Exploring Test Time Adaptation on the Indonesian Presidential 2024 Election Dataset

M Eng Project, Computer Science and Software Engineering

Dwikavindra Haryo Radithya
Student ID : 249533
Supervisor: Dr Abhirup Ghosh

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
2024-2025

Word Count: 12117

Abstract

This study explores the usage and effectiveness of Test Time Adaptation (TTA) methods for improving handwritten digit recognition in the context of 2024 Indonesian Presidential Election. The study utilizes two approaches: (1) Test Time Adaptation by Entropy Minimization (TENT) and Source Hypothesis Transfer (SHOT) in adapting pre-trained MNIST LeNet models to real world election tally form under condition of an imbalanced dataset with significant distribution shifts. The study uses a reverse engineering approach on the government's SIREKAP android application to evaluate pre-processing, post-processing, and model inference methodologies. Experiments with various Batch Normalization Layers placement on LeNet-5 models revealed that such placement plays a huge role into TENT's adaptation performance, with our best configuration achieving 74% accuracy. SHOT showed higher potential with accuracy up to 79%. However, it exhibited instability with fluctuating performance across different source domain adaptation dataset sizes. Our findings found that both method is able the government model found in SIREKAP application exhibiting performance that achieved 18% accuracy during evaluation on our election dataset. However, both performances are highly influenced by hyperparameter sensitivity. This study provides comprehensive insights for improving OCR models deployed in critical civic applications and highlights the challenges of implementing test-time adaptation in real-world scenarios.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr Abhirup Ghosh for his unwavering support during the process of this study. His guidance and kindness have helped me overcome the obstacles during all the processes of this study. I would also like to thank my family: Ir. Pramasaleh Haryo Utomo, MT. and Neni Widaningrum for the unconditional love and support throughout the process of this study. Additionally, I thanked my friends for the constant reminders and support to give the best of efforts for this study.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Background and Motivation

Digit recognition is a common and classic problem in OCR (Optical Character Recognition) application and pattern recognition dating back to early influential works such as LeCun et al. [1] proposal of doing handwritten recognition using the LeNet model architecture. Subsequently, this problem has since been used to benchmark many models [2] with modern approaches achieving accuracy of 99% against multiple dataset benchmarks such as MNIST, USPS digit dataset [3], [4].

Despite the success, real world application of digit recognition technology continues to present significant challenges. An example of this is presented in the 2024 Indonesian Election, which employed a digital mobile application called SIREKAP, to facilitate the effort in digitalizing election tallying forms [5]. The election comprises of 823,236 polling stations, potentially involving millions of handwritten digits. The sheer scale and the variability that real world data presented, made the application's OCR model in digit recognition misclassify digits when attempting to scan the tally form. This error is further worsen when predictions from the model's OCR are not able to be manually corrected by election officers [5]. According to the General Elections Commission of Indonesia (*Komisi Pemilihan Umum Indonesia*), the technical issue impacted 1,233 polling stations nationwide, that represented 0.2 percent of the 586,646 polling station that had transmitted result as of 19 February 2024, 11:25 PM Western Indonesian Time (WIB) [5]. This problem extended itself to worsening public perception as negative sentiment from the social media platform X reached 85% from 90,380 tweets [6]. Subsequently a public court hearing ,involving government appointed experts, revealed an accuracy of 99% under test conditions [7]. The experts from the public hearing also acknowledged that under real-world scenario could deteriorate this number to approximately to 93% due to variations in handwriting style, lighting conditions, and image quality [7].

In machine learning studies this problem is described to as distribution shift where training distribution differs from test distribution [8] causing substantial degradation of accuracy in machine learning systems. For example, a model trained on recognizing whether a given picture is a dog or not, may misclassify when given similar looking animals like dogs that has wolf-like attributes (e.g. Siberian Husky, or various breed of wolfdogs). To address this issue a traditional approach would require retraining the models as new data emerges, however this approach is computationally intensive and requires labelled data that may cause privacy concerns as users may not consent to their data being used for training purposes, and potentially lead to models opting for datasets with less features/modalities to address the issue in return causing performance degradation [9]. Koh et al. [8] attributed this problem as underrepresentation in datasets widely used by the machine learning community. In response to this problem, a novel method called test-time adaptation started emerging in 2020 [10] with initial works in the field done by [11], [12]. Test time adaptation retrains only parts of a neural network layer during inference process by making use of unlabelled data [13] and unsupervised learning techniques. By using this approach test-time adaptation mitigates both the intensive computational issues and privacy concerns.

1.2 Project Aim and Goal

Considering the potential of test-time adaptation can have on the Indonesian Presidential Election process, the project aims to do the following:

- Investigate the government's OCR implementation processes by reverse engineering the SIREKAP application, evaluating the methodological approach and model performance.
- Evaluate and investigate the feasibility of two methodologies of domain adaptation namely Test-Time Adaptation by Entropy Minimization (TENT) [11] and Source Hypothesis Transfer (SHOT) [14] with data distribution shift presented by the 2024 Indonesian Presidential Election.

1.3 Ethical Consideration

This research adheres to ethical and regulatory standards for AI systems processing consumer data in compliance with the General Data Protection Regulation (GDPR). All dataset information collected in Section 4.1.2 are publicly available data available through the official recapitulation / tally website [15] provided by the General Election Commission of Indonesia with no personally identifiable information that is processed, or stored beyond publicly accessible content. Efforts in reverse engineering are also made through publicly downloadable android application [16], following publicly available analysis [17] strictly adhering to guidelines outline in responsible disclosure practices, ensuring no unauthorized or harmful activities were conducted on the original application or servers . Test Time Adaptation (TTA) methodologies ensures privacy by only using unlabelled data with no access to source data , where it uses only statics information gained from the data a model receives [13] . The study ensures transparency by clearly documenting methods, process, and data usage providing reproducibility and accountability.

Note [17] is written in the language *Bahasa Indonesia.* Translation to *English* as shown in Appendix 3 for modern browsers (e.g. Microsoft Edge) is verified to be accurate.

Chapter 2 Project Management and Planning

The study is carried out as a project that started in September 2024 until April 2025 (8 months). The project is carried out in 3 phases: (1) Project Initiation, Planning, and Research on Methodology (2) Data Processing (Preparation, Feature Extraction, and Data Analysis) (3) Implementation (Reverse Engineering, Model Development and Training, Evaluation of SHOT and TENT, and Report Writing)

The project began with initiation and planning with the supervisor Dr. Abhirup Ghosh where we discuss on possible project ideas and narrow them down into topics we could work on. This discussion went on throughout the project and is carried out every 2 weeks to ensure important milestone and blockage are completed. In parallel we discussed the methods that would be experimented with our project and start reviewing the dataset from [15].

Feature Extraction was then initiated where the process of data collection was eventually defined in Section 5.1.2. This process took 3 months, and in January 2025 we were able to conduct a data analysis on the election dataset we collected in Section 5.1.2.

In Parallel we started the implementation phase, from reverse engineering and evaluation to model development and training, we made sure to research more on the methodology eventually narrowing down to TENT and SHOT respectively, despite this, research on methodology was still going in parallel as consideration of adding more methods was considered but not pursued further due to time constraints and inspector's advice by Professor Russel Beale. SHOT and TENT evaluation was then conducted where in parallel report writing was also conducted. Figure 2.1 defined a GANNT chart defining the overall timeline of major phases.

Project Management GANNT Chart

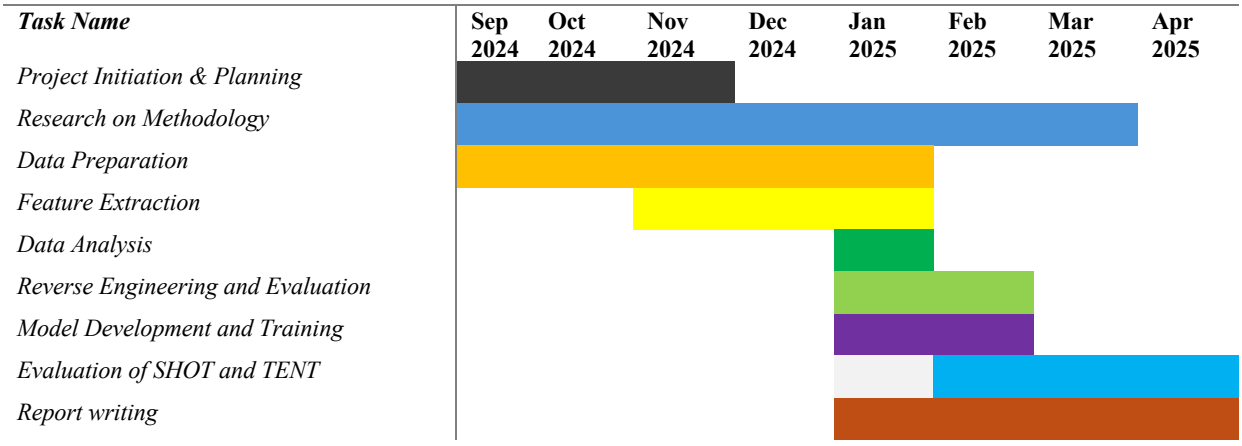


Figure 2.1 Project Management GANNT Chart

Chapter 3 Literature Review

In this chapter we provide a comprehensive understanding domain shift being the core problem of test time adaptation, subsequently we identify multitude of approaches, and the limitations of TTA methods.

