Learning to Count - Algorithmic Counting

by

Ruben Antonius Adrianus Overwater

To obtain the degree of Master of Science at Delft University of Technology,
to be defended publicly on Monday May 26, 2025

Student Number:	4832205
Supervisors:	Dr. J.C. van Gemert, Dr. J.F.P. Kooij
Faculty:	Mechanical Engineering, TU Delft
Department:	Cognitive Robotics (CoR)

TU Delft

Acknowledgments

I would like to sincerely thank my supervisors, Jan van Gemert and Julian Kooij, for enabling me to do this thesis in this field, as well as their guidance and support throughout the process. Their feedback and mentorship have been invaluable to my academic growth.

Working on this thesis has been an unexpectedly rewarding journey. I discovered a great interest in the topic and found great satisfaction in the research process itself. This experience has not only deepened my understanding of the subject but also revealed a passion I hope to continue exploring.

Lastly, I would like to thank my family and friends for their constant support and motivation during this time.

Learning to Count - Algorithmic Counting

R.A.A. Overwater
MSc. Robotics
Delft University of Technology

Supervised by **Dr. J.C. van Gemert** and **Dr. J.F.P. Kooij,** *Delft University of Technology*

***Abstract*—Visual counting is an important task in computer vision with broad applications in areas such as crowd monitoring, agriculture, and environmental analysis. While deep learning has significantly advanced this field by enabling models to learn robust feature representations, deep learning approaches suffer from sensitivity to data imbalances, which occur in the distribution of object counts across counting datasets as a result of annotation effort. Most state-of-the-art counting models, categorized into clustering-, detection-, regression-, and density estimation-based methods, are built upon Convolutional Neural Networks (CNNs) and Transformers, both of which are known to be susceptible to imbalances in the training data. This study introduces a hybrid model that incorporates a programmatically guaranteed counting mechanism using the RASP language [1] and the Tracr [2] compiler, enabling the construction of Transformer-based models that can reliably execute predefined tasks, such as counting. By combining this exact counting mechanism with a trainable embedding module, we present a model that is capable of learning to count various tokens, even under significant data imbalance. We validate our approach on a synthetic, imbalanced dataset and compare its performance, training time, and data efficiency against standard CNN- and Transformer-based models. Results suggest that our method achieves strong generalization across the full spectrum of object counts while requiring less training data, highlighting the potential for this architecture to be further investigated and adapted to be used for robust and efficient visual counting.**

***Index Terms*—RASP, Tracr, Counting, Transformer, Learning, Compiling**

I. Introduction

Visual counting is a rapidly growing research field with applications in crowd monitoring and control, environmental monitoring, and the agricultural sector [3]. The field studies methods to automate the counting of objects in images or count moving objects and repetitive actions in videos, a task that changed significantly with the advancements in deep learning. Before the widespread use of deep learning, researchers used handcrafted features to detect and count objects of interest while currently most state-of-the-art implementations use some form of deep learning model to learn the most optimal features or transformation. Difficulties in the field of visual counting that deep learning helped solve include the recognition of objects under different poses, varying lighting conditions and image quality [4]–[6] as well as the struggle with occluded objects, dense clusters and varying object sizes [7], [8].

Current deep learning based visual counting models use different techniques to estimate count, which can be used to subdivide these models into into four different categories [9]:

- **clustering-based counting**
- **detection-based counting**
- **regression-based counting**
- **density estimation-based counting**

Each technique has different benefits and applications, for example counting using clustering can operate on unseen data by clustering similar patches [10], whereas counting using regression and density estimation have applications in crowd counting scenarios due to the ability to estimate the count of large occluded clusters of objects, as can be seen in Figure 1. Although these techniques have different architectures and excel in different applications, they are all build upon a set of Convolutional Neural Network (CNN) and Transformer [11] layers.

These visual counting models require a lot of data to be trained, and this data is very costly to annotate. In this field, the four most common ways of annotation are:

- **segmentation annotation**
- **bounding box annotation**
- **center point/dot annotation**
- **total count annotation**

These annotations are arranged in descending order of effort required of human annotators to generate them. While detailed segmentation annotation masks provide the most information and can be transformed into the other types of annotation, they are labor-intensive to create. Simpler annotation techniques, such as total count labels, require less effort but lack spatial information, resulting in limited interpretability regarding what specific features a model learns to base its decision on during training to estimate the final count [13]. This highlights the trade-off between annotation effort and model performance present in this field.

Therefore, most datasets in the field present center point annotations, or other, more informative annotation types. This serves to provide models with spatial information in addition to the total count. Most publicly available datasets for counting tasks focus primarily on crowd counting, due to the significant amount of research dedicated to this domain. Notable crowd counting datasets include ShanghaiTech Parts A and B [14],

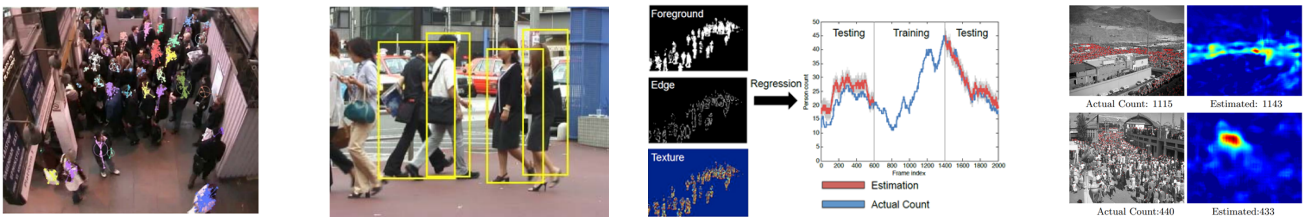

Fig. 1. Illustration of different visual counting techniques, from left to right: clustering based, detection based, regression based and density estimation based counting [9], [12].

UCF_CC_50 [15], UCF_QNRF [16], and WorldExpo'10 [17]. Therefore, in recent years, more counting datasets focused on more general object counting were introduced, such as RSOC [18], DOTA [19], CARPK [20], and FSC-147 [21].

Due to the increased annotation effort required for images with higher object counts, datasets often contain fewer instances with high counts. This results in a distribution where both low and high object counts are underrepresented, as is shown in Figure 2. The figure illustrates this distribution of counts for crowd counting datasets such as ShanghaiTech Parts A and B [14], UCF_CC_50 [15], and the general-purpose counting dataset FSC-147 [21]. The underrepresentation of low-count images arises because counting datasets are designed to differ from object classification datasets by including multiple instances of objects per image. For instance, the FSC-147 counting dataset was developed by Ranjan et al. [21] to address the limitations of datasets like COCO [22], which were deemed unsuitable for counting tasks due to an insufficient number of object instances per image. On the other hand, high-count images are underrepresented because annotating them is labor-intensive, requiring significant effort to label each instance accurately.

Deep learning models, including those based on Convolutional Neural Networks (CNNs) and Transformers, are sensitive to imbalances in training data distributions [23]–[25]. As discussed, since most visual counting models rely on CNNs and Transformers, this issue also arises in visual counting models as well. This sensitivity can lead to performance degradation, particularly when models encounter test samples with object counts that exceed those observed during training. For instance, Hobley et al. (2022) [26] observed a significant drop in performance of their class-agnostic counting model when evaluated on other datasets containing high count images. They attributed this decline to the presence of out-of-distribution samples, noting that 70% of the test images had more instances than the maximum count seen during training. This shows that this count imbalance naturally present in the datasets poses challenges for training robust counting models, as they may not generalize well across the full spectrum of object counts.

In this work, we propose a method that is able to truly count, and is therefore unaffected by imbalances in the dataset. For this we leverage the properties of RASP [1] and Tracr [2]. These frameworks allow us calculate transformer weights that, when implemented into a transformer, perform the exact task that the corresponding program written in the RASP language is designed to do. Since we encode a counting algorithm, we can be sure that the model is able to count. We propose a method that combines a learned embedding model with a RASP-based counting model, creating a hybrid end-to-end model that is guaranteed to count. Therefore, we hypothesize that this model is able to generalize well across the full spectrum of object counts even when trained on an imbalanced dataset. Moreover, since most of the model is precomputed we hypothesize that this model improves upon data efficiency and training time compared to conventional baseline models.

To test these hypotheses, we create a simplified imbalanced dataset. We train our model on this dataset and compare its performance to baseline models that consist of CNN and Transformer backbones. We compare the training time of these models and vary the size of the simplified dataset to test data efficiency. The final models are evaluated on the full count range present in the original imbalanced dataset, in order to verify whether they are truly able to count and deal with imbalanced training data. Our contributions are summarized by a model that is expected to:

- **generalize effectively despite significant data imbalance**
- **reduce the amount of training data required**
- **decrease training time**

In section II, we review Related Work, where we explore RASP and Tracr. Moreover, we also expand our research to the field of visual counting. In section III we outline the Methodology section. We describe the architecture of the models we propose, the datasets used to test and the design of the training loop. In the Experiments section (IV), we elaborate on the designed experiments and their outcome. In the Conclusion section (V), we reflect on the results of these experiments and discuss the most important findings, after which we conclude this paper and give some recommendations on future work.

II. Related Work

This section reviews the concepts and frameworks relevant to our work. We first elaborate on the RASP and Tracr frameworks, highlighting their potential for creating interpretable and robust Transformer models. We then discuss the state-of-the-art visual counting methods, emphasizing their limitations concerning dataset imbalances and the potential benefits of integrating RASP-based solutions.

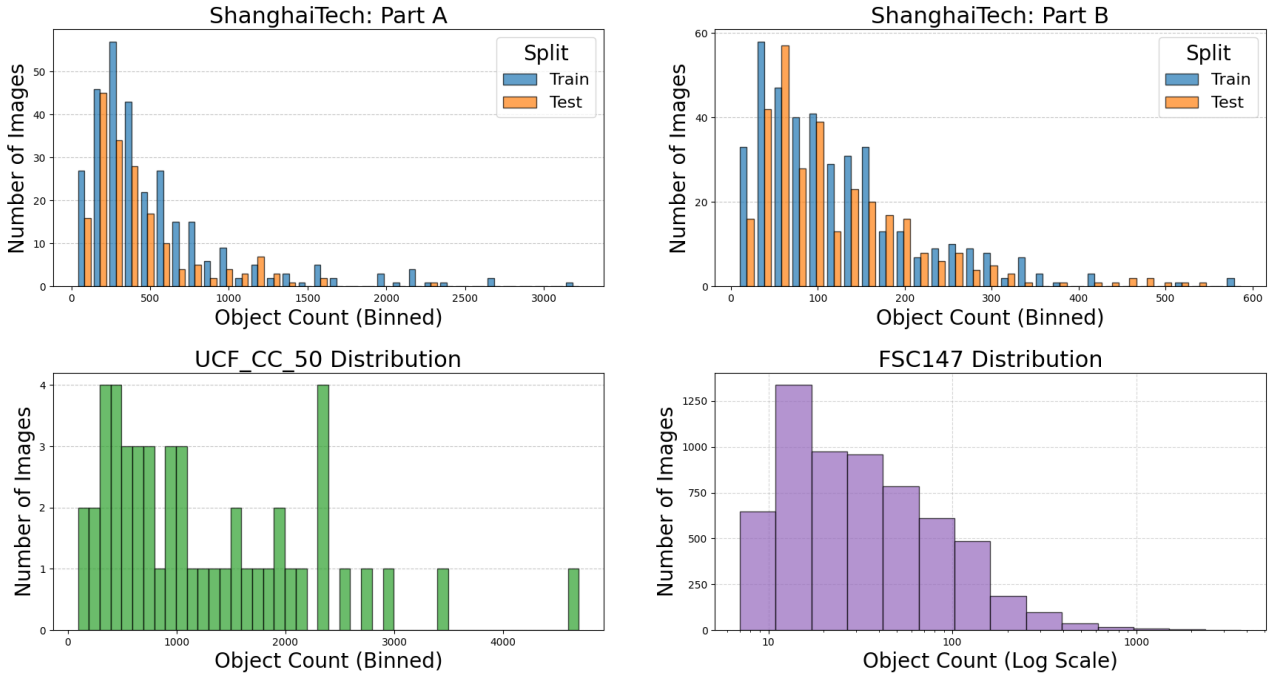

Fig. 2. The imbalance in distribution of counts shown for the crowd counting datasets ShanghaiTech Parts A and B [14] (top) and UCF_CC_50 [15] (bottom left), and for the general purpose counting dataset FSC-147 [21] (bottom right). The long tail of the FSC-147 is shown on a log scale for demonstration purposes.

