ML-Decoder: Scalable and Versatile Classification Head

Tal Ridnik* Gilad Sharir* Avi Ben-Cohen Emanuel Ben-Baruch Asaf Noy
DAMO Academy, Alibaba Group

Abstract

In this paper, we introduce ML-Decoder, a new attention-based classification head. ML-Decoder predicts the existence of class labels via queries, and enables better utilization of spatial data compared to global average pooling. By redesigning the decoder architecture, and using a novel group-decoding scheme, ML-Decoder is highly efficient, and can scale well to thousands of classes. Compared to using a larger backbone, ML-Decoder consistently provides a better speed-accuracy trade-off. ML-Decoder is also versatile - it can be used as a drop-in replacement for various classification heads, and generalize to unseen classes when operated with word queries. Novel query augmentations further improve its generalization ability. Using ML-Decoder, we achieve state-of-the-art results on several classification tasks: on MS-COCO multi-label, we reach 91.1% mAP; on NUS-WIDE zero-shot, we reach 31.1% ZSL mAP; and on ImageNet single-label, we reach with vanilla ResNet50 backbone a new top score of 80.7%, without extra data or distillation. Public code will be available.

1. Introduction

Image classification is a vital computer-vision task, that requires assigning a label or multiple labels to an image, according to the objects present in it. With single-label classification [38, 44], we assume that the image contains only one object, hence we can apply a softmax operation on the output logits. However, natural images usually contain multiple objects and concepts, highlighting the importance of multi-label classification [42, 34], where we predict each class separately and independently, in a similar fashion to multi-task problems [5, 32]. Notable success in the field of multi-label classification was reported by exploiting label correlation via graph neural networks [8, 7], and improving loss functions, pretrain methods and backbones [2, 30, 31, 1].

In a regime of extreme classification [45, 25], we need to predict the existence of a large number of classes (usually thousands or more), forcing our model and training scheme to be efficient and scalable. Multi-label zero-shot learning (ZSL) [41, 36] is an extension of multi-label classification, where during inference the network tries to recognize unseen labels, i.e., labels from additional categories that were not used during training. This is usually done by sharing knowledge between the seen classes (that were used for training) and the unseen classes via a text model [15, 28].

Classification networks usually contain a backbone, and a classification head [31, 33, 16]. The backbone outputs a spatial embedding tensor, and the classification head transforms the spatial embeddings into prediction logits. In single-label classification, this is commonly done by global-average-pooling (GAP) , followed by a fully connected layer [14]. GAP-based heads are also used also for multi-label classification [8, 40, 11]. However, the need to identify several objects, with different locations and sizes, can make the usage of average pooling sub-optimal. Recently, several works proposed attention-based heads for multi-label
classification. [13] offered a two-stream attention framework to recognize multi-category objects from global image to local regions. [47] suggested simple spatial attention scores, and then combined them with class-agnostic average pooling features. [23] presented a pooling transformer with learnable queries for multi-label classification, achieving top results.

GAP-based classification heads are simple and efficient, and scale well with the number of classes, since they have a fixed spatial pooling cost. However, they provide suboptimal results, and are not directly applicable to ZSL. Attention-based classification heads do improve results, but are often costly, even for datasets with a small number of classes, and practically infeasible to extreme classification scenarios. They also have no natural extension to ZSL.

In this paper, we introduce a new classification head, called ML-Decoder, that provides a unified solution for single-label, multi-label, and zero-shot classification, with state-of-the-art results (see Figure 1). ML-Decoder design is based on the original transformer-decoder [35], with two major modifications, that significantly improve its scalability and efficiency. First, it reduces the quadratic dependence of the decoder in the number of input queries to a linear one, by removing the redundant self-attention block. Second, ML-Decoder uses a novel group-decoding scheme, where instead of assigning a query per class, it uses a fixed number of queries, that are interpolated to the final number of classes via a new architectural block called group fully-connected. Using group-decoding, ML-Decoder also enjoys a fixed spatial pooling cost, and scales well to thousands of classes.