3.1 Understanding Domain Shift

Classical machine learning approach assumes test and training data to come from the same distribution with *i.i.d* properties (independent and identically distributed random variables). Identically distributed defines distribution where there is no fluctuation of data, and all data is taken from the same probability distribution. Independent defines that all samples are independent and not affecting each other result. In our problem, new digits taken from the SIREKAP application is independent because each handwritten digits scanned does not affect how the other digits are written (e.g. writing the number 1 in one box on one candidate tally form does not mean that the next box for the same candidate would yield 2 or a predictable sequence of number). However, our issue comes from identical distribution, where incoming data from real world digit scanning can have high variance in features or class imbalance. This why techniques such as pre-processing, post-processing, and sampling techniques are employed to ensure new data is as close as possible to the distribution that the model is trained on.

According to Farahani et al. [18] domain shift can be categorized into 3 classes

- **Prior shift or class imbalance**:
  Occurs when the frequency of certain labels dominates over other classes (e.g. For our case in the election, imagine the images with labels classified as the digit "1" appears more 100 times whereas images with classified as the digit "4" appears only 20 times). Formally this problem is described when probability distribution $P_s(y|x)$ are equivalent to probability of target distribution $P_t(y|x)$, however prior distribution of classes between domains $P_s(y), P_t(y)$ are different.

$$P_s(y) \neq P_t(y), \quad P_s(y|x) = P_t(y|x),$$

- **Covariate shift**:
  Occurs when input data *looks different* (e.g. image a degree of blur, image quality is not what the model was trained on). Formally this problem refers to a problem when marginal probability of source $P_s(x)$ differs to $P_t(x)$, while conditional probability remains constants across target $P_t(y|x)$ and source domain $P_s(y|x)$.

$$P_s(x) \neq P_t(x), \quad P_s(y|x) = P_t(y|x)$$

- **Concept shift**:
  Occurs when input data might *look the same,* but the meaning or labelling has changed (e.g. people might have different writings style that can make numbers like "1" and "7"). Formally this problem happens when data distribution of source $P_s(x)$ and target domain $P_t(x)$ remains unchanged while conditional distribution differs between source $P_s(y|x)$ and domain $P_t(y|x)$ distribution.

$$P_s(x) = P_t(x), \quad P_s(y|x) \neq P_t(y|x)$$

3.2 Test Time Adaptation in Solving Domain Shift

Test Time Adaptation is one the method attributed solving a domain shift issue. Liang et al. [19] formally categorize test time adaptation based on the characteristics of test data:

- **Test Time Batch Adaptation (TTBA):**
  Adapts a model by using mini batch of test samples during testing. Each batch is adapted independently, and prediction from each batch of data does not influence other adapted model. The method involves recalibrating Batch Norm (BN) statistics, to optimize model parameters.
- **Online Test Time Adaptation (OTTA):**
  Adapts the model to a stream of test samples sequentially. Updates a model based on previously seen samples to adapt new one. The method involves technique of entropy minimization, pseudo labelling, and regularization to address problems of overfitting (catastrophic forgetting) and error accumulation.
- **Test Time Domain Adaptation (TTDA):**
  Adapts the model by performing a one-time, offline adaptation using the entire unlabelled target domain, without access to source data.

3.3 Characteristics of TTA Methodologies

Despite test time adaptation promising methodologies, Zhao et.al [13] shows limitations.
Their comparative analysis reveals several key challenges for both method:

- **Hyperparameter sensitivity**

TENT and SHOT sensitivity with [13] testing against the CIFAR10 – C. In contrast with Wang et al. [11] finding when learning rates are increased along with adaptation steps, worse outcomes are seen with accuracy performance dropping from to 10-20% for both TENT and SHOT.

- **Batch Order (Index) Dependency**

[13] revealed SHOT performs under online adaptation across different batch indices (i.e., the order in which data samples appear in a batch). [13] observed a drop in accuracy when samples with higher batch indices are processed, suggesting that the model adapts better (b) with the help of increasing steps, but initial runs (a) suggested d that performance drop. In [13] this figure is used as to show how TTA adapts under different index ordering , however it only showed figures done on SHOT's online implementation with supporting theory that since TTA methods relies on batch data the reorder of these dataset would make these methods far susceptible to changes in order.

- **Quality of Pretrained models**

[13] also shows models that are trained using data augmentation techniques like AugMix and PixMix already shows high performance metrics before TTA methods. [13]compares OOD (Out-of-Distribution) accuracy performance which measures model effectiveness on data outside the trained distribution and ID (In-Distribution) accuracy performance which evaluates accuracy on data similar the training distribution. Whilst TTA method improved model's accuracy performance slightly. This counterintuitive finding reveals that models with stronger inherent out-of-distribution generalization benefit less from test-time adaptation. While data augmentation techniques improve a model's robustness to distribution shifts, they simultaneously reduce the potential gains from TTA method.

Chapter 4 Introduction to Test Time Adaptation

4.1 Test Time Adaptation by Entropy Minimization (TENT)

TENT objective is to align target domain features by updating a model normalization layer affine parameters across batches, whilst minimizing entropy via parameter modulization. In the case of digit classification, this means increasing a model's *confidence* in assigning a given image belonging to a specific class (e.g. increase probability for the class with the highest probability)

- **Objective Function**

TENT's mathematical objective is to minimize entropy via feature modulation, where entropy is often attributed to minimizing disorder or uncertainty. As written in [11] , TENT utilizes Shannon Entropy $H(\hat{y})$ to define its loss function $\mathcal{L}_{tent}(x_t)$ for model prediction $\hat{y} = f_\theta(x_t)$

$$\mathcal{L}_{tent}(x_t) = H(\hat{y}) = -\sum_c p(\hat{y_c}) \log p\,(\hat{y_c})$$

where $\hat{y} = f_\theta(x_t), p(\hat{y_c})$ is the predicted probability for class c (1)

However, it should be noted that only optimizing Shannon's Entropy would make the model too confident in predicting one class, to prevent this TENT utilizes joint optimization over batched predictions, with parameter's that are shared across batches it receives.

- **Parameter Modulation**

To minimize the effect of overconfidence in minimizing its entropy, TENT uses two types of parameters it describes as normalization statistics and affine parameters for all layers and channel during testing. Note that in implementation these statistics are only found in neural networks with Normalization Layers (e.g. BatchNorm2D, BatchNorm1D, LayerNormalization in pytorch library) [11] explains that TENT only enact parameter modulation on features that are linear (e.g. scales and shift), and low dimension. In the case of TENT this enacted by estimating mean μ and standard deviation σ from the input x for normalization $\bar{x} = \frac{x-\mu}{\sigma}$which is then transformed into $x' = \gamma\bar{x} + \beta$ where affine parameters γ (gamma) and β (beta) are optimized by the loss. These affine parameters are then shared across batches for stability and efficiency.

In Wang et al. [11] work , they attributed TENT's stability and efficiency, after testing other adaptation methodologies RG (adversarial domain adaptation algorithm by [20]), UDA-SS (self-supervised domain adaptation by [21]),TTT (test time training by [22]), BN (test-time normalization that updates batch normalization statistics by [23]) on a ResNet-26 model trained with the SVHN dataset and tested against MNIST/MNIST-M/USPS dataset where TENT is able to show the lowest rate of error. They also attributed TENT stability and efficiency after testing against a corrupted CIFAR 10 and CIFAR 100 dataset called C10/100 C (C for corrupted).

- **Requirement**

Based on the explanation above TENT would require a pre-trained model for a supervised task that is probabilistic and differentiable where deep networks for supervised learning typically satisfy these requirements, if there is a Normalization Layer. This satisfies our digit recognition problem where models were trained on MNIST and is expected to classify test set by choosing the highest probability presented.

4.2 Source Hypothesis Transfer for Unsupervised Domain Adaptation (SHOT)

SHOT objective is to align target domain features to the fixed decision boundaries learned from the source domain, without requiring access to the source data during adaptation. SHOT does this by freezing the classifier (also referred to as the hypothesis set (h_s) and only updating the feature encoder g_t using unlabelled target data. The adapted model is written as $f_t(x_t) = h_s\big(g_t(x_t)\big)$. In digit classification, this means mapping handwritten digits from the target domain (e.g. scanned electoral digits) into the same regions the model used to classify clean MNIST digits.

