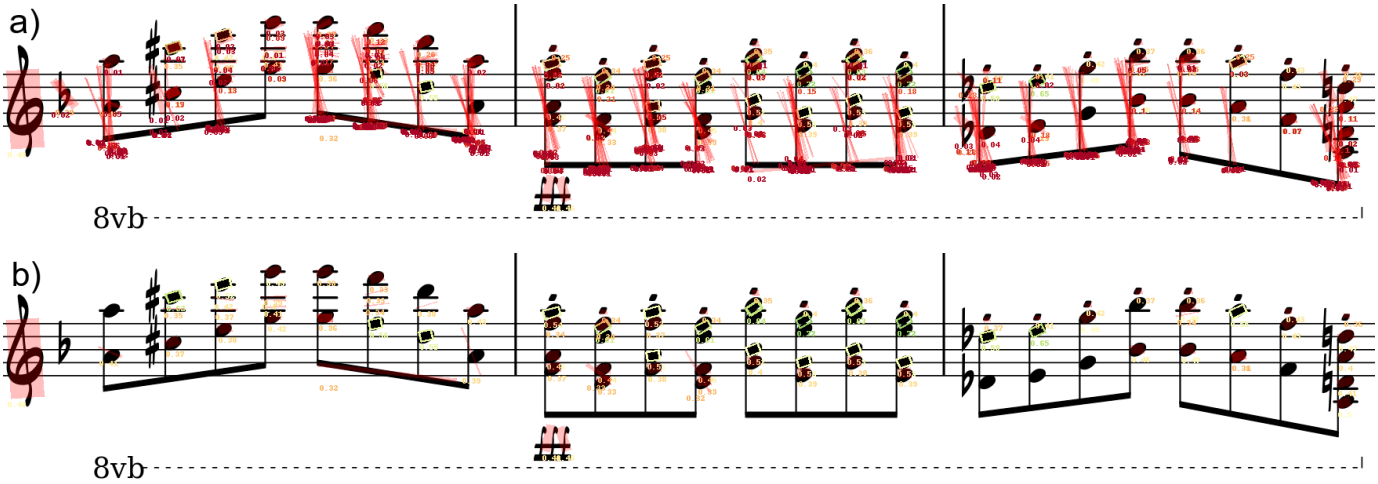


Fig. 7. A more difficult example where the prototype struggles to propose accurate detections. a) IoU-threshold = 0.1, fused score threshold 0.00001 = 0, skip threshold = 0.00001; b) IoU-threshold = 0.1, fused score threshold = 0.1, skip threshold = 0.1. Image-ID: lg-210359136-aug-lilyjazz–page-14.

The visualization using a combination of NMS and WBF provides a means of quickly identifying low confidence detections. This would be even more obvious in the case of a well performing model, as most bounding boxes would be displayed in dark-green and only a few red polygon shapes would stand out immediately. However, there are several improvements that can be considered regarding the visualization:

1) The font size of the plotted score values could be scaled relatively to the score. For example, low confidence scores (below a user-defined threshold) could be displayed in a larger font size to further highlight them.

2) The confidence score standard deviation across the detections in a cluster ($\mathbf{L}[pos]$) could be added to the plot. Depending on the informativeness of the standard deviation, the scores could also be replaced by this measure.

3) Alternative confidence measures such as nonconformity measures might be considered should the current scores fail to capture predictive uncertainty on unseen, real data [44]–[46].

### *B. Potential Improvements to the S$^2$A-Net BatchEnsemble Prototype*

Our prototype is sub-optimally trained only. Experimenting with different data augmentation pipelines, learning rate schedules and other hyper parameter settings seems the first approach to remedy this. However, hyperparameter tuning using a single GPU turned out to be limited by the considerable memory overhead of our prototype. Multiple times we ran into memory overflow issues when trying to replicate training- or test-time data augmentation pipelines that had been used for training/testing the single model version of the S$^2$A-Net. Therefore, debugging the distributed training routine seems to be a top priority. Moreover, no time was left for profiling the memory usage of our prototype. Catching up on this might also help to locate bottle necks and make our prototype more efficient in terms of memory costs. An other way to reduce memory costs, is offered by the Late-Phase Weights approach (cf. section II-H, [13]). However, starting ensemble training later will still require the allocation of more memory at that point in training.