ML-Decoder is flexible and efficient. It can be trained equally well with learnable or fixed queries, and can use different queries during training and inference (see Figure 2). These key features make ML-Decoder suitable for ZSL tasks. When we assign a query per class and train ML-decoder with word queries, it generalizes well to unseen queries, and significantly improves previous state-of-the-art ZSL results. We also show that the group-decoding scheme can be extended to the ZSL scenario, and introduce novel query augmentations during training to further encourage generalization.

The paper’s contributions can be summarized as follows:

- We propose a new classification head called ML-Decoder, which provides a unified solution for multi-label, zero-shot, and single-label classification, with state-of-the-art results.
- ML-Decoder can be used as a drop-in replacement for global average pooling. It is simple and efficient, and provides improved speed-accuracy trade-off compared to larger backbones, or other attention-based heads.
- ML-Decoder novel design makes it scalable to classification with thousands of classes. Complementary query-augmentation technique improves its generalizability to unseen classes as well.
- We verify the effectiveness of the ML-Decoder with comprehensive experiments on commonly-used classification datasets: MS-COCO, Open Images, NUS-WIDE, PASCAL-VOC, and ImageNet.

2. Method

In this section, we will first review the baseline classification heads. Then we will present our novel ML-Decoder, discuss its advantages, and show its applicability to several computer-vision tasks, such as multi-label, ZSL, and single-label classification.
2.1. Baseline Classification Heads

A typical classification network is comprised of a backbone, and a classification head. The network’s backbone outputs a spatial embedding tensor, $E \in \mathbb{R}^{H \times W \times D}$, and the classification head transforms the spatial embeddings tensor into $N$ logits, $\{l_i\}_{i=1}^{N}$, where $N$ is the number of classes. There are two baseline approaches for processing the spatial embeddings: GAP-based, and attention-based.

**GAP-based:** with a GAP-based classification head, we first reduce the spatial embeddings to a one-dimensional vector via simple global averaging operation on the spatial dimensions, outputting a vector $z \in \mathbb{R}^{D \times 1}$. Then, a fully connected layer transforms the embedding vector into $N$ output logits: $1 = Wz$, where $W \in \mathbb{R}^{N \times D}$ is a learnable linear projections matrix. GAP is commonly used for single-label classification tasks [31, 33, 16], and has some generalizations, for example [29, 20]. GAP was also adopted as a baseline approach for multi-label classification [2, 37, 24]

**Attention-based:** Unlike single-label classification, in multi-label classification several objects can appear in the image, in different locations and sizes. Several works [23, 13, 47] have noticed that the GAP operation, which eliminates the spatial dimension via simple averaging, can be sub-optimal for identifying multiple objects with different sizes. Instead they suggested using attention-based classification heads, which enable more elaborate usage of the spatial data, with improved results.

2.2. Recap - Attention and Transformer-Decoder

Among the attention-based classification heads proposed, a simple approach based on a transformer-decoder, similar to the one used by DETR for object detection [4], has achieved top results on multi-label classification [23].

A transformer-decoder unit relies on the multi-head attention module, introduced in [35]. A multi-head attention has three inputs: $Q, K, V$. If we define the attention operation to be:

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{d_k}})V$$ (1)

A multi-head module output is:

$$\text{MultiHeadAttn}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_d)W^O$$

where $\text{head}_i = \text{Attention}(QW^Q_i, KW^K_i, VW^V_i)$

$W^Q, W^K, W^V, W^O$ are learnable projections matrices.

An illustration of a transformer-decoder classification head is given in Figure 3 (left side). The transformer-decoder has two inputs: The spatial embedding tensor, $E$, and a set of $N$ learnable queries, $Q$, one for each class.

![Figure 3. Scalability - baseline transformer-decoder vs. our proposed ML-Decoder.](image)

The transformer-decoder processes the inputs via four consecutive stages called self-attention, cross-attention, feed-forward and token-pool:

- **self-attn:** $Q_1 \leftarrow \text{MultiHeadAttn}(Q, Q, Q)$
- **cross-attn:** $Q_2 \leftarrow \text{MultiHeadAttn}(Q_1, E, E)$
- **feed-forward:** $Q_3 \leftarrow \text{FF}(Q_2)$
- **token-pool:** $\text{Logits} \leftarrow \text{Pool}(Q_3)$ (2)

FF is a feed-forward fully connected layer, as defined in [35]. The token pooling stage is a simple pooling on the token embeddings’ dimension $D$, to produce $N$ output logits.