- **Objective Function**

SHOT mathematical objective are defined into two steps: (1) minimization of loss adopted from information maximization (IM by [24]) , (2) applying SHOT's novel pseudo labelling with the usage of centroid, and cosine similarity distance. The first step is implemented by doing the same minimization of Shannon Entropy Minimization in TENT $\mathcal{L}_{ent}$, however as explained before this is not enough as it may lead to overconfidence in predicting one class, as such another loss function $\mathcal{L}_{div}$ is introduced to promote *fair diversity* by ensuring all unlabelled data does not have the same one hot label encoding written in [14] :

$$\mathcal{L}_{ent} = -E_{x_t \in \mathbb{D}_t} \sum_{k=1}^{K} \delta_k\big(f_t(x_t)\big) \log \delta_k\left(f_t(x_t)\right) \quad (2)$$

Note that δ_k here equals to $p(\widehat{y_c})$ in TENT (Section 3.1)

$$\mathcal{L}_{div} = \sum_{k=1}^{K} \overline{p_k} \log \overline{p_k} \quad \text{where } \overline{p_k} = E_{x_t \in \mathbb{D}_t}\big[\delta_k\big(f_t(x_t)\big)\big] \quad (3)$$

The second step is to use a SHOT's novel pseudo labelling with centroid and cosine similarity, this approach is enacted because using only information maximization (IM) leads to certain datapoints still being clustered incorrectly.

This is described to as self-supervised pseudo labelling loss $\mathcal{L}_{ssl}$:

$$\mathcal{L}_{ssl} = -E_{(x_t, \widehat{y_t})} \sum_{k=1}^{K} 1[k = \widehat{y_t}] \log \delta_k\left(f_t(x_t)\right) \quad (4)$$

$$\mathcal{L}_{SHOT} = \mathcal{L}_{ent} + \mathcal{L}_{div} - \beta\mathcal{L}_{ssl}$$

where β is an arbitary number , $(\beta \in (0,1))$ (5)
that increases confidence in self supervised pseudo labeling loss

- **Centroid Similarity**

To minimize the wrong clustering produced by Information Maximation (IM) SHOT uses a centroid approach that involves: (1) initializing centroid $c_k^{(0)}$, (2) pseudo label assignment with cosine distance $\hat{y}_t$, and (3) updating centroid for subsequent calculations.

$$c_k^{(0)} = \frac{\sum_{x_t \in X_t} \delta_k\left(f_t(x_t)\right) \cdot g_t(x_t)}{\sum_{x_t \in X_t} \delta_k\left(f_t(x_t)\right)}$$

$$\hat{y}_t = \arg\min_k D_f\left(g_t(x_t), c_k^{(0)}\right), where\ D_f\ is\ the\ cosine\ similarity \quad (6)$$

$$c_k^{(1)} = \frac{\sum_{x_t \in X_t} 1[\hat{y}_t = k] \cdot g_t(x_t)}{\sum_{x_t \in X_t} 1[\hat{y}_t = k]} \quad (7)$$

- **Requirement**

.

Based on the explanation above, SHOT would require a pre-trained model with a specific architecture that separates the feature encoder from the classifier (hypothesis). The model must be trained on a source domain in a supervised manner and should be both probabilistic and differentiable. Unlike TENT which only updates normalization layer parameters, SHOT requires the ability to freeze the classifier (transfer weight only) while updating the feature encoder during adaptation (retrain backbone, fully connected, and bottleneck) as shown in Figure 4.1. This makes SHOT suitable for our digit recognition problem where the model is trained on source domain (e.g., MNIST) and needs to adapt to target domain (e.g., scanned electoral digits) without access to the original training data, by maintaining the same classification boundaries while adapting how features are encoded.

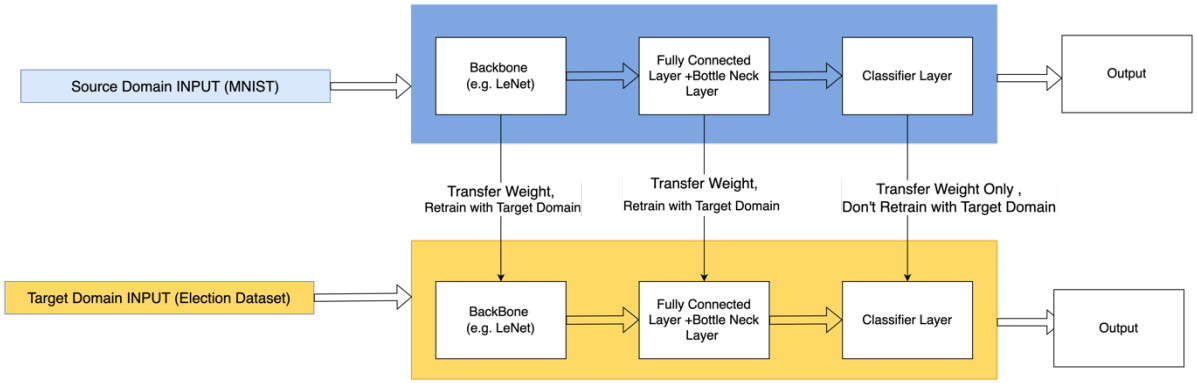


Figure 4.1 SHOT Framework

Chapter 5 Experiments

This chapter is split into two main parts, Section 5.1 - Section 5.2 and Section 5.3 - Section 5.5. Section 5.3 - Section 5.5 is focused on the datasets used by the study and the reverse engineering process and evaluations conducted. The results from Section 5.1-5.2 provides justifications, for subsequent experiments settings and choices in Section 5.3 – Section 5.5 for evaluating performance of TENT and SHOT. Results and Discussions from Section 5.4 and Section 5.5 is presented in Chapter 6.

Summary of Experiment Sections (in order of execution)		
Experiment	**Section**	**Goals**
1. Reverse Engineering for Justifying Section 5.3 – 5.5		
Dataset Exploration	Section 5.1	1. Insight on the MNIST dataset 2. Insight on the Election Dataset (Dataset Collection, Extraction and Analysis)
Reverse Engineer	Section 5.2	1. Evaluate SIREKAP application Model on MNIST and Election Dataset 2. Evaluate Pre-Processing and Post-Processing method of the SIREKAP application.
2. Defining Evaluation Process and Architectural Choices for Chapter 5 (Results and Discussion)		
Baseline	Section 5.3	1. Define the architecture of TENT and SHOT
Performance Evaluation	Section 5.4	1. Define Evaluation Protocol of TENT and SHOT
Evaluation Metrics	Section 5.5	1. Define Evaluation Metrics

Table 5.1 Chapter 5 Summary

5.1 Datasets

TTA (Test Time Adaptation) requires 2 different datasets, the source and target datasets respectively called source domain and target domain. Source Domain represents the dataset that the model is trained with whereas the target dataset represents the dataset that TTA should adapt the model to. The following datasets were utilized in this study:

5.1.1 Source Domain : MNIST

The MNIST dataset was selected based on findings from Experiment 5.2 which indicated that the SIREKAP application utilizes ensembled models trained on MNIST. As such this dataset is subsequently used as the source domain in all experiments conducted to replicate the model reportedly used by the government in the SIREKAP application during the 2024 Indonesian Presidential Election. MNIST contains 60,000 training images and 10,000 test images of 28x28 pixel grayscale handwritten digits (0-9) [1]. The LeNet models were exclusively trained on the MNIST dataset and is also used for evaluating the performance before and after TENT and SHOT is applied.

5.1.2 Target Domain : 2024 Indonesian Presidential Election Dataset

The target domain consists of manually cropped digits collected from the election tally forms (Form C1) used in the election to represent real world scenario with potential domain shift.

- **Source:**

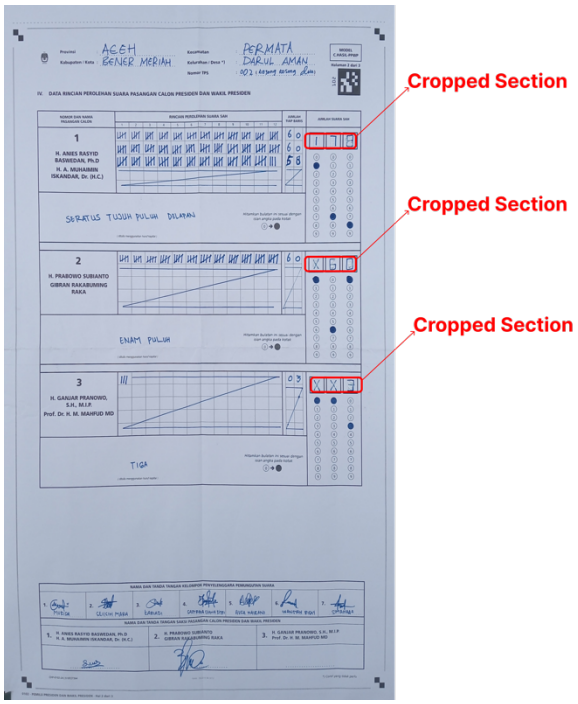


Figure 5.1 C1 Form Example gained publicly from [15]

The digit data was extracted from Form C1as seen in Figure 5.2 , obtained from the publicly available data from the General Elections Commissions Website Commission website [15]. There are a total of 3 candidates where each candidate on each C1 form has at maximum 3 boxes to write an input. Each can contain one of three values: (1) A handwritten digit (0-9), (2) an "X" (cross) should be used to represent unfilled values, (3) blank space. The official guidance instructed officers to write "X" to represent unfilled values, however we found many boxes were left with a blank space as seen in

- **Collection:**

An estimated 823,366 ballots [25] each associated with a corresponding C1 Form [26] were available during the 2024 election. Although the General Elections Commissions has not officially published this figure, it is widely cited by verified crowdsourced platform [26] and independent source [25]. Given that each candidate has 3 boxes per form , a maximum of 7,410,294 digits entries could be collected nationwide . To manage the scope of this project, we collected 200 forms collected across 30 provinces spanning 165 regencies, 187 districts, and 186 subdistricts to ensure diversity of across regions. The images were then manually downloaded and stored.