If hyperparameter tuning does not lead to significantly improved performance, implementational causes for the low performance should also be considered. For example, it is possible that the shortcut taken in the implementation of the BatchEnsemble S$^2$A-Net prototype had more severe consequences for the performance of the model than anticipated. In particular, performance problems may have been caused by skipping the BatchEnsemble implementation of the ACL in the ARN sub module of the FAM, which is crucial for refining the anchors. The ARN was trained on the entire training dataset, although it was only trained for one member and distributed to the rest. Still, this ARN might have performed sub-optimally for other members. Adopting BatchEnsemble for the ACL,

however, would require a complete re-implementation of the layer, which would exceed the remainder of the time frame available for this study.

Alternative approaches to economic ensembles could also be considered in case our prototype cannot be tuned to a sufficient performance. SnapshotEnsemble in particular could provide a relatively straightforward and quick approach to generate economic ensembles, if it really is compatible with the $S^2$A-Net architecture. At least this seems to be the case on paper, as apparently only the cyclic learning rate schedule has to be implemented [20]. Probably, the resulting SnapshotEnsembles will show less diversity than our prototype. Moreover, the long training (over 500 epochs) that was necessary to train the single model $S^2$A-Net could be prohibitive. SnapshotEnsemble relies on multiple learning rate annealing cycles ([20] used cycles of 50 epochs). That is, the learning rate is lowered quickly to get the model to converge early during training. Then the learning rate is increased again to get the model out of the current local minimum. However, useful early local minima may not be found in the case of the $S^2$A-Net.

## VII. Conclusion

Our literature survey of relevant state-of-the-art economic ensemble methods allowed for the identification of BatchEnsemble as an approach that is (1) efficient in terms of memory and computational cost and (2) is compatible with the Single-shot Alignment Network [21] currently used in the RealScore project.

Although, the memory overhead of our BatchEnsemble prototype was not as low as we had hoped for, it evinced considerable diversity across ensemble member predictions. Moreover, even using the sub-optimal output of our preliminary model, the visualization results are promising. With a properly trained model, low confidence scores should be easy to spot. Combining WBF and NMS to post-process detections improves the practical value of our visualizations and could pave the way for a fully developed visualization tool for ensemble predictions.

## Acknowledgment

The author would like to thank Pascal Sager, Raphael Emberger, Adhiraj Ghosh, Lukas Tuggener and Thilo Stadelmann for their support during the project.

## References

[1] (2022, Jan.) Realscore project page. [Online]. Available: https://www.zhaw.ch/de/forschung/forschungsdatenbank/projektdetail/projektid/3005/

[2] L. Tuggener, I. Elezi, J. Schmidhuber, and T. Stadelmann, "Deep watershed detector for music object recognition," *Proceedings of the 19th ISMIR Conference, Paris, France, September 23-27, 2018*, pp. 271–278, 2018.

[3] L. Tuggener, Y. P. Satyawan, A. Pacha, J. Schmidhuber, and T. Stadelmann, "The deepscoresv2 dataset and benchmark for music object detection," in *Proceedings of the 25th International Conference on Pattern Recognition 2020 (ICPR'20)*, 2020, pp. 1–8. [Online]. Available: doi.org/10.21256/zhaw-20647

[4] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *NIPS 2017*, pp. 1–15, 2017. [Online]. Available: arxiv.org/pdf/1612.01474v3.pdf

[5] F. K. Gustafsson, M. Danelljan, and T. B. Schön, "Evaluating scalable bayesian deep learning methods for robust computer vision," pp. 1–31, 2020. [Online]. Available: arxiv.org/pdf/1906.01620v3.pdf

[6] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *International Journal of Computer Vision*, vol. 128, no. 2, pp. 261–318, 2020.