2.3. ML-Decoder

2.3.1 Motivation

On multi-label datasets with small number of classes, such as MS-COCO [22] and Pascal-VOC [12] (80 and 20 classes respectively), transformer-decoder classification head works well, and achieves state-of-the-art results [23], with small additional computational overhead. However, it suffers from a critical drawback - the computational cost is quadratic with the number of classes. Hence, for datasets with large number of classes, such as Open Images [19] (9600 classes), using transformer-decoder is practically infeasible in terms of computational cost, as we will show in
Section 3.2. For real-world applications, a large number of classes is imperative to provide a complete and comprehensive description of an input image. Hence, a more scalable and efficient attention-based classification head is needed. In addition, transformer-decoder as a classification head is suitable for multi-label classification only. A more general attention-based head, that can also address other tasks such as single-label and ZSL, will be beneficial.

2.3.2 ML-Decoder Design

We will now describe our proposed classification head, ML-Decoder. Illustration of ML-Decoder flow is given in Figure 3 (right-side). Compared to transformer-decoder, ML-Decoder includes the following modifications:

(1) Self-attention removal: We start by observing that during inference, the self-attention module of transformer-decoder provides a fixed transformation on the input queries. However, as the queries are entering the cross-attention module, they are subjected to a projection layer, before going through the attention operation (Eq. 1). In practice, the projection layer can transform the queries to any desired output, making the self-attention module redundant. Hence, we can remove the self-attention layer, while still maintaining the same expressivity of the classification head, and without degrading the results. We will validate this empirically in Section 3.1. By removing the self-attention, we avoid a costly module, and relax the quadratic dependence of ML-Decoder in the number of input queries to a linear one, making it more practical and efficient.

(2) Group-decoding: In an extreme classification scenario, even linear dependency of the classification head with the number of classes can be costly. We want to break this coupling, and make the cross-attention module, and the feed-forward layer after it, independent of the number of classes, same as GAP operation. To this end, instead of assigning a query per class, we use as inputs a fixed number of group queries, K (see Figure 3).

After the feed-forward layer, we transform the group queries into output logits via a novel layer called group fully-connected. This layer performs simultaneously two tasks - (1) expand each group query to \( \frac{N}{K} \) outputs; (2) pool the embeddings’ dimension. If we define the group-factor to be \( g = \frac{N}{K} \), group fully-connected generates an output \( L_i \) with the following operation:

\[
L_i = (W_k \cdot Q_k)_j
\]

where: \( k = i \div g \), \( j = i \mod g \) \hspace{1cm} (3)

\( Q_k \in \mathbb{R}^D \) is the \( k^{th} \) query, and \( W_k \in \mathbb{R}^{g \times D} \) is the \( k^{th} \) learnable projection matrix. An illustration of group fully-connected layer is given in Figure 4, and a pseudo-code appears in appendix K.

The full flow of ML-Decoder with group-decoding is depicted in Eq. 4, where \( G_q \) are the input group queries:

\[
\begin{align*}
\text{cross-attn:} & \quad G_{q_1} \leftarrow \text{MultiHeadAttn}(G_q, E, E) \\
\text{feed-forward:} & \quad G_{q_2} \leftarrow \text{FF}(G_{q_1}) \\
\text{group FC:} & \quad \text{Logits} \leftarrow \text{Group-FC}(G_{q_2})
\end{align*}
\]

Some additional observations and insights into the group-decoding scheme:

- With full-decoding \( (g = 1) \), each query checks the existence of a single class. With group-decoding, each query checks the existence of several classes. We chose to divide the classes into groups in a random manner. Clustering the classes via semantic proximity is an alternative, but will require a cumbersome clustering process, with extra hyper-parameters that might need tuning per dataset. In Section 3.2 we will show that random group clustering is enough to provide results comparable to a full-decoding scheme.