A. *RASP*

The Restricted Access Sequence Processing (RASP) language, introduced by Weiss et al. [1], provides a programming framework designed to model and understand the computational capabilities of Transformer models [11]. Whereas recurrent neural networks (RNNs) have an abstract computational model in the form of finite state machines used for analysis, transformers do not. RASP addressed this gap by translating the fundamental operations of Transformer architectures, the attention mechanisms and feed-forward layers, into a set of three primitive operations. Expressing programs and solutions in terms of these primitives allows researchers to better understand and analyze transformer computations.

Weiss et al (2021) [1] demonstrate the utility of RASP by creating programs for various algorithmic tasks, such as computing histograms, sorting tokens, and the reversal of strings. By formulating these tasks as RASP programs, researchers can predict the maximum number of attention heads and the minimum number of layers required for a transformer to perform these tasks, thereby improving interpretability.

Furthermore, the authors show that it is possible for a transformer to learn a solution proposed by the RASP framework. They introduce an additional loss term, encouraging the transformer to learn the attention patterns created by the compiled RASP program. This allows for the training of interpretable transformer models where the underlying operations are known and can be analyzed and understood. By providing a means to "compile" high-level algorithmic operations into transformer models, RASP enables the development of more transparent and explainable AI systems.

In summary, the introduction of RASP presents a significant advancement in the interpretability and theoretical understanding of transformer models. By mapping transformer operations to simple primitives, Weiss et al (2021) [1] offer a powerful tool for analyzing existing models as well as designing new architectures with predictable and interpretable behavior.

B. *Tracr*

Building on RASP, Lindner et al. (2023) [2] proposes Tracr, a compiler that is capable of translating high-level RASP programs directly into transformer models. This compiler enables the construction of transformer models with known functionality based on RASP programs, and therefore improve the study of interpretability. Tracr compiled programs provide a ground truth functionality against which existing interpretability methods can be evaluated.

Compared to RASP, Tracr actually compiles the primitives to form a transformer model. They do this by ordering the RASP primitives from the program into a computational graph structure, after which they translate the nodes to corresponding attention heads and feed forward layers. The final output is a transformer model, implemented in the Haiku [27] framework.

By having a known functionality, these models serve as valuable tools for examining phenomena like superposition in transformers. Superposition is a process during which transformers can compress a large number of sparse features into fewer dimensions. By compressing Tracr compiled models

using gradient descent, the authors observe that compressed models tend to drop unnecessary features and represent less important features in superposition. However, by analyzing the encodings, the authors also note that these compressed models probably do not stay faithful to the originally intended RASP programs.

Furthermore, the ability of Tracr to generate models with known functionality provides a method to evaluate interpretability methods directly. Since regular tasks learned by transformers are often unknown, it is difficult to determine whether the interpretation is correct or accurate. With Tracr, researchers can compare the known functionality of compiled models to the explanations produced by interpretability tools, and thereby asses their accuracy and reliability.

In summary, Tracr offers a new approach based on RASP to study transformer interpretability by enabling the creation of models with known, human-readable functionality. This capability is not only used in understanding complex transformer behavior but it also provides a ground truth for evaluating the effectiveness of interpretability methods.

C. Visual Counting

Visual counting methods, as discussed previously, are typically subdivided into four categories: clustering-based, detection-based, regression-based, and density estimation-based methods. While these methods differ in their specifics and practical applications, they uniformly rely on deep learning architectures, predominantly Convolutional Neural Networks (CNNs) and Transformers, to extract informative features and predict the final counts [3], [9].

Despite numerous advancements after the introduction of deep learning, visual counting methods face challenges related to dataset imbalance. As illustrated in Figure 2, publicly available counting datasets such as ShanghaiTech A and B [14], UCF_CC_50 [15], and FSC-147 [21] show that both extremes of object counts are underrepresented. Lower count images are less frequent due to counting datasets having multiple-instance images, while higher count images are underrepresented due to the annotation effort required to create them [13] [21]. This imbalance impacts the generalization ability of models, particularly in scenarios where test data contains counts significantly outside the distribution observed during training [26].

To address these limitations, we propose a hybrid counting model that leverages the strengths of RASP and Tracr. Our proposed architecture integrates an exact counting mechanism derived from a RASP program compiled using Tracr into a Transformer model. Since this counting mechanism is exact, this model is truly able to count, and is therefore expected to be able to generalize well while being trained on imbalanced data. To improve the practical utility, we extend this compiled Transformer with a learnable embedding module to pre-process the input, enabling it to count a diverse set of tokens from its vocabulary effectively. In this way, our model is expected to generalize well to imbalanced data. Since most of the model is precomputed, we also expect our model to improve of data efficiency and training time, compared to regular CNN and Transformer based models.

To conclude, RASP and Tracr are tools that can be used to compile algorithmic logic and programs into transformer weights without the need for gradient-based training. By integrating RASP and Tracr into visual counting, we expect to address the limitations posed by dataset imbalance, by creating a model that generalizes well to imbalanced data. Specifically, our contributions are characterized by a model that is expected to:

- **generalize effectively despite significant data imbalance**
- **reduce the amount of training data required**
- **decrease training time**

The following sections provide insights into our proposed method, dataset preparation, and experimental design.

III. Method

The RASP source code created by Weiss et al. [1] provides an example for a counting function. This function can be found below in Listing 1. However, as can be seen, this example code has a limitation as it requires the token to be counted to be specified at compile time. As a result, this token is permanently embedded into the compiled Transformer weights, restricting flexibility after model creation.

Listing 1. A default RASP function to count occurrences of a predefined token in a sequence.

```
def make_count(sop, token):
  """Returns the count of `token` in `
      sop`.

  The output sequence contains this
      count in each position.

  Example usage:
    count = make_count(tokens, "a")
    count(["a", "a", "a", "b", "b", "c
        "])
    >> [3, 3, 3, 3, 3, 3]
    count(["c", "a", "b", "c"])
    >> [1, 1, 1, 1]

  Args:
    sop: Sop to count tokens in.
    token: Token to count.
  """
  return rasp.SelectorWidth(rasp.Select
      (sop, sop, lambda k, q: k == token
      )).named(f"count_{token}")
```

To overcome this limitation, we develop a novel RASP-based counting method capable of counting variable tokens determined at runtime. The corresponding RASP program is presented in Listing 2. Unlike the default RASP function, this custom implementation counts occurrences of the first token appearing in the input sequence. Our approach leverages the three fundamental RASP primitives, *Select*, *SelectorWidth*, and *Aggregate*, which are sufficient to represent essential computational patterns within Transformer architectures [1].

Specifically, our program first identifies and extracts the token at the initial position, and subsequently counts its occurrences. This counting operation is achieved by generating a boolean array indicating token equality at each position between the token to be counted and the input sequence. After summing this boolean array and broadcasting the resulting count to uniformly replace the input sequence, the sequence is returned, effectively counting the first token occurring in the sequence (Line 8, Listing 2). This novel formulation is unique compared to existing RASP examples, which all require static compile-time tokens to perform tasks, such as count, shift tokens or compute token frequencies, and thus lack dynamic flexibility. Consequently, the top half of our hybrid model should learn to select and insert the appropriate counting token at runtime at the initial position, enabling robust counting across any token defined in the model's vocabulary.

Listing 2. A custom RASP function that counts the occurrences of the first token in the total sequence.

```
1 def count_agnostic_first(self):
2     """Counts the first token in the
          sequence
3     """
4     ALL_TRUE = rasp.Select(rasp.tokens,
           rasp.tokens, rasp.Comparison.
          TRUE)
5     IDX = rasp.SelectorWidth(ALL_TRUE)
          * 0
6     SELECT_FIRST = rasp.Select(rasp.
          indices, IDX, rasp.Comparison.EQ
          )
7     FIRST_TOK = rasp.Aggregate(
          SELECT_FIRST, rasp.tokens)
8     COUNT = rasp.SelectorWidth(rasp.
          Select(rasp.tokens, FIRST_TOK,
          rasp.Comparison.EQ))
9     return COUNT
```

The developed RASP program (Listing 2) is compiled into a Transformer model using the Tracr framework, producing an model using the Haiku library [27], which relies on the JAX framework for computational performance [28]. Despite JAX's efficiency, we converted our compiled models into PyTorch due to its broader adoption, established community support, and widespread academic usage. The conversion procedure required modifying the Tracr source code and implementing additional utilities to facilitate translation between Haiku and PyTorch model definitions. These modifications can be found in our source code repository [1].

The final PyTorch model, as presented in Table I, consists of four distinct Transformer blocks. Notably, the compiled RASP model inherently includes an Argmax operation, a non-differentiable step in the final unembedding layer. For training purposes, we removed this Argmax operation and instead utilized the raw logits directly in the loss calculation against the count supervisory signal. However, as the initial compiled model includes a fixed embedding layer, it does not permit gradient updates upstream of this layer, necessary for end-to-end training. To enable effective training, we inserted a learnable classifier module between the frozen embedding and the compiled Transformer, operating on the embeddings of the input tokens.

[1]https://github.com/roverwater/Algorithmic-Counting

TABLE I
ARCHITECTURE OF OUR RASP BASED MODEL, TRANSLATED INTO PYTORCH

Module	Description
Embedding Module	
`pos_embed`	Embedding(39, 132)
`token_embed`	Embedding(7, 132)
Transformer Module	
`ATT_Module` (x4)	
Softmax	Softmax(dim=-1)
Query Matrix	Linear(132, 39)
Key Matrix	Linear(132, 39)
Value Matrix	Linear(132, 39)
Linear Matrix	Linear(39, 132)
`MLP_Module` (x4)	
Linear 1	Linear(132, 78)
Activation	ReLU()
Linear 2	Linear(78, 132)
Unembedding Module	
`unembed`	Linear(132, 39, bias=False)

We conducted various experiments to design an effective classifier. Initially, a simple fully-connected neural network was employed between the embedding and Transformer modules. However, this proved inadequate due to convergence issues. Subsequently, we tested inserting an additional Transformer block, mirroring the downstream Transformer architecture. This modification achieved perfect in-distribution accuracy but exhibited poor out-of-distribution generalization to sequences of different lengths. Our objective was to teach the transformer to only replace the embedding of first token, inherently ensuring sequence-length-independent generalization. Consequently, this approach was discarded because it did not achieve this objective and intended behavior.

Further attempts utilized classifiers operating explicitly on a single token embedding (Listing 3). Although both transformer-based and linear classifiers succeeded in isolated testing scenarios with direct supervision on the output of the embeddings classifier layer, their performance significantly deteriorated when integrated into the full model. We hypothesize that there are two reasons for this behavior:

- The shear challenge of precisely learning highly sparse token embeddings positioned early within the computational graph, requiring exact matches downstream.
- Insufficient gradient signals propagating from high-level count supervision through multiple Transformer layers, complicating the accurate recovery of sparse embeddings.