[7] C. Henning, J. von Oswald, J. Sacramento, S. C. Surace, J.-P. Pfister, and B. F. Grewe, "Approximating the predictive distribution via adversarially-trained hypernetworks," *Bayesian Deep Learning Workshop 2018*, pp. 1–11.

[8] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," *arXiv e-prints*, pp. 1–12, 2016. [Online]. Available: arxiv.org/pdf/1506.02142v6.pdf

[9] Y. Wen, D. Tran, and J. Ba, "Batchensemble: an alternative approach to efficient ensemble and lifelong learning," *arXiv e-prints*, pp. 1–20, 2020. [Online]. Available: arxiv.org/pdf/2002.06715v2.pdf

[10] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. V. Dillon, B. Lakshminarayanan, and J. Snoek, "Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift," *arXiv e-prints*, pp. 1–25, 2019. [Online]. Available: arxiv.org/pdf/1906.02530v2.pdf

[11] S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall, and D. Batra, "Why m heads are better than one: Training a diverse ensemble of deep networks."

[12] J. Sacramento, J. von Oswald, S. Kobayashi, C. Henning, and B. F. Grewe, "Economical ensembles with hypernetworks," *arXiv e-prints*, pp. 1–25, 2020. [Online]. Available: arxiv.org/abs/2007.12927.pdf

[13] J. von Oswald, S. Kobayashi, J. Sacramento, A. Meulemans, C. Henning, and B. F. Grewe, "Neural networks with late-phase weights," *arXiv e-prints*, 2021. [Online]. Available: arxiv.org/pdf/2007.12927v3.pdf

[14] S. Fort, H. Hu, and B. Lakshminarayanan, "Deep ensembles: A loss landscape perspective," *arXiv e-prints*, pp. 1–15, 2020. [Online]. Available: arxiv.org/pdf/1912.02757v2.pdf

[15] T. Garipov, P. Izmailov, D. Podoprikhin, D. P. Vetrov, and A. G. Wilson, "Loss surfaces, mode connectivity, and fast ensembling of dnns," *arXiv e-prints*, pp. 1–13, 2018. [Online]. Available: arxiv.org/pdf/1802.10026v1.pdf

[16] U. Asif, J. Tang, and S. Harrer, "Ensemble knowledge distillation for learning improved and efficient networks," *arXiv e-prints*, pp. 1–8, 2020. [Online]. Available: arxiv.org/pdf/1909.08097v3.pdf

[17] F. Wenzel, J. Snoek, D. Tran, and R. Jenatton, "Hyperparameter ensembles for robustness and uncertainty quantification," *arXiv e-prints*, pp. 1–31, 2021. [Online]. Available: arxiv.org/pdf/2006.13570v3.pdf

[18] Y. Xia, J. Zhang, T. Jiang, Z. Gong, W. Yao, and L. Feng, "Hatchensemble: an efficient and practical uncertainty quantification method for deep neural networks," *Complex & Intelligent Systems*, vol. 521, no. 7553, p. 436, 2021.

[19] N. Durasov, T. Bagautdinov, P. Baque, and P. Fua, "Masksembles for uncertainty estimation," pp. 1–12, 2021. [Online]. Available: arxiv.org/pdf/2012.08334v2.pdf

[20] G. Huang, Y. Li, and G. Pleiss, "Snapshot ensembles: Train 1, get m for free," *arXiv e-prints*, pp. 1–14, 2017. [Online]. Available: arxiv.org/pdf/1704.00109v1.pdf

[21] J. Han, J. Ding, J. Li, and G.-S. Xia, "Align deep features for oriented object detection," *arXiv e-prints*, pp. 1–10, 2021. [Online]. Available: arxiv.org/pdf/2008.09397v3.pdf