- In terms of flops, the group fully-connected layer is equivalent to a fully-connected layer in a GAP-based head \( (N \times D \text{ multiplications}) \). Both are linearly dependent in the number classes, but in practice they have a small computational overhead, even for thousands of classes. In terms of memory consumption, performing the two tasks of group fully-connected together, in a single operation, is more efficient than doing them consecutively, since there is no need to store large intermediate maps.

- The only component in ML-Decoder that depends on the input image size is the cross-attention module. We can think of the cross-attention layer as doing spatial pooling, similar to GAP. With group-decoding, ML-Decoder has fixed spatial pooling cost, independent of \( N \).

(3) Non-learnable queries: [23] argued that transformer-decoder for multi-label classification achieves top results only with learnable queries. However, we observe that the queries are always fed into a multi-head attention layer, that applies a learnable projection on them (Eq. 1). Hence, setting the queries weights as learnable is redundant - a learnable projection can transform any fixed-value query to any
value obtained by a learnable query. We will validate this empirically in Section 3.1, showing that the same accuracies are obtained when training ML-Decoder with learnable or fixed queries. In addition to simplifying the training process, using fixed queries will enable us to do ZSL.

2.3.3 ML-Decoder for ZSL

Next, we will present the adaptations needed in order to use ML-Decoder in a multi-label ZSL scenario, and discuss key features of ML-Decoder that make it suitable for the task. We will also show that the group-decoding scheme can be extended to ZSL, and present novel query augmentations that further improve ML-Decoder generalizability.

**NLP-based queries:** We begin by presenting a version of ML-Decoder for ZSL with a full-decoding scheme (each label has a corresponding query). As discussed in the previous section, the input queries can be either learnable or fixed. For ZSL, we use fixed NLP-based queries - for each label, a word embedding vector is extracted using a language model, and set as the input query. We also use shared projection matrix in the group fully-connected layer (setting $W_k = W$ in Eq. 3). With NLP-based queries and a shared projection matrix, semantic information can propagate from the seen (training) classes to the unseen (test) classes during inference, enabling generalization.

**ML-Decoder features:** ML-Decoder contains several favorable features which make it well suited for ZSL. Firstly, its attention mechanism is based on dot-product similarity between vectors (Eq. 1). Since NLP word embeddings preserve this dot-product semantic similarity [15], the unseen labels are more likely to be matched with the most similar keys and values within the decoder. In addition, ML-Decoder with a shared projection matrix allows a variable number of input queries, and is not sensitive to the order of queries. This is beneficial since in ZSL we perform training and testing on different sets of classes, and therefore different sets of queries. For ZSL we train exclusively on the seen labels, and perform inference on the unseen classes, while for Generalized ZSL (GZSL), we perform inference on the union of the unseen and seen sets of labels.

**Group-decoding:** Group-decoding (with $K < N$) requires modifications in order to work in a ZSL setting. In appendix B we thoroughly detail our variant of group-decoding for ZSL.

**Query augmentations** With NLP-based queries, ML-Decoder naturally extends to the task of ZSL. Yet, we want to apply dedicated training tricks that further improve its generalizability. It is a common practice in computer-vision to apply augmentations on the input images to prevent overfitting, and improve generalizability to new unseen images. Similarly, we introduce query augmentations to encourage generalization to unseen class queries. The first augmentation is random-query, which adds additional random-valued queries to the set of input queries, and assigns a positive ground truth label signifying “random” for these added queries. The second augmentation is query-noise, where we add each batch a small random noise to the input queries. See Figure 8 in the appendix for illustration of the augmentations.

The main motivation for attention-based heads in multi-label classification was the need to identify several objects, with different locations and sizes [23]. We will show in Section 4.3 that the benefit from ML-Decoder is more general, and fully applies also to single-label problems, where the image usually contains a single object.

3. Experimental Study

In this section, we will bring ablation tests and inner comparisons for our proposed ML-Decoder classification head. First, we will test ML-Decoder with different types of input queries. Then we will compare ML-Decoder to other classification heads, such as transformer-decoder and GAP-based. Finally, we will provide an ablation study for ZSL with augmentation queries and group-decoding.