Although insightful, these hypotheses remain speculative, since we did not further verify these hypotheses.

Listing 3. Initial classifier design, operating on token level.

```
1 def forward(self, x):
2     token = x[:, 1, :]
```

```
    x_out = x.clone()
    x_out[:, 1, :] += self.classifier(
        token) # += or =
    return x_out
```

Ultimately, we adopted an alternative classifier design with a small number of learnable parameters leveraging the Gumbel-Softmax distribution [29] to approximate differentiable token selection (Listing 4). Specifically, the classifier maintains a learnable logit vector (line 1), which is passed through the Gumbel-Softmax function to probabilistically select the token to insert (line 7 - 13). This token is taken from the matrix containing the token embeddings, representing the vocabulary originally computed while compiling the RASP code using Tracr. Positional embeddings were decoupled from token embeddings, as previous experiments showed improper positional encoding as a major factor hindering performance (line 15). Additionally, a temperature annealing schedule was implemented to be used during training to initially encourage exploration before converging to a stable token selection. In the final experiments to reduce variability the temperature is kept at an constant value.

Listing 4. Design of the final classifier used, using Gumbel Softmax to predict token.

```
token_logits = nn.Parameter(torch.zeros
    (self.vocab_size))
self.token_embeddings #Token embedding
    matrix

def forward(self, x, temperature,
    tmp_position_idx=1):
    batch_size = x.shape[0]

    soft_weights = F.gumbel_softmax(
        self.token_logits.unsqueeze(0).
            expand(batch_size, -1),
        tau=temperature,
        hard=False
    )

    replacement_token_emb = torch.
        einsum("bv,vd->bd", soft_weights
        , self.token_embeddings)

    pos_emb = self.pos_embed(torch.
        tensor(tmp_position_idx, device=
        x.device))
    replacement_emb =
        replacement_token_emb + pos_emb.
        unsqueeze(0)

    x_modified = x.clone()
    x_modified[:, tmp_position_idx, :]
        = replacement_emb

    return x_modified
```

Figure 3 depict the token selection probabilities, taken from within the embedding classier, evolving over epochs. Early training demonstrates exploration among multiple candidate tokens, quickly narrowing the probability distribution. After approximately 10 epochs, token ”1” emerges as the dominant candidate, rapidly converging to near-certainty by around epoch 50. This behavior confirms successful convergence and validates the intended use of our embedding classifier.

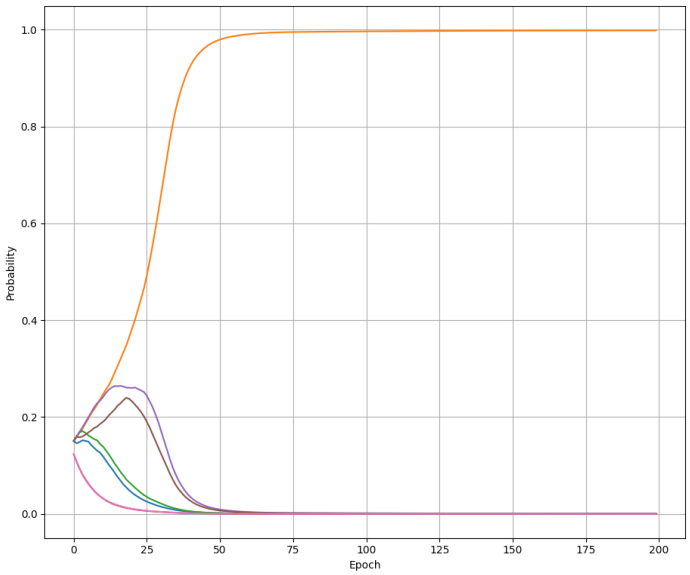

Fig. 3. Evolution of the selected token during training, showing probability against epoch. Observe the probability of the token denoted in orange being selected increasing as training progresses.

Through these developments, our proposed hybrid counting model integrates the algorithmic counting capability provided by RASP with a flexible embedding-selection mechanism, enabling generalizable counting across input tokens and sequences.

IV. Experiments

To validate the hypotheses formulated in the Introduction, we designed a set of comparative experiments involving six distinct models:

- **RASP-based compiled model (ours)**
- **RASP-based model architecture (uninitialized and fully trainable)**
- **CNN feature extractor with classification head**
- **CNN feature extractor with regression head**
- **Transformer feature extractor with classification head**
- **Transformer feature extractor with regression head**

The model referred to as the ”RASP-based model architecture” shares the same overall structure as our compiled solution. However, unlike the compiled version, this model is randomly initialized at the start of training, with all layers, including the embedding layers, set as fully trainable.

For each model, we selected loss functions appropriate for their architecture. Specifically, regression-based models utilized Mean Squared Error (MSE), classification-based models employed Cross Entropy Loss, and our proposed RASP-based compiled model employed Kullback–Leibler Divergence Loss (KL-divergence) for its ability to deal with multiclass classification. Unlike standard classification, RASP requires predicting structured distributions over output sequences. For

example, in our counting solution given an input of [1, 1, 1, 0], the model learns to assign uniform probabilities to specific positions (e.g., class 3), resulting in logits that yield [3, 3, 3, 3] after the final argmax operation. KL-divergence is suited here because it measures how well the predicted soft distribution matches the target distribution.

All models were trained on a synthetic dataset comprising 1000 samples, each instance containing a sequence of 28 tokens. An illustrative data sample is shown in Listing 5. We observe a beginning of sequence ('BOS') token followed by a temporary ('TMP') token. The intended use of our model is to learn to replace the embedding of the 'TMP' token with the embedding of the token of interest, which, in this example, is '1'. Each dataset instance includes sequences of tokens along with associated labels indicating the token count of the token of interest. We maintained a fixed 90/10 training-validation split across experiments, varying only the data points allocated to each subset to ensure robustness. Multiple learning rates were tested for each model, and we report results corresponding to the optimal learning rate that achieved the fastest convergence. The count distribution within the dataset, designed to mimic imbalances encountered in real-world counting datasets, is depicted in Figure 4.

Listing 5. Data set sample. The temporary token 'TMP' is the token that will be replaced by the learned token of interest. In this sample, the token of interest is '1', corresponding to count label 3.

```
input = [['BOS','TMP','SEP','1','1','0'
    ,'2','1','2', ....]]
label = [[3]]
```

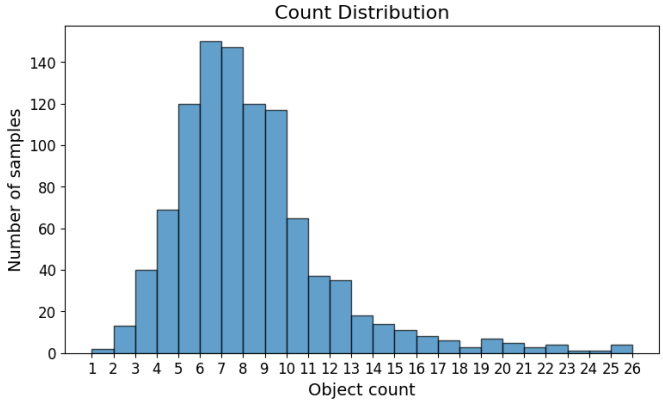

Fig. 4. Distribution of counts in the synthetic dataset, mimicking the imbalance in counting datasets as demonstrated in Figure 2.

A. Generalization under Data Imbalance

To examine each model's ability to generalize across an imbalanced distribution of counts, we evaluated performance on a balanced test set. This evaluation set contained 100 samples per count class across the entire possible count range (0–25). Due to combinatorial constraints, some counts at the upper end included overlapping sequences. The evaluation was replicated over seven experimental runs, and detailed results are reported in Appendix A. In run 2, our RASP model did not converge to a solution. We hypothesize that poor convergence occasionally occurs due to random unfortunate initialization, combined with weak gradient signals. These weak gradients likely originate from the nature of the total count supervision, which must propagate through a deep architecture containing numerous frozen layers and sparse embeddings, thereby limiting effective gradient flow. For the sake of comparison, in the results down below we decided to omit this example and present the top 5 performing runs for our RASP model.

Our RASP-based compiled model demonstrated strong generalization across all count categories, consistently achieving near-perfect accuracy (Figure 5). Minor deviations from perfect accuracy in some runs (notably run 5) occurred due to incomplete convergence of the Gumbel-Softmax token selection, explaining the noise observed. Extended training beyond the 1000 training epochs or hard token selection could further mitigate these discrepancies.

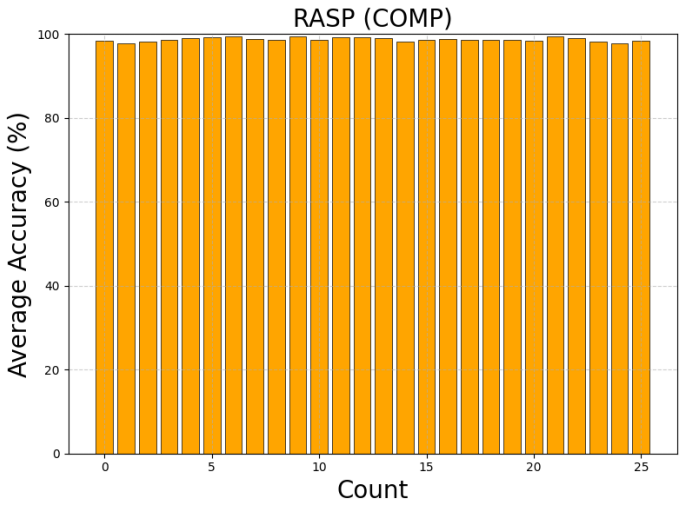

Fig. 5. Generalization performance of RASP-based compiled model over the full count range. Observe the near perfect performance across the count range, while trained on data distributed as seen in Figure 4.

In contrast, Transformer- and CNN-based models with classification heads (Figures 6, 7) showed substantial accuracy drops at the extremes of the count distribution. These models achieved near-perfect performance only within the highly represented count ranges, failing notably at the lower and upper extremes due to data scarcity.

Transformer-based regression models and the fully trainable (uninitialized) RASP model architecture exhibited improved generalization compared to classification counterparts but still struggled with counts at dataset extremes (Figures 8, 9).

Remarkably, the CNN-based regression model exhibited perfect generalization performance across the entire count range (Figure 10). This outcome suggests the CNN-based regressor effectively interpolated across the imbalanced dataset distribution. However, we hypothesize that further increasing

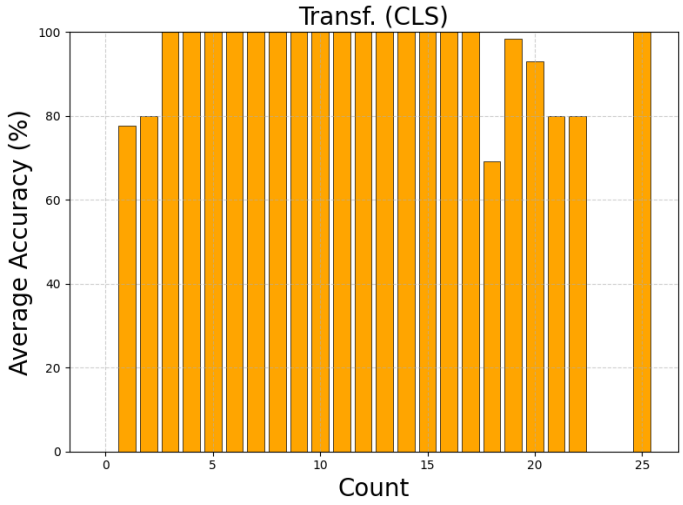

Fig. 6. Transformer model with classification head generalization. Observe the poor performance near the extremes of the count ranges, while trained on data distributed as seen in Figure 4.