[22] R. Solovyev, W. Wang, and T. Gabruseva, "Weighted boxes fusion: Ensembling boxes from different object detection models," *arXiv e-prints*, pp. 1–9, 2021. [Online]. Available: arxiv.org/pdf/1910.13302v3.pdf

[23] K. Kamnitsas, W. Bai, E. Ferrante, S. McDonagh, M. Sinclair, N. Pawlowski, M. Rajchl, M. Lee, B. Kainz, D. Rueckert, and B. Glocker, "Ensembles of multiple models and architectures for robust brain tumour segmentation," *arXiv e-prints*, pp. 1–12, 2017. [Online]. Available: arxiv.org/pdf/1711.01468v1.pdf

[24] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," *arXiv e-prints*, pp. 1–29, 2016. [Online]. Available: arxiv.org/pdf/1609.09106v4.pdf

[25] D. Krueger, C.-W. Huang, R. Islam, R. Turner, A. Lacoste, and A. Courville, "Bayesian hypernetworks," *arXiv e-prints*, pp. 1–13, 2018. [Online]. Available: arxiv.org/pdf/1710.04759v2.pdf

[26] N. Pawlowski, A. Brock, M. C. Lee, M. Rajchl, and B. Glocker, "Implicit weight uncertainty in neural networks," *arXiv e-prints*, pp. 1–15, 2018. [Online]. Available: arxiv.org/pdf/1711.01297v2.pdf

[27] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv e-prints*, pp. 1–11, 2015. [Online]. Available: arxiv.org/pdf/1502.03167v3.pdf

[28] P. K. Mudrarkarta, M. Sandler, A. Zhmoginov, and A. Howard, "K for the price of 1: parameter-efficient multi-task and transfer learning," *arXiv e-prints*, pp. 1–15, 2019. [Online]. Available: arxiv.org/pdf/1810.10703v2.pdf
[29] J. Frankle, D. J. Schwab, and A. S. Morcos, "Training batchnorm and only batchnorm: on the expressive power of random features in cnns," *arXiv e-prints*, pp. 1–28, 2021. [Online]. Available: arxiv.org/pdf/2003.00152v3.pdf
[30] I. Loshchilov and F. Hutter, "Sgdr: Stocastic gradient descent with restarts," *arXiv e-prints*, pp. 1–9, 2016. [Online]. Available: arxiv.org/pdf/1608.03983v2.pdf
[31] L. N. Smith, "Cyclical learning rates for training neural networks," *arXiv e-prints*, pp. 1–10, 2017. [Online]. Available: arxiv.org/pdf/1506.01186v6.pdf
[32] (2022, Jan.) $S^2$a-net github branch dev_tugg. [Online]. Available: https://github.com/raember/s2anet/tree/dev_tugg
[33] (2022, Jan.) Mmdetection project github. [Online]. Available: https://github.com/open-mmlab/mmdetection
[34] (2022, Jan.) Oriented bounding box annotations github. [Online]. Available: https://github.com/yvan674/obb_anns
[35] (2022, Jan.) Mmcv github branch tuggeluk. [Online]. Available: https://github.com/tuggeluk/mmcv
[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv e-prints*, pp. 1–12, 2015. [Online]. Available: arxiv.org/pdf/1512.03385v1.pdf
[37] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," *arXiv e-prints*, pp. 1–10, 2017. [Online]. Available: arxiv.org/pdf/1612.03144v2.pdf
[38] A. Neubeck and L. van Gool, "Efficient non-maximum suppression," in *18th International Conference on Pattern Recognition, 2006*, Y. Y. Tang, Ed. Los Alamitos, Calif.: IEEE Computer Society, 2006, pp. 850–855.
[39] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv e-prints*, pp. 1–14, 2016. [Online]. Available: arxiv.org/pdf/1607.06450v1.pdf
[40] G. Franchi, A. Bursuc, E. Aldea, S. Dubuisson, and I. Bloch, "Encoding the latent posterior of bayesian neural networks for uncertainty quantification," *arXiv e-prints*, pp. 1–16, 2021. [Online]. Available: arxiv.prg/pdf/2012.02818v2.pdf
[41] (2022, Jan.) Batchensemble ensemble_conv2d()-layer github. [Online]. Available: https://github.com/giannifranchi/LP_BNN/blob/main/networks/batchensemble_layers.py
[42] (2022, Jan.) $S^2$a-net batchensemble prototype github branch dev_sage. [Online]. Available: https://github.com/raember/s2anet/tree/dev-sage
[43] (2022, Jan.) Weighted-boxes-fusion github. [Online]. Available: https://github.com/ZFTurbo/Weighted-Boxes-Fusion
[44] H. Linusson, "Nonconformity measures and ensemble strategies: An analysis of conformal predictor efficiency and validity," Dissertation, Stockholm University, 2021. [Online]. Available: https://www.dissertations.se/dissertation/fb4c5a70b3/
[45] T. Pereira, S. Cardoso, D. Silva, M. Guerreiro, A. de Mendonça, and S. C. Madeira, "Ensemble learning with conformal predictors: Targeting credible predictions of conversion from mild cognitive impairment to alzheimer's disease," *arXiv e-prints*, pp. 1–4, 2018. [Online]. Available: arxiv.org/pdf/1807.01619v2.pdf
[46] V. Vovk, "The basic conformal prediction framework," in *Conformal Prediction for Reliable Machine Learning*, V. Balasubramanian, Ed. San Francisco: Elsevier Science & Technology, 2014, pp. 3–19.