3.1. Comparing Query Types

As discussed in Section 2.3.2, due to the linear projection in the attention module, ML-Decoder retains the same expressivity when it uses learnable or fixed queries. In Table 7 in the appendix we compare results for ML-Decoder with different types of queries, on MS-COCO multi-label dataset (see appendix C for full training details on MS-COCO).

Indeed we see that learnable, fixed random, and fixed NLP-based word queries all lead to the same accuracy, 88.1% mAP, as expected. For reducing the number of learned parameters, we shall use fixed queries.
3.2. Comparing Different Classification Heads

In Table 1 we compare MS-COCO results for training with different classification heads.

<table>
<thead>
<tr>
<th>Classification Head</th>
<th>Num of Classes</th>
<th>Num of Queries</th>
<th>Flops [G]</th>
<th>mAP [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAP</td>
<td>80</td>
<td>—</td>
<td>23.0</td>
<td>87.0</td>
</tr>
<tr>
<td>Transformer-Decoder</td>
<td>80</td>
<td>80</td>
<td>24.1</td>
<td>88.1</td>
</tr>
<tr>
<td>ML-Decoder</td>
<td>80</td>
<td>20</td>
<td>23.6</td>
<td>88.0</td>
</tr>
<tr>
<td>ML-Decoder</td>
<td>80</td>
<td>80</td>
<td>23.9</td>
<td>88.1</td>
</tr>
</tbody>
</table>

Table 1: Comparison of multi-label MS-COCO mAP score for different classification heads. Architecture - TResNet-M [31].

From Table 1 we learn the following observations:

- ML-Decoder (and transformer-decoder) provide a significant improvement (more than 1% mAP), compared to GAP-based classification head.
- When using the same number of input queries (80), transformer-decoder and ML-Decoder reach the same accuracy, demonstrating that indeed the self-attention module provides a redundant transformation (see Section 2.3.2), and removing it in ML-Decoder reduces the computational cost without impacting the results.
- Using group decoding with a ratio of \( \frac{N}{K} = 4 \) has a minimal impact on the results - a reduction of only 0.1% mAP. However, since MS-COCO dataset has a small number of classes, the additional flops from an attention-based classification head are minimal, so reducing the number of queries is not essential in this case.
- In Table 2 we repeat the same comparison on Open Images multi-label dataset, which has significantly more classes - 9600 instead of 80. Full training details on Open Images are given in appendix H.

<table>
<thead>
<tr>
<th>Classification Head</th>
<th>Num of Classes</th>
<th>Num of Queries</th>
<th>Flops [G]</th>
<th>mAP [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAP</td>
<td>9600</td>
<td>—</td>
<td>5.8</td>
<td>86.0</td>
</tr>
<tr>
<td>Transformer-Decoder</td>
<td>9600</td>
<td>9600</td>
<td>178.6</td>
<td>NA</td>
</tr>
<tr>
<td>ML-Decoder (+Self-Attention)</td>
<td>9600</td>
<td>400</td>
<td>12.4</td>
<td>86.8</td>
</tr>
<tr>
<td>ML-Decoder</td>
<td>9600</td>
<td>100</td>
<td>6.3</td>
<td>86.7</td>
</tr>
<tr>
<td>ML-Decoder</td>
<td>9600</td>
<td>200</td>
<td>6.7</td>
<td>86.8</td>
</tr>
<tr>
<td>ML-Decoder</td>
<td>9600</td>
<td>400</td>
<td>7.6</td>
<td>86.8</td>
</tr>
</tbody>
</table>

Table 2: Comparison of Open Images mAP score for different classification heads. Architecture - TResNet-M.

In contrast, ML-Decoder with group-decoding increases the flops count by only 10% - 20%, while significantly improving the mAP score on this challenging extreme classification dataset, compared to GAP. We also see from Table 2 that group decoding with a ratio of \( \frac{N}{K} = 48 \) already gives the full score benefit, and further increasing the ratio to \( \frac{N}{K} = 96 \) reduces the score by only 0.1%.