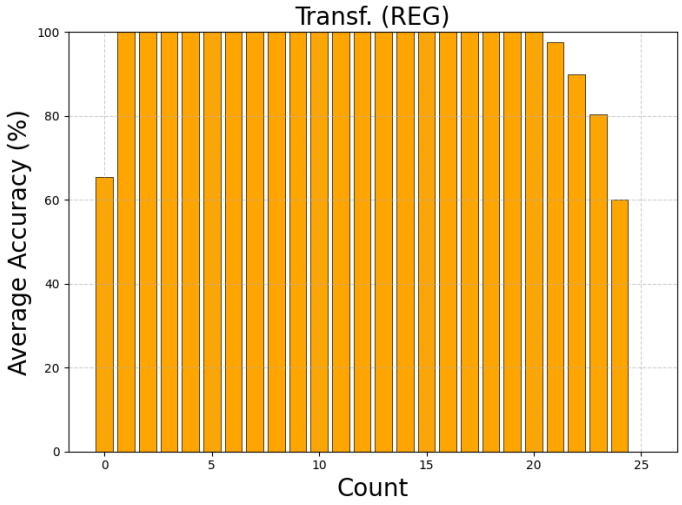

Fig. 8. Transformer model with regression head generalization. Observe the poor performance near the extremes of the count ranges, while trained on data distributed as seen in Figure 4.

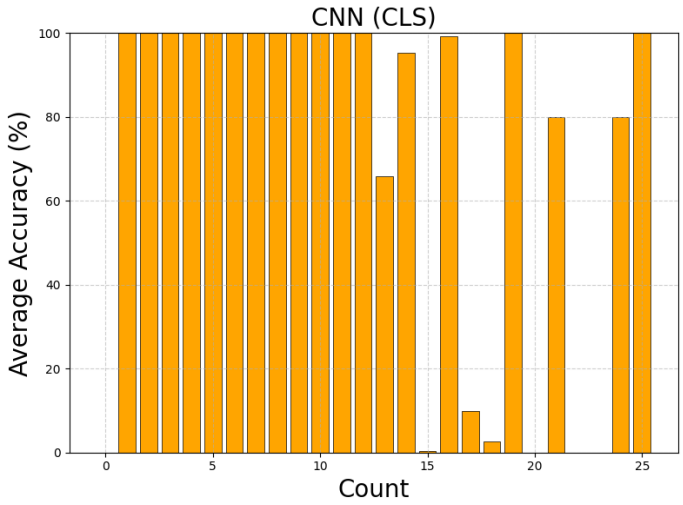

Fig. 7. CNN model with classification head generalization. Observe the poor performance near the extremes of the count ranges, while trained on data distributed as seen in Figure 4.

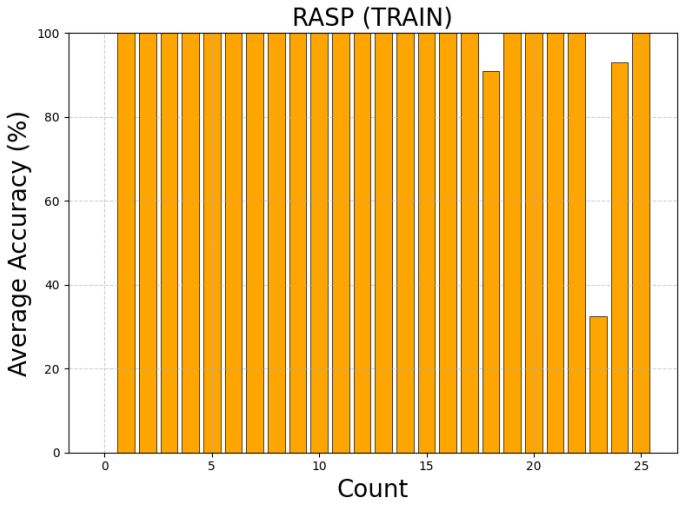

Fig. 9. Trainable RASP model (uninitialized) generalization. Observe the poor performance near the extremes of the count ranges, while trained on data distributed as seen in Figure 4.

imbalance, especially extending into extrapolative ranges, may eventually challenge the regression model's performance.

B. Data Efficiency

To assess data efficiency, we trained each model variant on progressively smaller datasets, specifically datasets of 100, 250, 500, 750, and 1000 samples (maintaining a 90/10 split). Detailed outcomes are documented in Appendix B.

Our RASP-based compiled model exhibited good data efficiency, achieving strong generalization across the count range even at reduced dataset sizes of 250 samples (20). In contrast, classification-based models exhibited significantly deteriorating performance with fewer samples, requiring larger datasets for robust generalization. However, the CNN-based regression model maintained consistently strong performance even on reduced datasets, such as 100 samples (19), reaffirming its capability for effective interpolation given very limited data.

C. Training Time Efficiency

To evaluate training-time efficiency, we monitored validation accuracy over training epochs across models. Figure 11 illustrates results from the optimal experimental run (run 4) of our RASP-based compiled model relative to other architectures.

Our compiled RASP model reached 100% accuracy much faster (epoch 95) compared to the CNN regression model (epoch 200). However, performance variability across multiple runs (notably runs 5-7) suggests marginal gains over conventional models in some scenarios and negligible differences in others. Consequently, we cannot definitively claim superior

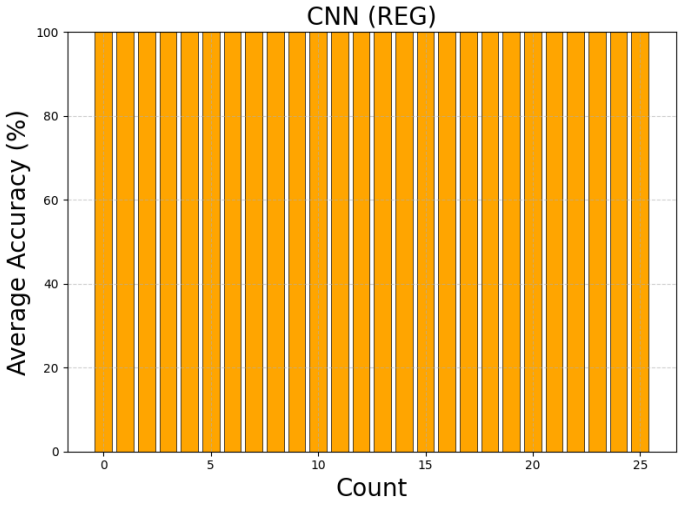

Fig. 10. CNN model with regression head generalization. Observe the perfect performance across the count range, while trained on data distributed as seen in Figure 4.

training efficiency for our compiled model based solely on these experiments.

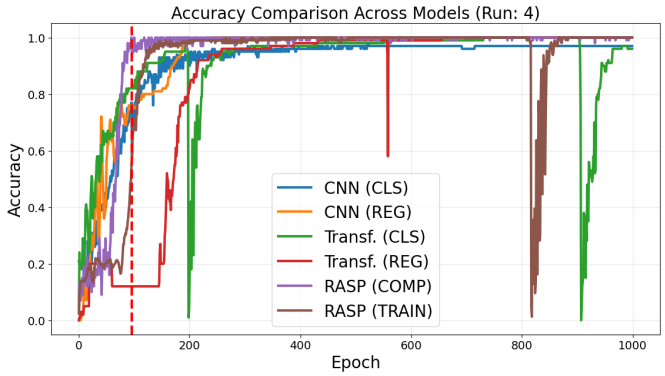

Fig. 11. Validation accuracy over training epochs (optimal run 4). Observe the marginal improvement between RASP (COMP) (ours) and other models.

V. Discussion and Conclusion

This work leverages the Restricted Access Sequence Processing (RASP) language [1] and the Tracr compiler framework [2], to introduce a hybrid visual counting model explicitly designed to mitigate challenges associated with imbalanced counting datasets. We compared our proposed model against CNN- and Transformer-based counting models across multiple experiments: generalization under dataset imbalance, data efficiency, and training-time efficiency.

Our experiments show that the compiled RASP-based model achieves strong generalization performance across imbalanced count distributions, notably surpassing Transformer and CNN models with classification heads, as well as the Transformer model with a regression head and the uninitialized, fully trainable model based on the RASP architecture, particularly at the extremes of the dataset distribution. The explicit counting mechanism provided by RASP ensures that the model's accuracy remains invariant to data imbalance, addressing limitations encountered in the field of viusal counting [26]. However, the CNN-based counting model with a regression head model unexpectedly matched our RASP-based model's performance in generalization experiments, suggesting that regression presents strengths in interpolating between sparse counts. Nevertheless, we hypothesize that this advantage may diminish as the imbalance grows or when extrapolation beyond training distributions becomes necessary, scenarios that can occur in practical applications (observe the extremely long and sparse tail of the FSC-147 dataset (Figure 2).

Regarding data efficiency, our RASP-based model demonstrated robustness, successfully learning accurate counting representations from significantly reduced datasets. In contrast, the classification based models exhibited notable performance degradation with limited training data, while the CNN-regression architecture also retained its strong interpolation capability even for small datasets, surpassing our model. Further experiments are necessary to evaluate CNN-regression architectures on datasets with greater imbalance, to determine whether their performance advantage remains under more challenging extrapolation conditions compared to our proposed RASP-based solution. Overall we show that the integration of counting via RASP within a Transformer-based context improves data efficiency, which is particularly relevant in scenarios with limited annotation resources [13].

Concerning training efficiency, our model achieved slightly faster convergence compared to baseline models, but these improvements were small and inconsistent across runs. Hence, definitive claims regarding substantial training-time advantages cannot be drawn.

A. Limitations and Future Work

Despite promising results, our approach presents certain limitations which require additional research. The token-selection mechanism introduced via the Gumbel-Softmax formulation occasionally exhibited unstable convergence, suggesting that further optimization improvements are necessary. Moreover, our current experiments are performed with simplified synthetic datasets; therefore, future studies should validate this methodology on real-world visual counting datasets. However for this, more research should be performed on RASP, as it is currently only possible to operate on discrete tokens. Different methods, such as mapping visual patches to discrete token vectors could also be investigated.

Additionally, while our RASP-based model demonstrated robustness under imbalance, further experiments should be performed on datasets with greater imbalance. Datasets with extreme imbalance, that require significant interpolation and extrapolation, are hypothesized to show the superior performance of our solution compared to a CNN model using regression to estimate the final count.

Our results indicate that hybrid models that use RASP based models offer promising solutions to challenges in visual count-

ing. While additional development is needed, the approach shows considerable potential.

ACKNOWLEDGMENT

I would like to express gratitude towards Jorge Romeu Huidobro, for getting me up to speed with the RASP and Tracr framework.

REFERENCES

[1] Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers, 2021.

[2] David Lindner, János Kramár, Sebastian Farquhar, Matthew Rahtz, Thomas McGrath, and Vladimir Mikulik. Tracr: Compiled transformers as a laboratory for interpretability, 2023.

[3] Luca Ciampi. Deep learning techniques for visual counting, 2022.

[4] Tae-su Wang, Gi-Tae Kim, Jungpil Shin, and Si-Woong Jang. Hierarchical image quality improvement based on illumination, resolution, and noise factors for improving object detection. *Electronics*, 13(22), 2024.