- **Extraction:**

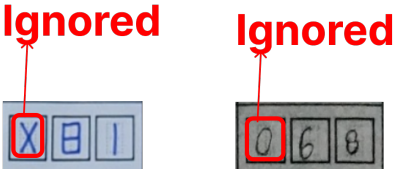


Figure 5.2 Example of Not Cropped Input from C1 Form

Form each C1 form, sections containing handwritten digits were manually cropped. Our efforts ensure the boxes were fully cropped out, although on some pictures this was not possible, due to differing tilt positions of the scanned C1 Form. Each digit were then saved

with additional information including ballot id, province, regency, district, sub district, and candidate number. Boxes that were filled with "X", was left completely blank, or began with a leading zero in the first box was not included. (e.g. for the input '028','0' is ignored only '2' and '8' is extracted) (Refer to Figure 5.2)

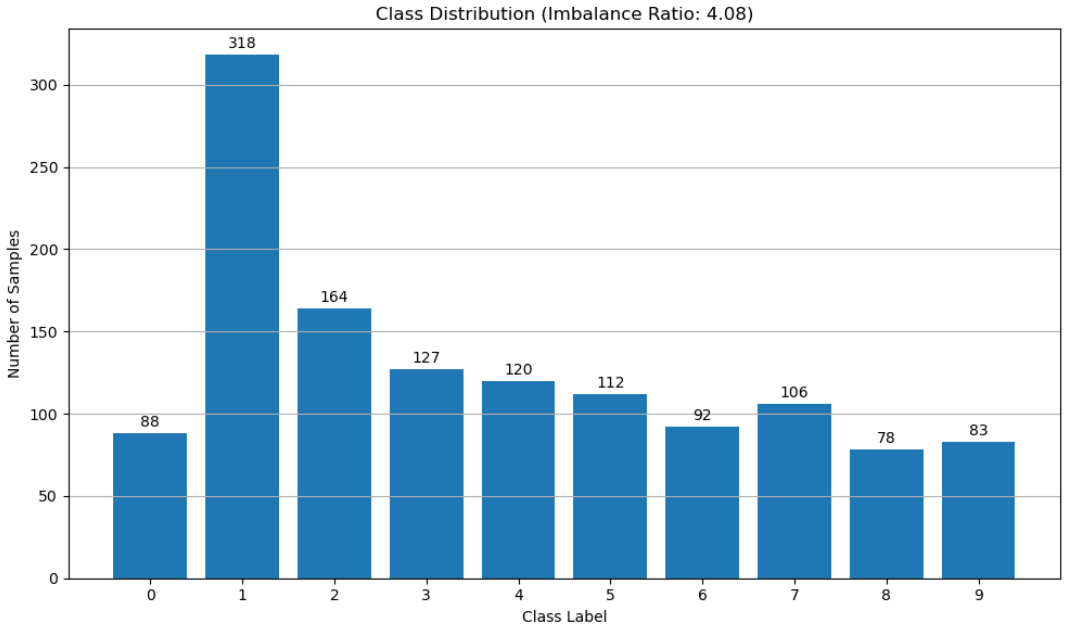


Figure 5.3 Class Imbalance from Election Dataset

A total of 1,288 digits were extracted using the process mentioned. Our analysis in class imbalance on Figure 5.3, showed the dataset exhibited a significant class imbalance. A class imbalance occurs with the digit '1' being the most frequent and the digit '8' being the least frequent. The imbalance ratio of the most, the imbalance ratio of the most frequent and least digit is shown to be considerably high at 4.08. This was originally considered as an issue as TENT [11] and SHOT [14] assumes a balanced dataset when adapting. However, after consultation, the decision was made to proceed with the current dataset as the current dataset size and imbalance provided a valuable and challenging test scenario to investigate the ability of TENT and SHOT adaptation method to generalize effectively on real-world, non-uniform data distributions and assess their robustness against potential overfitting to the limited and skewed target samples.

5.2 Reverse Engineering

5.2.1 Findings: Reverse-Engineered SIREKAP Application

To replicate the model's performance used during the real-world digit recognition process, the study applies reverse engineering by decompiling the SIREKAP Android application. The primary goals of the reverse engineering process are: (1) find and evaluate models that were used for the OCR process, (2) determine the image pre-processing methods. The reverse engineering methodology closely follows [17] .

We analysed the SIREKAP 2024 v2.4, released on February 14, 2024 which is the version that was released during the election downloaded from the *pureAPK* repository [16] and decompiled using the online decompilation tool [27]. Analysis of the decompiled code yielded several key insights. A surprising fact, revealed the application code was not obfuscated, allowing for a full examination of all the code and assets used.

- **Ensembled Models**:

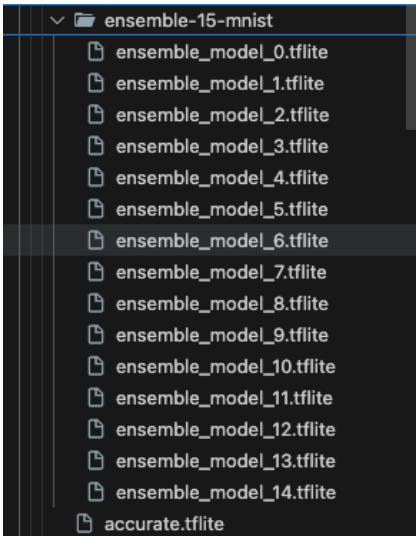


Figure 5.4 Reverse Engineered Folder Naming Indicating Ensembled Approach

```
for (Interpreter next : list) {
    Tensor inputTensor2 = next.getInputTensor(0);
    Tensor outputTensor2 = next.getOutputTensor(0);
    TensorBuffer createFixedSize3 = TensorBuffer.createFixedSize(inputTensor2.shape(), inputTensor2.dataType());
    Intrinsics.checkNotNullExpressionValue(createFixedSize3, "createFixedSize(inputDet… inputDetails.dataType())");
    createFixedSize3.loadBuffer(allocateDirect);
    TensorBuffer createFixedSize4 = TensorBuffer.createFixedSize(outputTensor2.shape(), outputTensor2.dataType());
    Intrinsics.checkNotNullExpressionValue(createFixedSize4, "createFixedSize(outputDe…outputDetails.dataType())");
    next.run(createFixedSize3.getBuffer(), createFixedSize4.getBuffer());
    for (int i3 = 0; i3 < 10; i3++) {
        fArr[i3] = fArr[i3] + createFixedSize4.getFloatArray()[i3];
    }
}
```

Figure 5.6 SIREKAP Application Ensembled Code

Pseudocode: Ensemble Model Prediction Aggregation

Step 1: Initialize `fArr[0 to 9]` as an array of zeros

Step 2: For each model in the ensemble:

- Input the image into the model using `inputBuffer`
- Receive the prediction scores from the model in `outputBuffer`
- Each index in `outputBuffer` corresponds to a class, and its value represents the model's confidence for that class.
  (e.g., index 0 (for value indicating '0') = 2.33, index 4 = 4.00 (for value indicating '4'), etc.)
- **For** $i = 0$ **to** 9:
  - Update the final score: `fArr[i] = fArr[i] + outputBuffer[i]`
    (Accumulate the score for each class across all models)

Figure 5.5 SIREKAP Models Ensembled Pseudocode Approach from Figure 5.4

The application uses an ensembled approach, using 15 ensembled models for digit recognition suggested by Figure 5.4 with a folder named "ensembled-15-mnist" alongside code shown by Figure 5.6 that the models were trained using the MNIST dataset. As such the model performance is validated against MNIST test dataset in subsequent experiments.