In Figure 5 we compare on MS-COCO the mAP score vs. flops for GAP-based and ML-Decoder classification head, with three different architectures - TResNet-S, TResNet-M, TResNet-L (equivalent in runtime to ResNet34, ResNet50 and ResNet101 [31]).

We see from Figure 5 that using ML-decoder provides a better flops-accuracy trade-off compared to using GAP-based classification head with a larger backbone.

In addition to the flops-accuracy measurements, in Table 12 in the appendix we provide full speed-accuracy comparisons of different classification heads, measuring inference speed, training speed, maximal batch size and flops. This table can aid in future comparisons of our work for different speed-accuracy metrics.

3.3. Zero-shot Learning

This section presents an ablation study of ML-Decoder for ZSL, on NUS-WIDE dataset [9]. NUS-WIDE is the most widely used benchmark for the multi-label ZSL task, and therefore we focus on this dataset. It is comprised of 925 seen labels, and 81 unseen labels. Full training and dataset details are given in appendix F.

In Table 3 we compare the different types of query augmentations presented in Section 2.3.3. It is evident from the table that both random-query and additive noise contribute to the model’s ability to generalize to unseen classes. When applying both of them, we see an increase of 1.2% in the mAP score of the unseen classes. We also tested the impact of query augmentations on the seen
<table>
<thead>
<tr>
<th>Augmentation Type</th>
<th>mAP [%] (ZSL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>29.9</td>
</tr>
<tr>
<td>Additive noise</td>
<td>30.6</td>
</tr>
<tr>
<td>Random-query</td>
<td>30.7</td>
</tr>
<tr>
<td>Both</td>
<td>31.1</td>
</tr>
</tbody>
</table>

Table 3. Comparison of NUS-WIDE ZSL mAP scores for ML-Decoder with different query augmentations.

classes. Both on MS-COCO and on NUS-WIDE, adding query augmentations had no impact on the seen classes’ mAP score (88.1% on MS-COCO, 22.7% on NUS-WIDE). This is in agreement with our results in Section 3.1, where learnable and fixed queries provided the same results on the seen classes. The benefit from using query augmentation is by better generalization to the unseen classes.

In Table 4 we compare group-decoder to full-decoder for the ZSL task. The modifications to the group-decoding scheme for ZSL are described in appendix B.

<table>
<thead>
<tr>
<th>Classification Head</th>
<th>Num of Classes</th>
<th>Num of Queries</th>
<th>mAP [%] (ZSL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML-Decoder</td>
<td>1006</td>
<td>1006</td>
<td>31.1</td>
</tr>
<tr>
<td>ML-Decoder</td>
<td>1006</td>
<td>100</td>
<td>28.7</td>
</tr>
</tbody>
</table>

Table 4. Comparison of NUS-WIDE ZSL mAP scores for ML-Decoder with different number of input queries.

We see from the Table 4 that the group-decoding scheme works well also for the ZSL scenario, with a small decrease in the mAP scores compared to full-decoding.

3.4. Additional Tests

We validate our architecture choices with additional ablation experiments in the appendix. Layer stacking: we investigate the effect of adding layers to ML-Decoder (Table 10).

Using a transformer backbone: we repeated the measurements of Table 1 with ViT-B-16 backbone. Using a class token or GAP pooling with ViT-B-16 gave a score of 85.2 mAP, while using the ML-Decoder head improve the score by 0.7% to 85.9%.

Statistical noise from group-decoding: we repeated the measurements of Table 1 with five different seeds, and see that the average variance between different seeds was 0.05%. Hence, the additional statistical noise from choosing random groups for group-decoding is negligible.

Additional speed-accuracy comparisons: in addition to flops-accuracy measurements, we also preformed direct speed measurements. In Table 11 in the appendix we measure the inference times of the different modules of a decoder head. In Table 12 in the appendix we provide full speed-accuracy comparisons of different classification heads, measuring inference speed, training speed, maximal batch size and flops. Table 11 and Table 12 can aid in future comparisons of our work for different speed-accuracy metrics.

4. Results

In this section, we will evaluate our ML-Decoder-based solution on popular multi-label, ZSL, and single-label classification datasets, and compare results to known state-of-the-art techniques.