[5] Aoran Yi and Nipon Anantrasirichai. A comprehensive study of object tracking in low-light environments. *Sensors*, 24(13):4359, 2024.

[6] Aetesam Ali Khan Ashar, Asir Abrar, and Jiangjiang Liu. A survey on object detection and recognition for blurred and low-quality images: Handling, deblurring, and reconstruction. In *Proceedings of the 2024 8th International Conference on Information System and Data Mining*, ICISDM '24, page 95–100, New York, NY, USA, 2024. Association for Computing Machinery.

[7] Lei Chen, Xinghang Gao, Fei Chao, Xiang Chang, Chih Min Lin, Xingen Gao, Shaopeng Lin, Hongyi Zhang, and Juqiang Lin. The effectiveness of a simplified model structure for crowd counting, 2024.

[8] Adriano D'Alessandro, Ali Mahdavi-Amiri, and Ghassan Hamarneh. Counting objects in images using deep learning: Methods and current challenges. Technical Report 2986682, Research Square, June 2023.

[9] Chen Change Loy, Ke Chen, Shaogang Gong, and Tao Xiang. *Crowd Counting and Profiling: Methodology and Evaluation*, pages 347–382. Springer New York, New York, NY, 2013.

[10] Yanan Luo, Jinhui Yi, Yazan Abu Farha, Moritz Wolter, and Juergen Gall. Rethinking temporal self-similarity for repetitive action counting, 2024.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[12] Lokesh Boominathan, Srinivas S S Kruthiventi, and R. Venkatesh Babu. Crowdnet: A deep convolutional network for dense crowd counting. In *Proceedings of the 24th ACM International Conference on Multimedia*, MM '16, page 640–644, New York, NY, USA, 2016. Association for Computing Machinery.

[13] Uddhav Bhattarai and Manoj Karkee. A weakly-supervised approach for flower/fruit counting in apple orchards. *Computers in Industry*, 138:103635, 2022.

[14] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-image crowd counting via multi-column convolutional neural network. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 589–597, 2016.

[15] Haroon Idrees, Imran Saleemi, Cody Seibert, and Mubarak Shah. Multi-source multi-scale counting in extremely dense crowd images. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2547–2554, 2013.

[16] Haroon Idrees, Muhmmad Tayyab, Kishan Athrey, Dong Zhang, Somaya Al-Máadeed, Nasir M. Rajpoot, and Mubarak Shah. Composition loss for counting, density map estimation and localization in dense crowds. *CoRR*, abs/1808.01050, 2018.

[17] Cong Zhang, Hongsheng Li, Xiaogang Wang, and Xiaokang Yang. Cross-scene crowd counting via deep convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 833–841, 2015.

[18] Guangshuai Gao, Qingjie Liu, and Yunhong Wang. Counting from sky: A large-scale data set for remote sensing object counting and a benchmark method. *IEEE Transactions on Geoscience and Remote Sensing*, 59(5):3642–3655, May 2021.

[19] Gui-Song Xia, Xiang Bai, Jian Ding, Zhen Zhu, Serge Belongie, Jiebo Luo, Mihai Datcu, Marcello Pelillo, and Liangpei Zhang. Dota: A large-scale dataset for object detection in aerial images, 2019.

[20] Meng-Ru Hsieh, Yen-Liang Lin, and Winston H. Hsu. Drone-based object counting by spatially regularized regional proposal network, 2017.

[21] Viresh Ranjan, Udbhav Sharma, Thu Nguyen, and Minh Hoai. Learning to count everything, 2021.

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[23] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259, October 2018.

[24] Zeju Li, Konstantinos Kamnitsas, and Ben Glocker. Analyzing overfitting under class imbalance in neural networks for image segmentation. *IEEE Transactions on Medical Imaging*, 40(3):1065–1077, March 2021.

[25] Kaidi Cao, Colin Wei, Adrien Gaidon, Nikos Arechiga, and Tengyu Ma. Learning imbalanced datasets with label-distribution-aware margin loss, 2019.

[26] Michael Hobley and Victor Prisacariu. Learning to count anything: Reference-less class-agnostic counting with weak supervision, 2022.

[27] Tom Hennigan, Trevor Cai, Tamara Norman, Lena Martens, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020.

[28] Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. In *Proceedings of the SysML Conference 2018*, 2018.

[29] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2017.