We tried visualizing these architecture using a publicly available tool used for analysing a compiled model called 'Netron' [28], however visualizing the architecture

proved to be challenging as *Netron* doesn't provide an easily understandable visualization on the specific architecture used, an example can be seen in *Figure 5.6* .

- **Model Format** :
  The models are stored in Google's TensorFlow [29] Lite files (.tflite), a format common for mobile deployment due. However, our process and evaluation use Meta's Pytorch [30] library with the Pytorch Tensor file (.pt) format thus there is a need to convert this model into the same process for evaluation. Using this information we converted the models by following the pipeline: (1) Convert the model to an ONNX [31]format, (2) Convert the ONNX format to Pytorch Tesnor (.pt) format for compatibility with the evaluation pipeline. However, Google's TensorFlow Lite typically uses quantization a process that represents computer bits in lower bit format from for 4-16 bits that provides easier computation in mobile devices.
- **Quantization Format:**
  To determine whether the models use quantized format we converted the models using ONNX format. Inspection using Netron [28] Figure 5.6 indicated that the models operate using 32 bit and are not quantized thus all evaluation models used in subsequent experiments does not need to convert to quantized format and the previous model conversion method can be followed without worrying for loss of data.

Figure 5.6 Inspection on Netron[28] Showing Proof of No Quantization

- **Pre-processing Pipeline:**

```
Imgproc.resize(clone, mat2, new Size(28.0d, 28.0d));
Imgproc.adaptiveThreshold(mat2, mat2, 255.0d, 1, 0, 11, 7.0d);
Mat mat3 = new Mat();
Core.bitwise_not(mat2, mat3);
```

Figure 5.7 Pre-processing Pipeline Code

Our reverse engineering process also uncovers the pre-processing pipeline of the digits where images are: (1) Converted to Grayscale, (2) Inverted (to have a black background and reveal a white colour on the digit), (3) Applied Adaptive Thresholding using the open computer vision (open-cv) android library with code shown in Figure 5.7 explained in Table 5.2. For easier reference in future sections, we refer to this pre-processing pipeline as the 'SIREKAP Pre-Processing' method.

Adaptive Thresholding Parameters Found in the Reverse Engineered Pre-processing Pipeline			
Parameter	**Values shown in Comparison with Figure 4.X**	**Value**	**Description**
maxValue	255.0d	255.0	The value assigned to pixels meeting the threshold condition (white).
adaptiveMethod	1	cv2.ADAPTIVE_THRESH_MEANN_C	Computes threshold from the mean of the local neighbourhood.
thresholdType	0	cv2.THRESH_BINARY	Applies a binary threshold where pixels are set to either maxValue (white) or 0 (black).
blockSize	11	11	Defines the size (11x11) of the pixel neighbourhood used to calculate the local threshold (must be an odd number ≥ 3).
c	7.0d	7.0	A constant subtracted from the computed mean neighbourhood value to fine-tune sensitivity and control threshold aggressiveness.

Table 5.2 Explanation of the Pre-processing Parameters Found in the Reversed Engineering Pre-Processing Pipeline Code from Figure 5.7

- **Post-processing:**

Pseudocode: Heuristic-Based Class Selection

Input: `probabilities[0 to 9]` (Note code shows the parameter is named probabilities it is actually not and only the summation value from the ensembled process as shown in the code in Step 3 where "`probabilities[0]` ≥ 10"
Output: Selected class index (integer)

Step 1: Find the index of the class with the highest score

- Initialize `firstMax = 0`
- **For** $i = 0$ to 9:
  - **If** `probabilities[i] > probabilities[firstMax]`, update `firstMax = i`

Step 2: Find the index of the second-highest score

- Initialize `secondMax = 0`
- **For** $i = 0$ to 9:
  - **If** `probabilities[i] > probabilities[secondMax]` and `i` ≠ `firstMax`, update `secondMax = i`

Step 3: Apply custom heuristics:

- **If** `firstMax` ≠ 0 **or** `probabilities[0]` ≥ 10.0:
  - **If** `firstMax` ≠ 1 **or** `probabilities[1]` ≥ 14.5:
    - **If** `firstMax == 3` **and** `probabilities[3] < 10.0` **and** `probabilities[9] > 3.0`, **return** 9
  - **Else if** `probabilities[secondMax] > 0.0`, **return** `secondMax`
- **Else if** `secondMax == 8`, **return** `secondMax`

Step 4: Return `firstMax` as the default prediction

Figure 5.8 SIREKAP Post-Processing Pseudo Code Ensembled Pseudocode Approach

A heuristic-based post-processing method was implemented that modifies predicted probabilities for visually similar digits such as '9' and '3' shown in Figure 5.8. Note our decompilation result using the tool [27] failed to clearly show an understandable

code shown in Appendix 5, as such the code can be seen from [17] named 'applyHeuristic'

5.2.2 Evaluation Findings: Reverse-Engineered SIREKAP Models

Per Model Accuracy on Ensembled Models to Confirm Models were Trained on MNIST	
Model Name	**Accuracy**
ensemble_model_0	0.9983
ensemble_model_1	0.9980
ensemble_model_2	0.9984
ensemble_model_3	0.9983
ensemble_model_4	0.9982
ensemble_model_5	0.9980
ensemble_model_6	0.9984
ensemble_model_7	0.9986
ensemble_model_8	0.9984
ensemble_model_9	0.9983
ensemble_model_10	0.9979
ensemble_model_11	0.9982
ensemble_model_12	0.9981
ensemble_model_13	0.9988
ensemble_model_14	0.9984

Table 5.3 SIREKAP Individual Model Accuracy on MNIST

Per Model Accuracy on Ensembled Models on Election Dataset	
Model Name	**Accuracy**
ensemble_model_0	0.1172
ensemble_model_1	0.2671
ensemble_model_2	0.1762
ensemble_model_3	0.1211
ensemble_model_4	0.6879
ensemble_model_5	0.4107
ensemble_model_6	0.1289
ensemble_model_7	0.0854
ensemble_model_8	0.4720
ensemble_model_9	0.1234
ensemble_model_10	0.1398
ensemble_model_11	0.1095
ensemble_model_12	0.3758
ensemble_model_13	0.1273
ensemble_model_14	0.1366

Table 5.4 SIREKAP Individual Model Accuracy on Election Dataset

Ensemble Accuracy Tested	
	Accuracy
Ensemble Accuracy on MNIST	0.8823
Ensemble Accuracy on Election Data	0.1832

Table 5.5 Ensembled Model Accuracy

Note All Models in Table 5.3 - Table 5.4 uses the pre-processing and post-processing pipeline in Section 5.2.1. Despite independent models gaining performance on MNIST with high 99% Accuracy, ensembled approach gained 88% due to post-processing method as explained in Section 5.2.2 below "Resize Image".

To provide clarity regarding domain shift present in the election application ensembled models, we evaluate the ensembled models' performance on the source domain MNIST dataset and the target domain election dataset we collected. Table 5.3 confirms the models are likely trained with MNIST achieving results in the range of 99% accuracy. However, when evaluated on the election dataset, performance drops drastically, with individual models dropping to as low as 11% accuracy as shown in Table 5.4. In return the ensembled approach that the government provided also yielded poor results as shown in Table 5.5.
This is unexpected as government claims 93% accuracy on test condition previously mention in Section 1.1 from [7] despite us following the app's approach in pre-processing and post-processing the data previously explained in Section 5.2.1 as such we decided to conduct more experiments to: (1) Decide if the pre-processing used by the government is the issue, (2) Decide if the post-processing process used by the government is the issue. To decide if the government's pre-processing method causes these drops, we define 2 used additional pre-processing method namely:

- **Custom Pre-processing Method :**
  The following methodology is a customized method following [1] with the following pipeline :

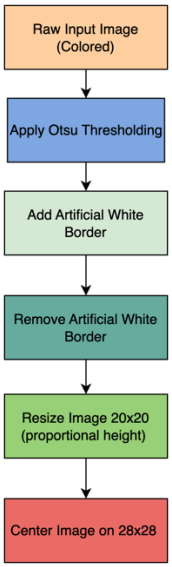


Figure 5.9 Custom Pre-processing Method

Custom Pre-processing Pipeline Explanation	
Pipeline	**Description**
Raw Input Image	Raw input image expected to be coloured (e.g. Election Dataset Digits)
Apply Otsu Thresholding	Grayscale Image, and applies otsu thresholding with configuration in Table 5.6
Add Artificial White Border	Added artificial white border because on some images in the Election Dataset, some still have left-over borders after being cropped manually. We couldn't remove this during manual cropping because some images are tilted. We added artificial border on the sides of the image, to remove the left-over border by finding the largest contour which would be the white border.
Remove Artificial White Border	Remove white border
Resize Image 20x20 (proportional height)	Resize Image whilst preserving ratio as explained in [1]
Centre Image on 28x28	Centre the resized image to a blank picture 28x28 and then invert it

Table 5.6 Custom Pre-Processing Pipeline to Explain Figure 4.9

Otsu Thresholding Parameters for the Custom Pre-Processing Method		
Parameter	**Value**	**Explanation**
src	*img*	Input image (must be grayscale)
thresh	*0*	Threshold Value ignored in Otsu's method, but required syntactically
maxval	*255*	Assign value to pixels above the threshold (e.g. white in binary image)
type	*cv2.THRESH_BINARY + cv2.THRESH_OTSU*	Applies binary thresholding + Otsu's algorithm for automatic thresholding

Table 5.7 Otsu Thresholding Parameter For Custom Pre-processing Method

- **Resize Image :**

The following methodology serves as a baseline method for comparison between the 2 methodologies beforehand using the following pipeline: (1) inverting image to binary, (2) resizing image to 32x32 to fit the LeNet-5 input size.