4.1. Multi-label Classification

4.1.1 MS-COCO

MS-COCO [22] is a commonly-used dataset to evaluate multi-label image classification. It contains 122,218 images from 80 different categories, divided to a training set of 82,081 images and a validation set of 40,137 images.

In Table 5 we compare ML-Decoder results to top known solutions from the literature. Full training details appear in appendix C.

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>Input Resolution</th>
<th>mAP [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML-GCN [8]</td>
<td>ResNet101</td>
<td>448x448</td>
<td>83.0</td>
</tr>
<tr>
<td>KSSNET [24]</td>
<td>ResNet101</td>
<td>448x448</td>
<td>83.7</td>
</tr>
<tr>
<td>SSGRL [6]</td>
<td>ResNet101</td>
<td>576x576</td>
<td>83.8</td>
</tr>
<tr>
<td>MS-CMA [43]</td>
<td>ResNet101</td>
<td>448x448</td>
<td>83.8</td>
</tr>
<tr>
<td>ASL [2]</td>
<td>ResNet101</td>
<td>448x448</td>
<td>85.0</td>
</tr>
<tr>
<td>Q2L [23]</td>
<td>TResNet-L</td>
<td>448x448</td>
<td>88.4</td>
</tr>
<tr>
<td></td>
<td>TResNet-L</td>
<td>448x448</td>
<td>89.2</td>
</tr>
<tr>
<td>ML-Decoder</td>
<td>ResNet101</td>
<td>448x448</td>
<td>87.1</td>
</tr>
<tr>
<td>ML-Decoder</td>
<td>TResNet-L</td>
<td>448x448</td>
<td>90.0</td>
</tr>
</tbody>
</table>

Table 5. State-of-the-art comparison on MS-COCO dataset. TResNet-L has equivalent runtime to ResNet101, with design tricks that lead to improved results [31].

Since previous works did not always report their computational cost, and released a reproducible code, we cannot provide a full speed-accuracy comparison to them. Still, we see that with ML-Decoder we improve results on MS-COCO dataset, with minimal additional computational cost compared to plain GAP-based solution (see also Table 1). We hope that future works will use our results as a baseline for a more complete comparison, including computational cost and accuracy. For that matter, in Table 13 in the appendix we report our results and flops count for different input resolutions. Note that with input resolution of 640, we reach with TResNet-L and new state-of-the-art result of 91.1% on MS-COCO.

4.1.2 Additional Multi-label Datasets

Pascal-VOC: In Table 14 in the appendix we present results on another popular multi-label dataset - Pascal-VOC [12].
With ML-Decoder we reach new state-of-the-art result on Pascal-VOC - 96.6% mAP.

**Open Images:** In Table 15 in the appendix we present results on an extreme classification multi-label dataset - Open Images [19], which contains 9600 classes. Also on this dataset ML-Decoder outperforms previous methods, achieving 86.8% mAP. Notice that due the large number of classes, some attention-based methods are not feasible for this dataset. Hence, no results for them are available.

### 4.2. Zero-Shot Learning

In Table 6 we present a SotA comparison on NUS-WIDE multi-label zero-shot dataset [9]. Similar to previous works, we use F1 score at top-K predictions and mAP as evaluation metrics. mAP is measured both on ZSL (unseen classes only) and GZSL (seen+unseen classes). Full training details appear in appendix F.