APPENDIX A
GENERALIZATION RUN DATA

Figures 12, 13, 14, 15, 16, 17, 18

APPENDIX B
DATA EFFICIENCY DATA

Figures 19, 20, 21, 22

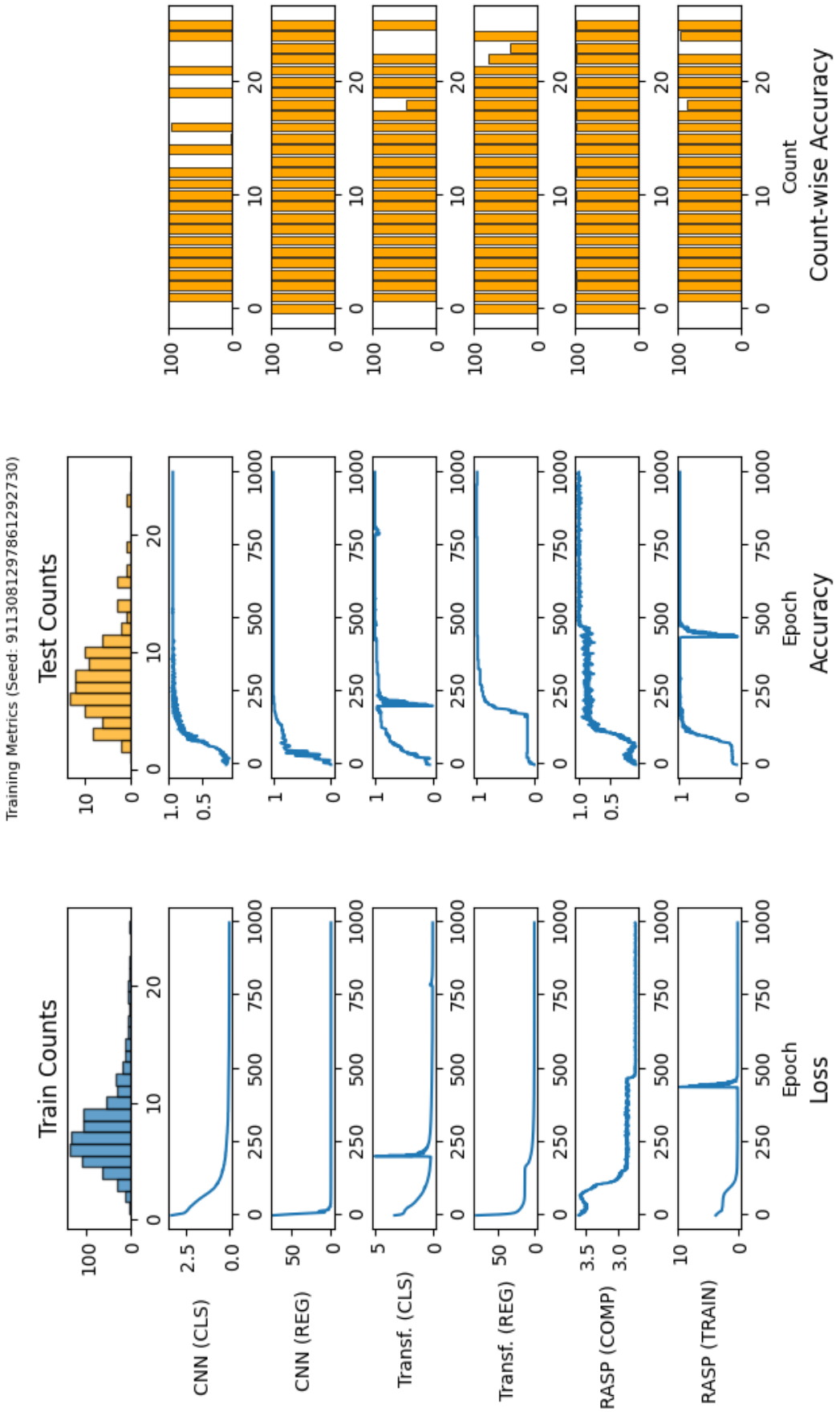

Fig. 12. Data run 1

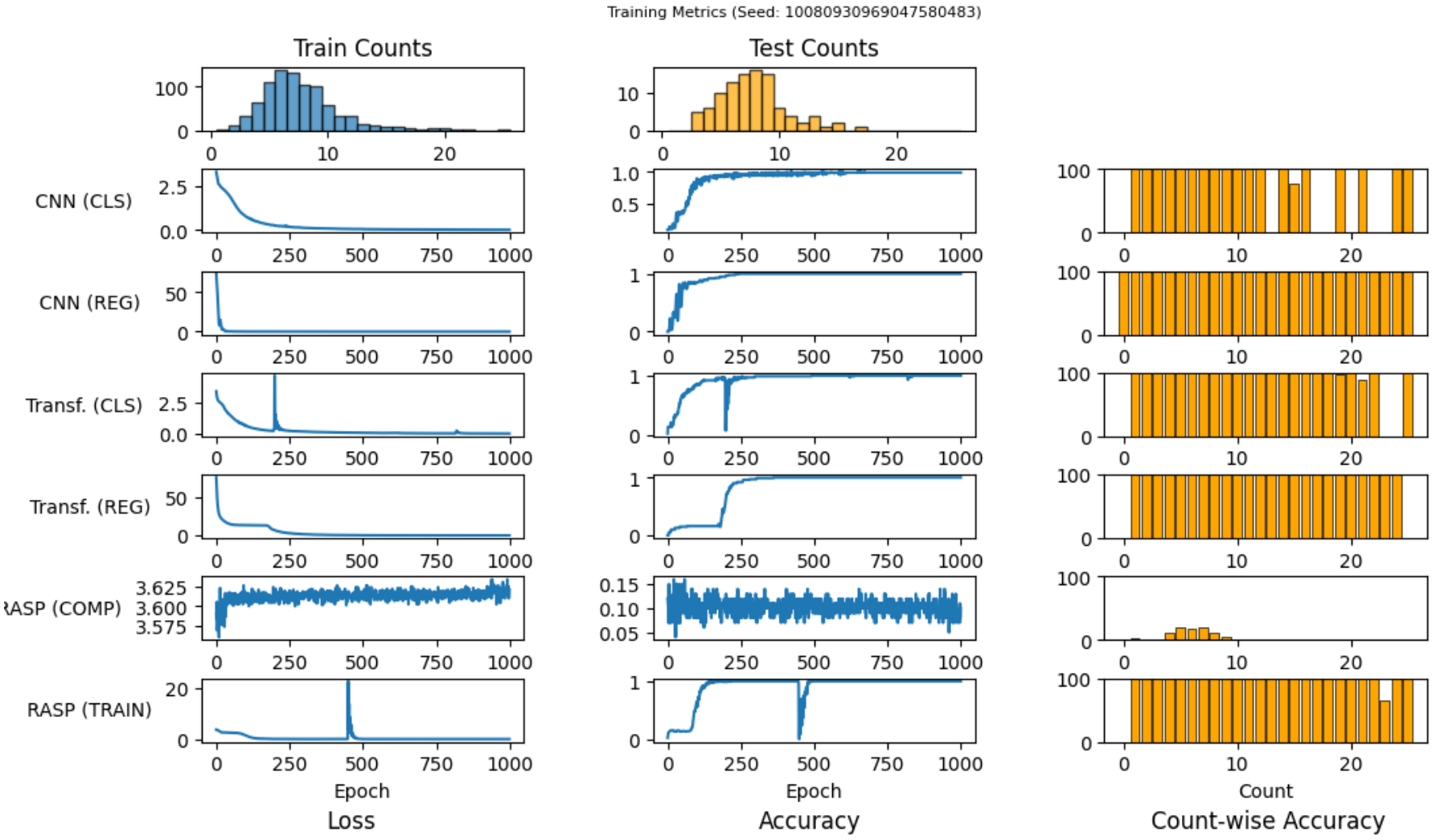

Fig. 13. Data run 2

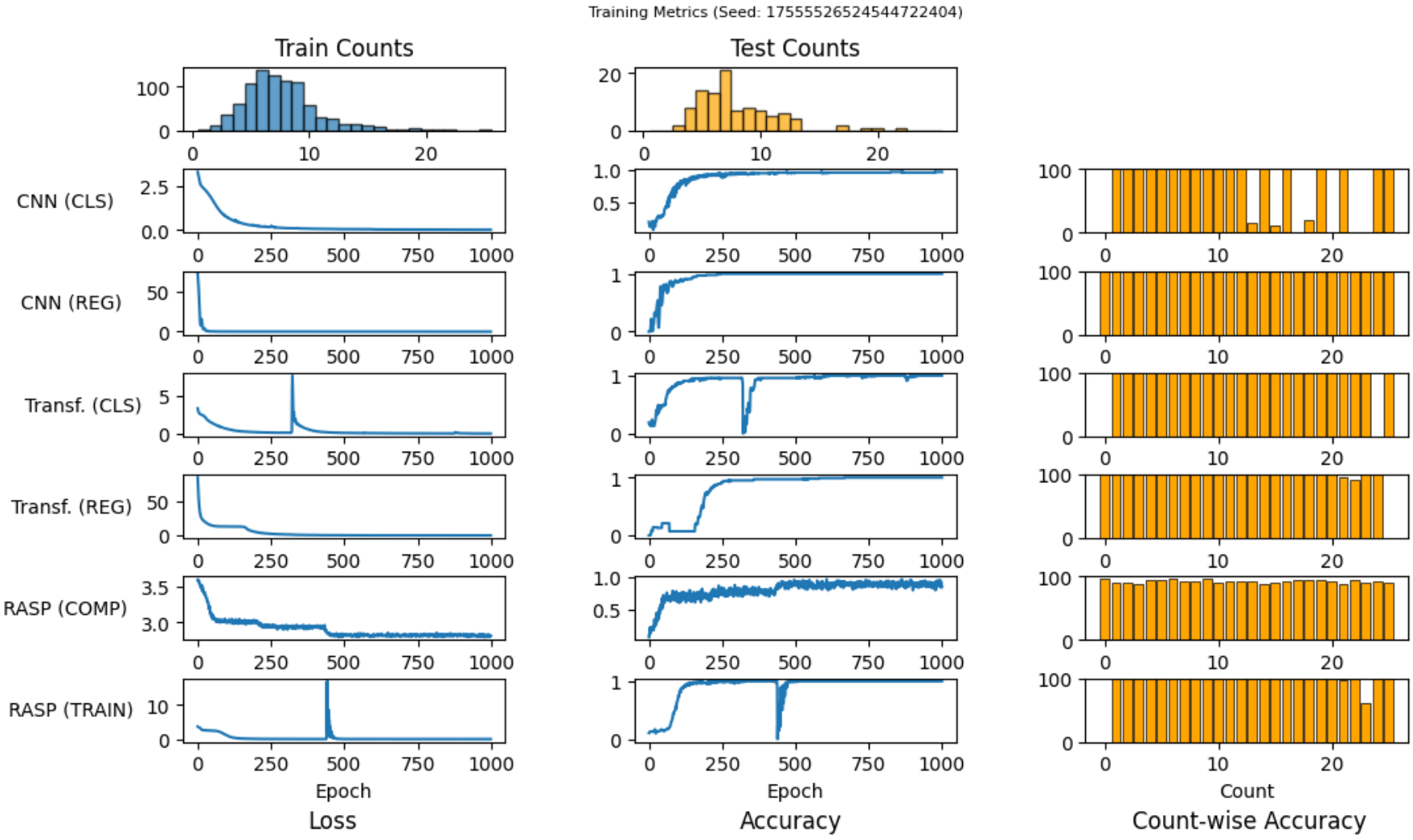