Different Pre-processing Method Performance with Post-processing	
Method	**Accuracy**
Custom Pre-processing	0.4922
Resize Image	0.4860
SIREKAP Pre-processing	0.5986

Table 5.8 LeNet-5 Base Accuracy with SIREKAP with Different Pre-processing and Post-processing applied

Different Pre-processing Method Performance without Post-processing	
Method	**Accuracy**
Custom Pre-processing	0.5885
Resize Image	0.6172
SIREKAP Pre-processing	0.7189

Table 5.9 LeNet-5 Base Accuracy with SIREKAP with Different Pre-processing and Post-processing not applied.

Our evaluation process for Table 5.8 and Table 5.9 uses **Base Variation** model used in TENT's experiment mention in Section 5.3.1. This approach is taken instead of evaluating using the ensembled methodology due to: (1) The method's used for adapting using TENT and SHOT uses one model only, as such we do not replicate the ensembled approach, (2) TENT and SHOT performance on source domain yielded similar result on source domain (difference of 1%) in Section 6.2.1 which also similar to individual performance of the individual models found before ensembling in Table 5.3.

Table 5.8 and Table 5.9 showed us that the government post-processing heuristic method does not perform well on all pre-processing method. This evident by significant increase of accuracy on all pre-processing methods applied. However, the government pre-processing methodology performs the best out of all methods. We conclude that the government post-processing methodology is likely to be one of the causes of the decreasing performance when evaluating with our election dataset. Stemming from this conclusion, all subsequent experiments in TENT and SHOT would use the pre-processing method SIREKAP pre-processing.

5.3 Baselines

5.3.1 Configurations

LeNet models were selected for this study because of their straightforward design and minimal computational requirements, making them suitable for working with low-resolution images such as those in the MNIST dataset. While [11] does not specify the LeNet-5 architecture, it was chosen here as an efficient and lightweight model. On the other hand, [14] employs its own variant of LeNet, which we maintained to align with the original implementation. Despite the fundamental simplicity of the base models, the overall evaluation procedure became computationally expensive due to the testing evaluations process that is repeated 100 times.

5.3.1.1 TENT Configurations:

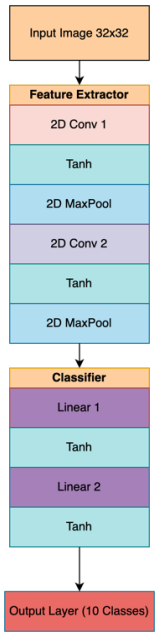


Figure 5.10 TENT LeNet-5 Base Architecture

This section defines the models that would be used in all TENT' experimentation established a base architecture as seen on Figure 5.10, our implementation for this follows the original LeNet-5 architecture in [1]. To explore the impact of Batch Normalization (BN) placement with TENT, we define variations of the base model by adding BN layers at different points in the network in Table 5.10.

Variations of LeNet-5 Model used In TENT Experiments					
Model Name	***BN after 2D Conv1***	***BN after 2D Conv2***	***BN after Linear1***	***BN after Linear2***	***Description***
Base Variation					Baseline model with no BatchNorm as seen in Figure 5.10
Variation #1					BN after every layer
Variation #2					BN only in feature extractor
Variation #3					BN after all convolutional layers and Linear1
Variation #4					BN after conv layers and Linear2
Variation #5					BN after Conv1 and Linear2 only
Variation #6					BN after Conv2 and Linear2 only

Variation #7					BN is applied after Linear2 has BN

Table 5.10 Variation of Batch Norm Location Defined (green coloured = BN applied, red coloured = No BN)

To provide clarity on the variations in Table 5.10, we provide the following explanations:

- **Model Architecture Split:**
  The placement of Batch Normalization (BN) layers in our experiments is defined based on their position relative to major transformation layers specifically, after convolutional layers or fully connected (linear) layers. As shown in our base architecture in Figure 5.10, BN layers can be optionally inserted in the **Feature Extractor Section**, following the layers labelled **"2D Conv1"** and **"2D Conv2"**, and in the **Classifier section**, following **"Linear 1"** and **"Linear 2"**. These four positions serve as the decision points for constructing the variations described.
- **Table Format and Column Meaning:**
  We define the "Base Variation" as the original LeNet-5 model without any Batch Normalization layers applied. In the visualization table, cells are color-coded, **green** indicates that a BN layer is applied after a specific layer, while **red** indicates that no BN layer is present in that position.

Our generation for these variations were random. Due to the possibility of the cells being only 2 colours (red or green) with 4 possible locations, we determine that the possible maximum of variations is 15 excluding the base configuration. This calculation was gained using binary combination using (8). Our variations covered half of all possible combinations that was enough to understand whether BN Layer placement effected prediction before and after TENT is applied.

$$\text{total combinations} = (possible\ choices\ in\ layers)^{layers} = 2^4 = 16 \quad (8)$$

For all the model variations explained above we then train each of them using the configuration settings in Table 5.11 using the pytorch library.

Training Settings Used For Models in Table 5.10 used in TENT	
Settings	**Description**
Optimizer	Stochastic Gradient Descent
Scheduler	ReduceLRonPlateu with learning rate 0.1
Epochs	25
Loss Function	Cross Entropy

Table 5.11 Training Settings Used for All LeNet-5 in Table 5.10 used in TENT.

All TENT models were trained on source domain (MNIST) for 100 times. After each training run, the model is evaluated on the source domain test dataset. If the resulting accuracy surpasses the current best, the model is retained. This process is repeated 100 times to select the model with the highest achieved accuracy.

5.3.1.2 SHOT Configurations:

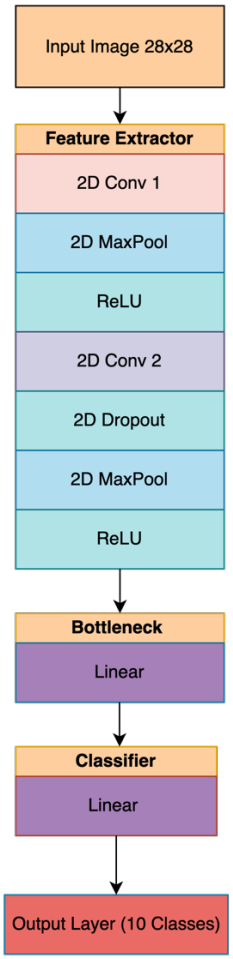


Figure 5.11 SHOT LeNet Base Architecture

Training Settings Used for LeNet in Figure 5.11 used in SHOT	
Settings	**Description**
Optimizer	Stochastic Gradient Descent (SGD) with a learning rate 0.01
Scheduler	Custom scheduler [32] with learning rate of 0.01
Epochs	30
Loss Function	Cross Entropy with Label smoothing
Fixed Random Seed	2020

Table 5.12 Training Settings Used for LeNet in Figure 5.11 used in SHOT

In SHOT, we use the same training approach as TENT explained in Section 5.3.1.1 where the best model is chosen after 100 training iteration, we opted by using the given LeNet architecture from the official implementation [32] . Additionally, we use the settings used in training the LeNet model as shown in Table 5.12. We made the decision to use a fixed random seed of 2020 due to the high computation that SHOT requires for our experiments in the post-adaptation detailed in Section 5.4. A fixed random seed of 2020 would ensure reproducibility and consistency when investigating the results gained from our experiment.

Note we acknowledge that there are differences introduced from the training configurations used by SHOT and TENT configurations that would result in unfairness. We mitigated this concern by using LeNet based architecture for both methods, and present pre-adaptation and post-adaptation results in Section 6.1 and Section 6.2. On pre-adaptation all models perform comparably on the MNIST test dataset with result in all evaluation metrics within the range of 99%. However, pre-adaptation results on election dataset showed a contrasting difference where SHOT's custom LeNet architecture showed drop in accuracy to 28% and TENT LeNet -5 architecture also showed a drop in accuracy to a range of 68% to 72%. Despite this, post-adaption for both methods SHOT and TENT showed comparable performance with results in the range 68-80%.