<table>
<thead>
<tr>
<th>Method</th>
<th>Task</th>
<th>mAP</th>
<th>F1 (K = 3)</th>
<th>F1 (K = 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSE [27]</td>
<td>ZSL</td>
<td>10.4</td>
<td>25.8</td>
<td>23.6</td>
</tr>
<tr>
<td></td>
<td>GZSL</td>
<td>3.7</td>
<td>11.5</td>
<td>13.5</td>
</tr>
<tr>
<td>Fast0Tag [46]</td>
<td>ZSL</td>
<td>15.1</td>
<td>27.8</td>
<td>26.4</td>
</tr>
<tr>
<td></td>
<td>GZSL</td>
<td>2.1</td>
<td>7.0</td>
<td>8.1</td>
</tr>
<tr>
<td>Attention per Label [18]</td>
<td>ZSL</td>
<td>12.9</td>
<td>24.6</td>
<td>22.9</td>
</tr>
<tr>
<td></td>
<td>GZSL</td>
<td>2.6</td>
<td>6.4</td>
<td>7.7</td>
</tr>
<tr>
<td>Attention per Cluster [17]</td>
<td>ZSL</td>
<td>19.4</td>
<td>31.6</td>
<td>28.7</td>
</tr>
<tr>
<td></td>
<td>GZSL</td>
<td>5.6</td>
<td>14.4</td>
<td>16.8</td>
</tr>
<tr>
<td>LESA [17]</td>
<td>ZSL</td>
<td>26.3</td>
<td>33.1</td>
<td>30.7</td>
</tr>
<tr>
<td></td>
<td>GZSL</td>
<td>9.3</td>
<td>16.1</td>
<td>19.0</td>
</tr>
<tr>
<td>BiAM [26]</td>
<td>ZSL</td>
<td>25.9</td>
<td>30.5</td>
<td>27.8</td>
</tr>
<tr>
<td></td>
<td>GZSL</td>
<td>12.1</td>
<td>18.5</td>
<td>21.0</td>
</tr>
<tr>
<td>ML-Decoder</td>
<td>ZSL</td>
<td>31.1</td>
<td>34.1</td>
<td>30.8</td>
</tr>
<tr>
<td></td>
<td>GZSL</td>
<td>19.9</td>
<td>23.3</td>
<td>26.1</td>
</tr>
</tbody>
</table>

Table 6. State-of-the-art comparison for multi-label ZSL and GZSL tasks on NUS-WIDE dataset.

We see from Table 6 that our approach significantly outperforms the previous top solution by 4.8% mAP (ZSL), setting a new SotA in this task.

Note that previous methods were mostly aimed at optimizing the task of zero-shot, at the expense of the seen classes (ZSL vs. GZSL trade-off). SDL [3], for example, proposed to use several principal embedding vectors, and trained them using a tailored loss function for ZSL. In contrast to previous methods, ML-Decoder offers a simple unified solution for plain and zero-shot classification, and achieves top results for both ZSL and GZSL. ML-Decoder sets a new SotA score also on the GZSL task, outperforming SDL with a significant improvement (from 12.1% to 19.9%). This clearly demonstrates that ML-Decoder generalizes well to unseen classes, while maintaining high accuracy on the seen classes.

### 4.3. Single-label Classification

To test ML-Decoder effectiveness for single-label classification, we used ImageNet dataset [10], with the high-quality training code suggested in [39] (A2 configuration). Comparison of various ResNet architectures, with different classification heads, appears in Figure 6. As can be seen, when replacing the baseline GAP-based head with ML-Decoder, we significantly improve the models' accuracy. ML-Decoder also provides a better speed-accuracy trade-off compared to using GAP with a larger backbone.

Notice that we used ML-Decoder without introducing any change or tuning any hyper-parameter in the training configuration. The fact that ML-Decoder can serve as a drop-in replacement for GAP-based classification head, in a highly-optimized single-label training setting (baseline ResNet50 achieves 79.7%), and still provides an additional boost, demonstrate its effectiveness and versatility. Also, note that following [39], our training configuration for ImageNet is using a multi-label loss (sigmoid with BCE loss instead of softmax and CE loss). That’s the default mode we present in Figure 2. In Table 16 in the appendix we validate that also with softmax, ML-Decoder provides a benefit compared to plain GAP.

When increasing the number of training epochs to 600 (A1 configuration in [39]), with ML-Decoder and vanilla ResNet50 backbone we reach 80.7% accuracy. To the best of our knowledge, this is the top results so far achieved with ResNet50 (without extra data or distillation).

### 5. Conclusions and Future Work

In this paper, we introduced ML-Decoder, a new attention-based classification head. By removing the redundant self-attention layer and using a novel group-decoding scheme, ML-Decoder scales well to thousands of classes, and provides a better speed-accuracy trade-off.
References