Fig. 14. Data run 3

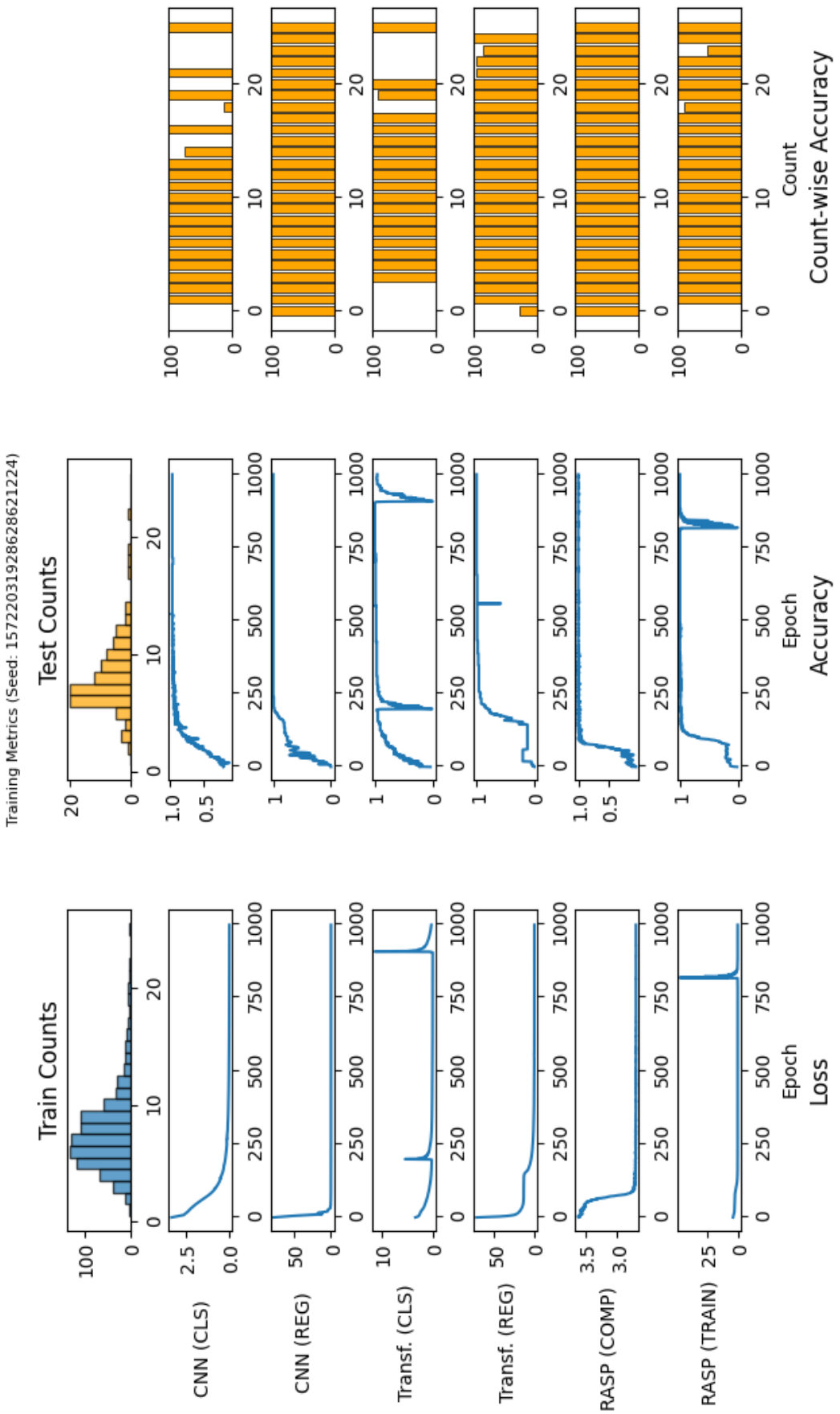

Fig. 15. Data run 4

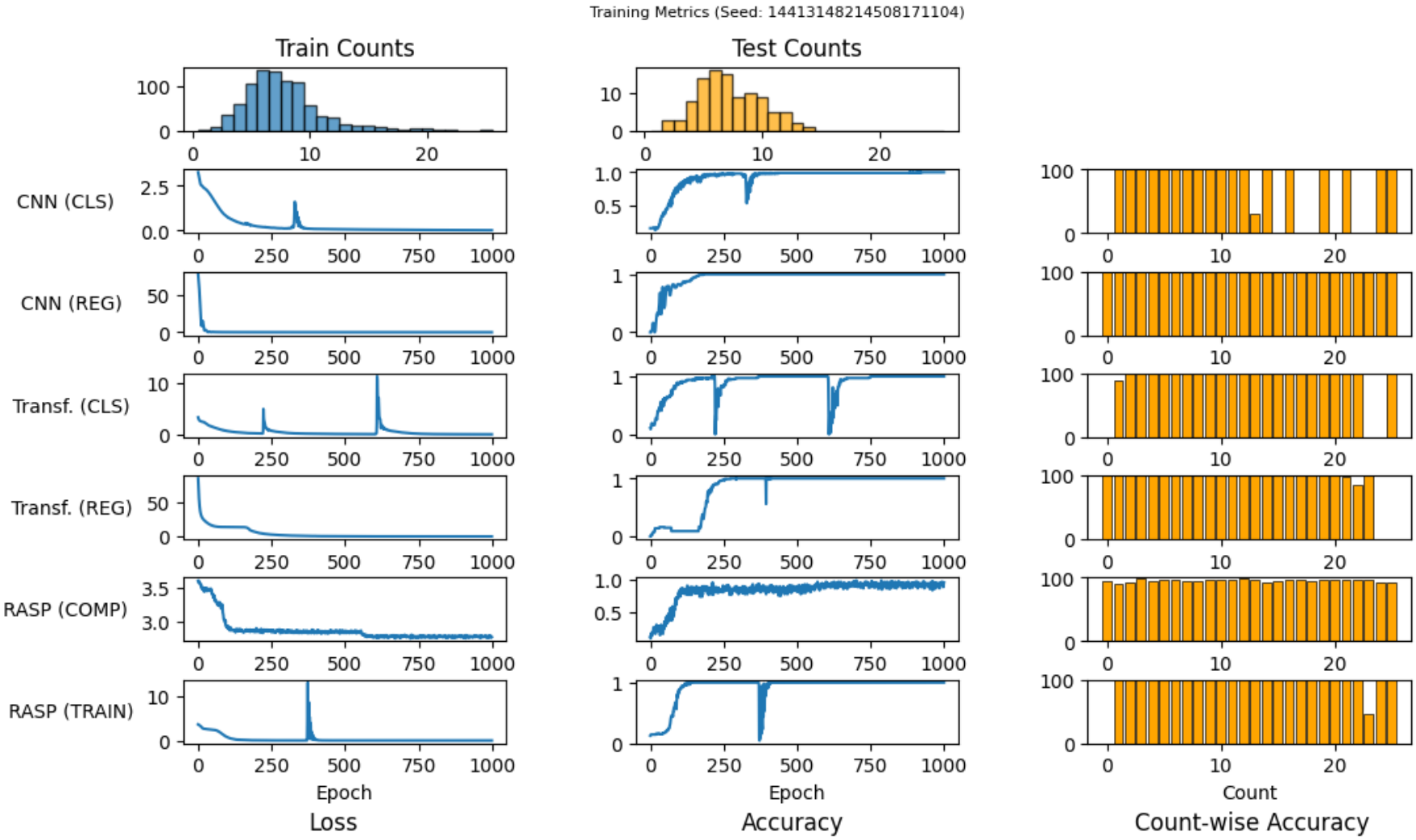

Fig. 16. Data run 5

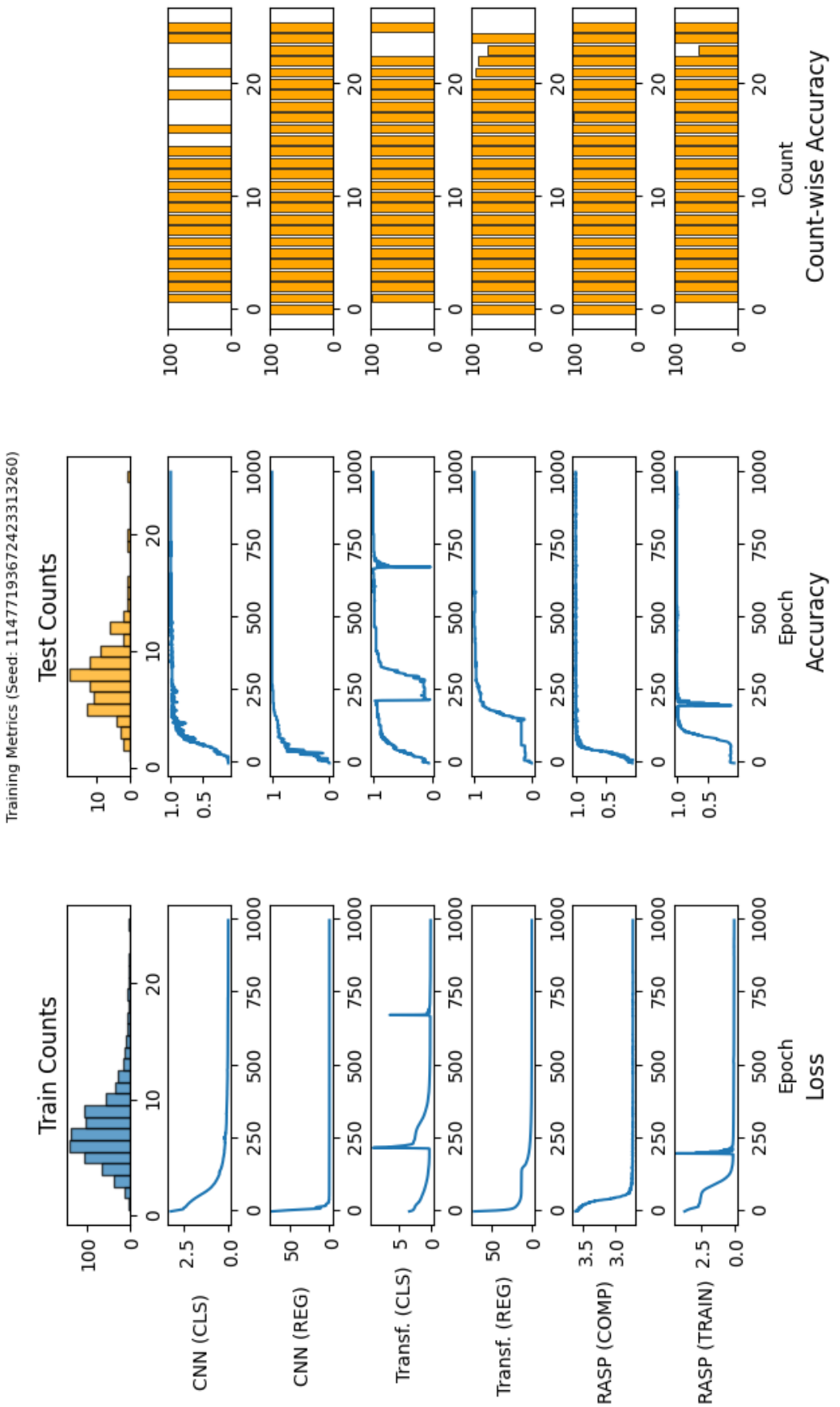

Fig. 17. Data run 6

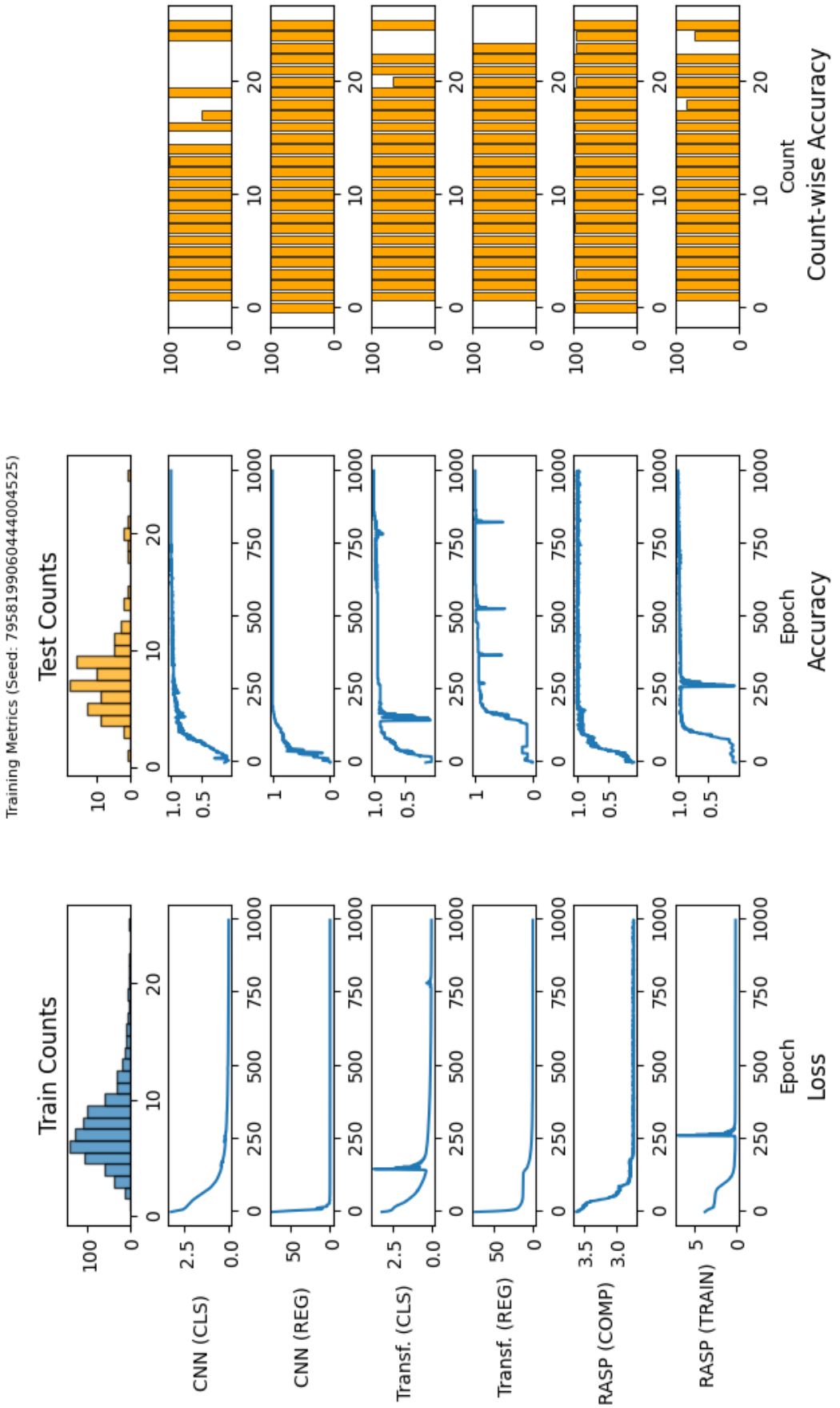

Fig. 18. Data run 7