5.4 Performance Evaluation

We define performance evaluation into two phases pre-adaptation phase and adaptation phase. Pre-adaptation phase is defined as the phase before TENT and SHOT is applied to their respective models. In this phase we evaluated the performance using the evaluation metrics established in Section 5.5. We evaluated two different performances: (1) Source Domain Performance, (2) Target Domain Performance. Source Domain Performance evaluated performance of all models under TENT and SHOT respectively mentioned in Section 5.3.1 to show performance before adaptation on MNIST dataset. Target Domain Performance evaluated performance for all models under TENT and SHOT respectively on the Source Domain Election Dataset mentioned in Section 5.1.2.

In post-adaptation for both TENT and SHOT, adaptation was first performed on the test data, followed by evaluation with adaptation disabled. After each evaluation, the adapted model weights are reset to their pre-adaptation weights, and the adaptation method is repeated. This process is repeated for 100 times to ensure consistency. After getting 100 iterations we average and show the standard deviation using evaluation metrics found in Section 5.5 after adaptation.

Adaptation phase is defined as the phase after TENT and SHOT is applied to their respective model. In this phase we evaluated performance using the evaluation metrics in Section 5.5 where we also evaluated on: (1) Source Domain Performance, (2) Target Domain Performance. In this phase we evaluated different aspects from applying different settings in experiments for TENT and SHOT respectively.

TENT Optimizer and Loss Function for Adaptation	
Name	**Description**
Optimizer	Stochastic Gradient Descent
Loss Function	SoftMax Entropy Minimization

Table 5.13 TENT Optimizer and Loss Function Used for Adaptation

SHOT Optimizer and Loss Function for Adaptation	
Name	**Description**
Optimizer	Stochastic Gradient Descent
Loss Function	SHOT's custom loss function (5) that uses entropy minimization, information maximization, and centroid based pseudo labelling. Specific code can be found in [32].

Table 5.14 SHOT Optimizer and Loss Function Used for Adaptation

SHOT and TENT, works by retraining certain layers as explained thoroughly in Chapter 4. In brief, TENT works by targeting the BN layers, and optimizing their affine parameters with parameter modulation as explained in Section 3.1 whereas SHOT works by optimizing the feature extractor part of a model, whilst keeping the classifier layer frozen/not retrained as explained in Section 4.2. To retrain these layers a loss function, and an optimizer is required. To visualize this we give a representation in the form of a table in Table 5.13 and Table 5.14, following it's respective original implementation in TENT [33] and SHOT [32] where no values are changed.

TENT Different Settings/Conditions Evaluated			
Settings/Conditions	**Values Tested**	**Description**	**Intention**
Step Size	1,5,10.	Specifies the count of optimization iterations executed on one batch of data prior to moving on to the subsequent batch. This parameter regulates the number of times the model adjusts its weights with the same batch before moving to the next batch.	Evaluate effect of increasing step sizes
Batch Size	16,32,64,128,256,512.	Specifies target domain batch size, we tested values	Evaluate effect of increasing batch sizes
Adaptation Data Size Experiment	10-90%	All TENT LeNet-5 variants were evaluated using varying "train:test" splits of the source domain dataset. The training (adaptation) fraction was varied from 10% to 90%, with the remaining portion used for evaluation (e.g., we train 60% for adaptation with TENT and 40% for testing/evaluation). During evaluation, adaptation was disabled to assess performance using the adapted model. **Implementation can be seen in Appendix 1**	Evaluate the dataset required for TENT to learn the full dataset (e.g. How much data does TENT require to see improvements?)

Table 5.15 Different Settings/ Condition Evaluated for TENT Adaptation

SHOT Different Settings/Conditions Evaluated			
Settings/Conditions	**Values Tested**	**Description**	**Intention**
Adaptation Data Size Experiment	10-90%	The SHOT LeNet model was evaluated using varying "train:test"splits. Specifically, the adaptation dataset was created by sampling 10% to 90% of the source domain data, with the remaining portion used for testing (e.g., 60% of the source data for adaptation and 40% for testing/evaluation). During evaluation, adaptation was disabled to assess performance using the adapted model. **Implementation can be seen Appendix 2**	Evaluate the dataset required for SHOT to learn the full dataset. [14] mentioned that SHOT only requires small iteration/dataset size to differentiate using the centroid and information maximization explained in Section 4.2 (e.g. How much data does SHOT require to see improvements?)

Table 5.16 Different Settings/ Condition Evaluated for SHOT Adaptation

Table 5.15 and Table 5.16 presented different settings/conditions for TENT and SHOT evaluated after adaptation. Whilst TENT was tested across a broader range of settings (e.g. step size, batch size, adaptation data size) SHOT was only evaluated under the adaptation data size condition. This limitation primary comes from the computational intensity of SHOT, unlike TENT that only performs entropy minimization on BN Layers, SHOT involves training on larger optimization scope as explained in Section 4.2. As a result, SHOT's adaptation phase is significantly more resource-demanding when we must repeat our post-adaptation experiment for **100 times** to ensure robustness. Our experiments showed that evaluating for SHOT settings already took 1-2 days for each of the values tested.

5.5 Evaluation Metrics

Digit classification is a classic problem in the computer vision field, thus evaluation metrics for this study uses standard classification metrics used to quantify performance with an explanation given for the context.

- **Accuracy:**
  Calculates how many digit predictions are correct.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \quad (9)$$

- **Per Class Accuracy:**
  Despite accuracy already being counted, per class accuracy calculates how many digits predictions are correct given a class. This allows us to observe the domain shift that occurs to the imbalanced dataset presented.

$$\text{Accuracy}_{\text{class } c} = \frac{\text{Number of correct predictions for class } c}{\text{Total number of samples of class } c} \quad (10)$$

- **Precision:**
  Calculates how many predicted digits of a given class is correct. (e.g. Out of all the model prediction that confirms the result is 3, how many times is it correct?)

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}} (11)$$

- **Recall:**
  Calculates how many actual digits of the class were successfully predicted. (e.g. Out of all the real labels of the digit as 3 how many did the model find)

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}} (12)$$

- **F1-Score:**
  Harmonic mean of precision and recall balances both recall and precision.

$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} (13)$$

Chapter 6 Experiments

This chapter discusses the results based on the experiments and justifications detailed in Chapter 5. In this chapter baseline performance and domain gap is established, subsequently we present results of TENT and SHOT adaptation.

6.1 TENT

The following sections presents the results discussed for TENT tested under different variations settings as explained in Tanble 5.10 in Section 5.3, explaining the many variants of the LeNet-5 models tested for TENT.

6.1.1 Performance Evaluation (Pre-Adaptation)

We establish a baseline performance as a reference point to clearly evaluate the performance of TENT in subsequent experiments. In Table 6.1and Table 6.2 we evaluate the performance by inferring on both the source and target domain.

LeNet-5 Models Performance Before Adaptation with TENT on Source Domain MNIST				
Model	**Accuracy**	**Recall**	**Precision**	**F1**
Base Variation	0.9950	0.9884	0.9846	98.28
Variation #1	0.9909	0.9908	0.9909	99.09
Variation #2	0.9920	0.9919	0.9920	99.19
Variation #3	0.9886	0.9885	0.9885	98.85
Variation #4	0.9900	0.9899	0.9899	98.99
Variation #5	0.9900	0.9899	0.9898	98.99
Variation #6	0.9908	0.9907	0.9908	99.07
Variation #7	0.9908	0.9907	0.9908	99.07

Table 6.1 Performance of LeNet-5 BatchNorm Variants on the MNIST Source Domain Before Adaptation. *All models are defined in Section 4.3.1*

LeNet-5 Models Performance Before Using Adaptation with TENT on Target Domain Election Dataset				
Model Name	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Base Variation	0.7189	0.7042	0.6901	0.6760
Variation #1	**0.7477**	**0.7356**	**0.7182**	**0.7049**
Variation #2	0.6894	0.6876	0.6599	0.6458
Variation #3	0.6475	0.6699	0.6205	0.6105
Variation #4	0.7081	0.6956	0.6960	0.6772
Variation #5	0.7259	0.7199	0.6976	0.6801
Variation #6	0.6801	0.7171	0.6566	0.6411
Variation #7	0.6801	0.7171	0.6566	0.6411

Table 6.2 Performance of LeNet-5 BatchNorm Variants on the Election Target Domain Before Adaptation. *Highest values are bolded. All models are defined in Section 5.3.1*

All LeNet-5 models variants achieved performance of 99% in all metrics as described by Table 6.1 during the pre-adaptation phase when evaluating against the source domain MNIST dataset. This result concludes that the models are successfully trained on MNIST, and no overfitting has occurred upon training and testing on the source domain dataset shown by the consistent evaluation metrics. Table 6.2 shows a performance drop when evaluating on target domain

election dataset with **Variation #1** achieving the highest accuracy of 74.77% that is higher the baseline performance **Base Variation** of 71.89% where there are no Batch Norm Layers. The variation in performance across models aligns with the architectural differences described in Table 5.10, suggested difference BN Layers placement in the architecture does affect a model's performance.