## Appendix A

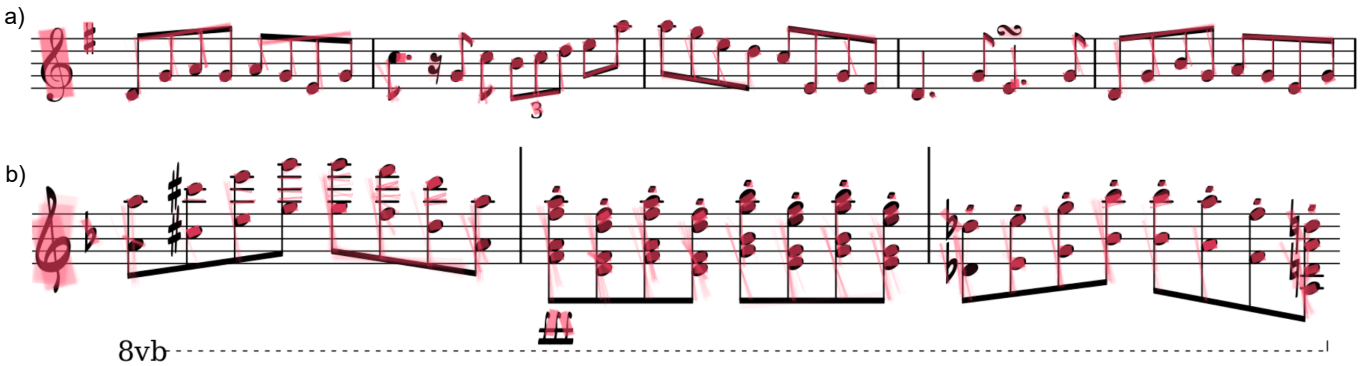


Fig. A1. Plotting all detections of all ensemble members on one image using an alpha level = $1/m$, where $m$ is the ensemble size. Detections were post-processed using NMS with: Score threshold = 0.3, IoU-threshold = 0.1.
a) Image-ID: lg-110143839-aug-gutenberg1939-.
b) Image-ID: lg-210359136-aug-lilyjazz–page-14.

# Selbständigkeitserklärung

Mit der Abgabe dieser Abschlussarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat (Bei Teamarbeiten gelten die Leistungen der übrigen Teammitglieder nicht als fremde Hilfe).
Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Abschlussarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Ort, Datum: Winterthur, 31.1.2022 Unterschrift Studierende/r: .....