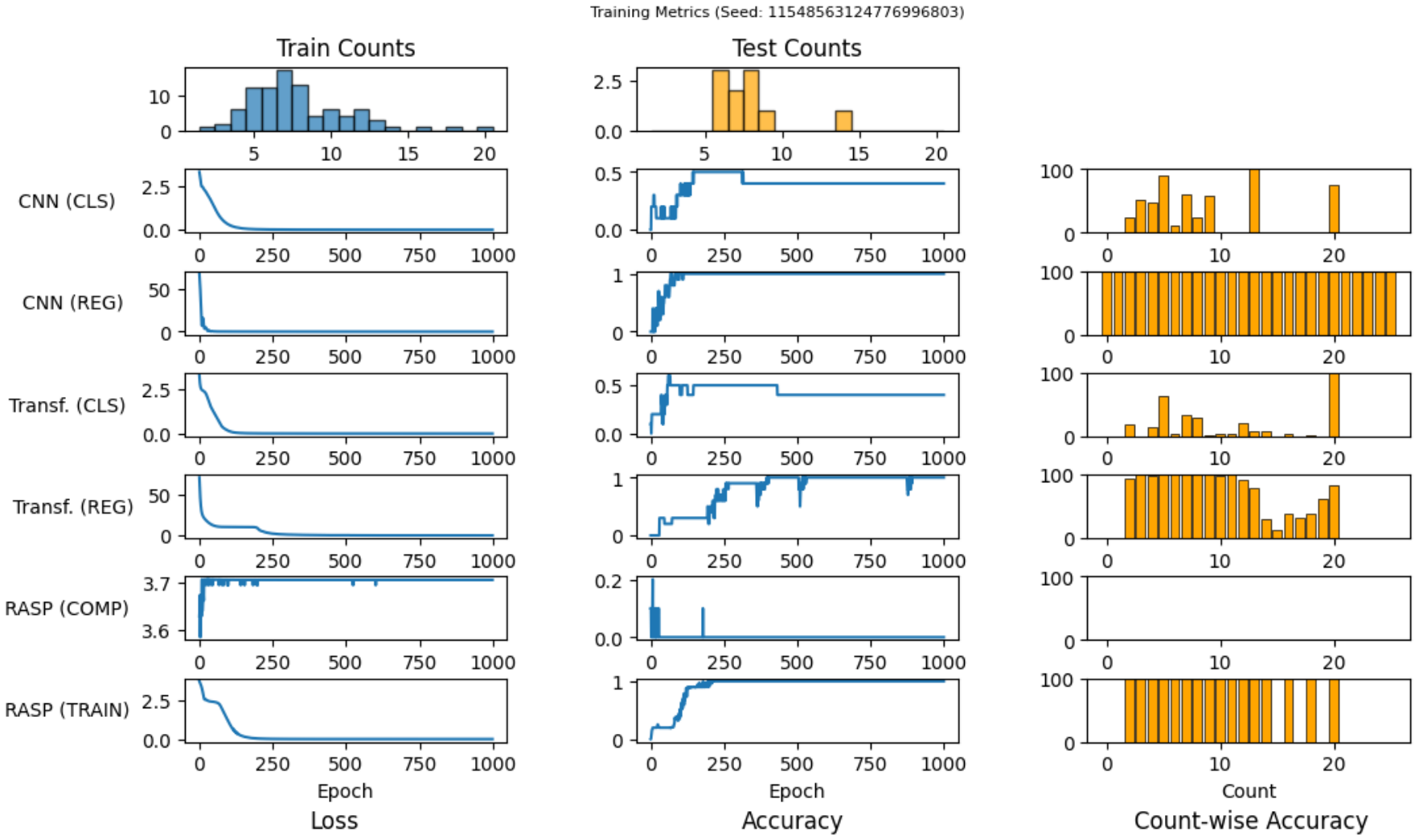

Fig. 19. Data for dataset size 100

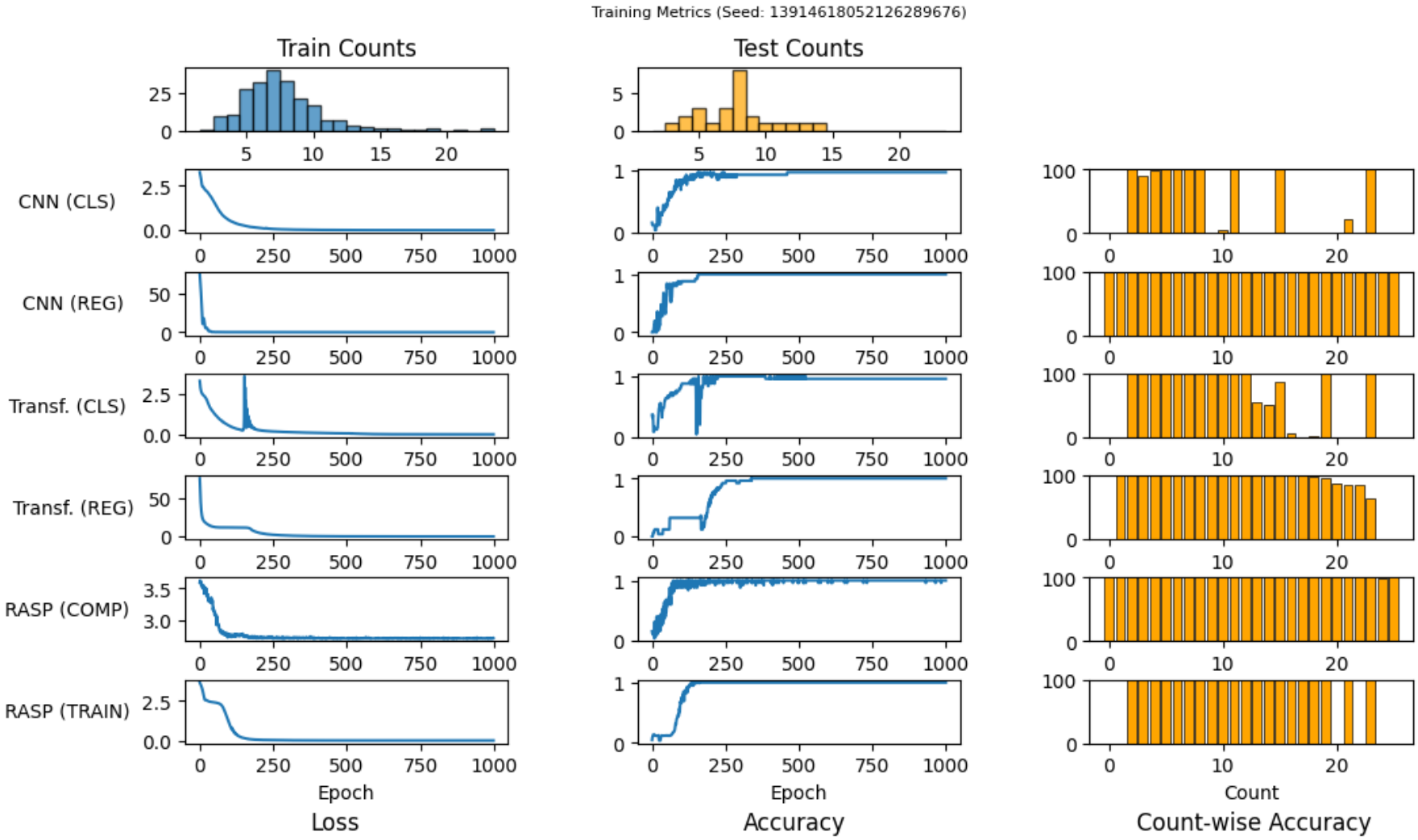

Fig. 20. Data for dataset size 250

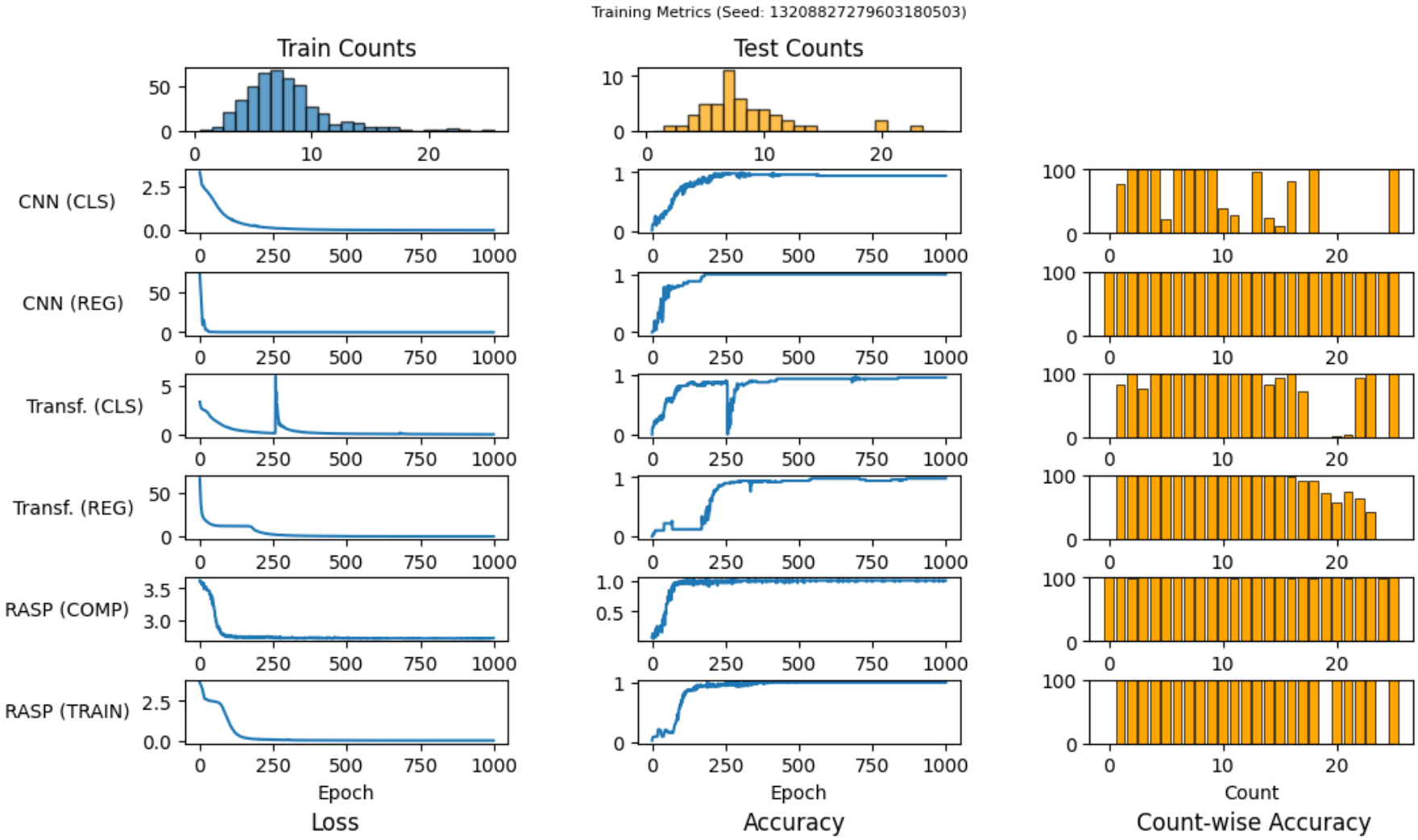

Fig. 21. Data for dataset size 500

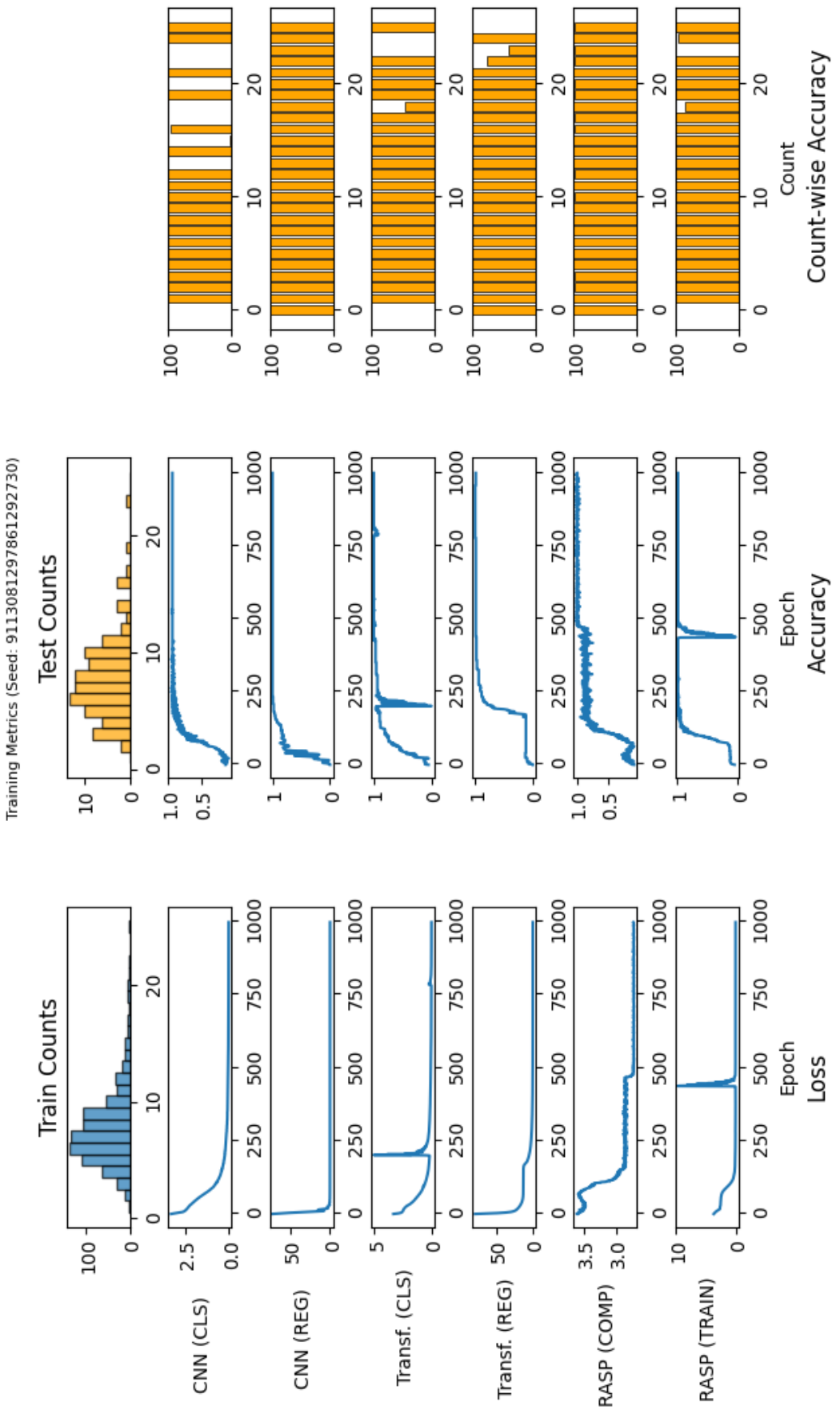

Fig. 22. Data for dataset size 1000