6.1.2 Adaptation Performance Evaluation

6.1.2.1 Hyperparameter Sensitivity

Our findings in this section are intended to provide an understanding of whether changing step or batch size proved to increase and help with TENT performance on different Variations of Batch Norm Placement.

LeNet5 Models Performance after being applied TENT				
Model Name	**Average Accuracy**	**Average Precision**	**Average Recall**	**Average F1 Score**
Variation #1	0.7186 ± 0.0018	0.7327 ± 0.0035	0.6885 ± 0.0019	0.6704 ± 0.0021
Variation #2	0.6872 ± 0.0015	0.6901 ± 0.0021	0.6540 ± 0.0015	0.6411 ± 0.0016
Variation #3	0.6331 ± 0.0017	0.6644 ± 0.0016	0.5766 ± 0.0015	0.5683 ± 0.0016
Variation #4	**0.7427 ± 0.0016**	**0.7193 ± 0.0024**	**0.7094 ± 0.0015**	**0.6968 ± 0.0015**
Variation #5	0.7292 ± 0.0011	0.7252 ± 0.0013	0.6881 ± 0.0013	0.6752 ± 0.0013
Variation #6	0.6620 ± 0.0018	0.7028 ± 0.0020	0.6332 ± 0.0021	0.6202 ± 0.0019
Variation #7	0.6622 ± 0.0017	0.7028 ± 0.0021	0.6334 ± 0.0020	0.6205 .0018

Table 6.3 Performance of LeNet-5 Variants After TENT Adaptation on Target Domain (All models are evaluated on the election dataset. Metrics are reported as mean ± standard deviation, based on 100 runs*)*.

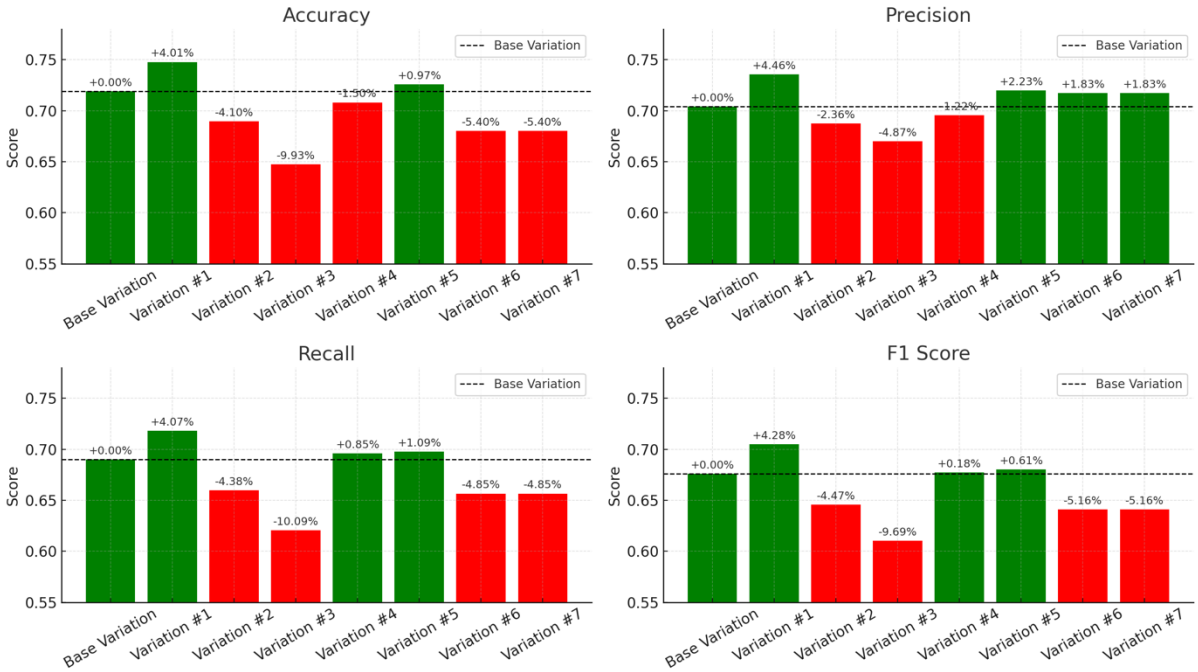


Figure 6.1 Metric Difference Comparison of Base Variation found in Table 6.2 and Variations 1-7 Based on Table 5.10 (To Compare Performance on Source Domain Before and After TENT adaptation).

Note All experiments shown in Table 6.2 and Figure 6.3 used the same settings with step size = 1, and batch size = 512.

After applying TENT to all BN Layers Variations shown in Table 6.3, we observed an unexpected drop of performance when compared with Base **Variation** except for **Variation #1** and **Variation #4**. In the Table 6.3 we observed that TENT introduced small standard deviations. This variability is attributed to TENT' retraining the BN Layers and receiving the image in a different order for each iteration across 100 iterations that we evaluated. This variability further strengthens our finding in Section 6.1.1 where placement of BN Layers in an architecture does play a role in TENT performance. Study by Hasani and Kotanlu [34] confirms this finding, as they observed that the placement of BN Layers affected the iterations it took for models ALEXNET,VGG-16, and RESNET-20 to achieve a desired accuracy.

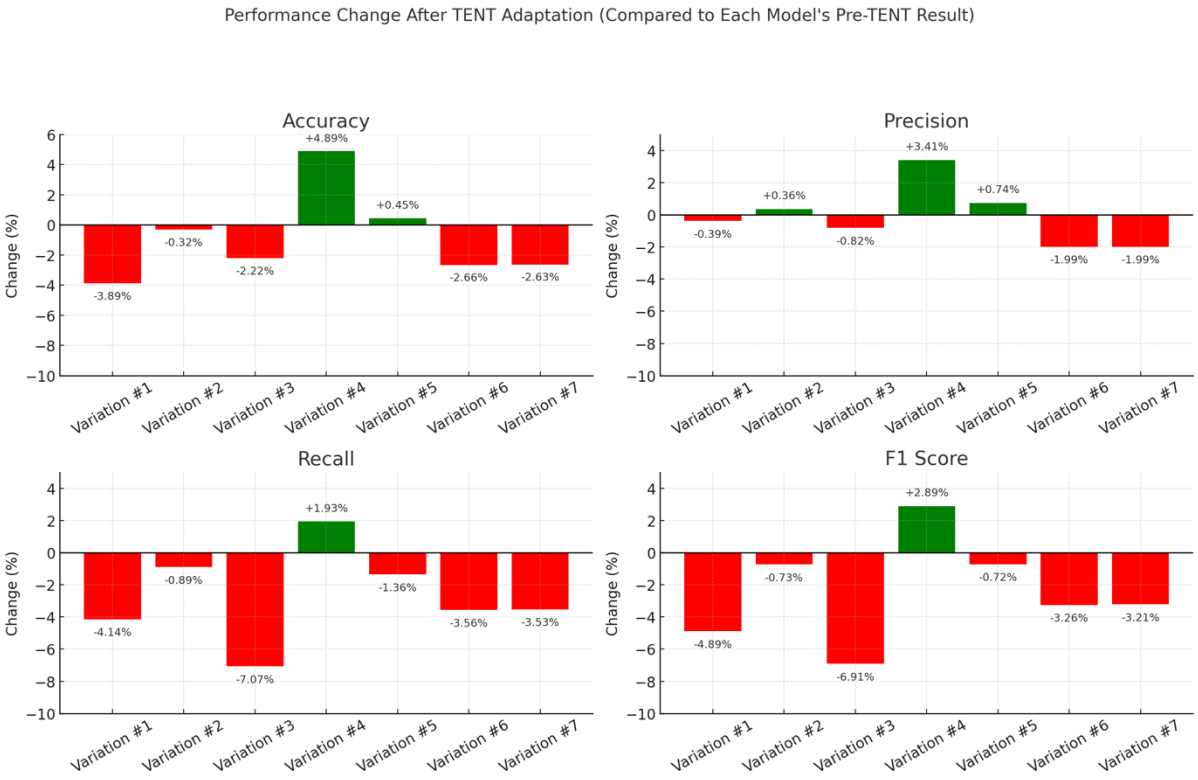


Figure 6.2 Metric Difference Comparison of Variation 1-7 found in Table 6.2 in comparison to Variations 1-7 found in Table 5.10 (To Compare Performance on Source Domain Before and After TENT adaptation) .

However, Figure 6.2 above shows, only **Variation 4#** as the increasing variant in overall performance when comparing with its performance on source domain before adaptation. This findings is also supported by [34] findings where arrangements of BN Layers positions tested yields different results. Despite this, [34] suggest more iterations on the dataset are needed when BN Layers are placed after Convolutional Layers. To investigate further we investigated by varying the hyperparameter step size on all variations.

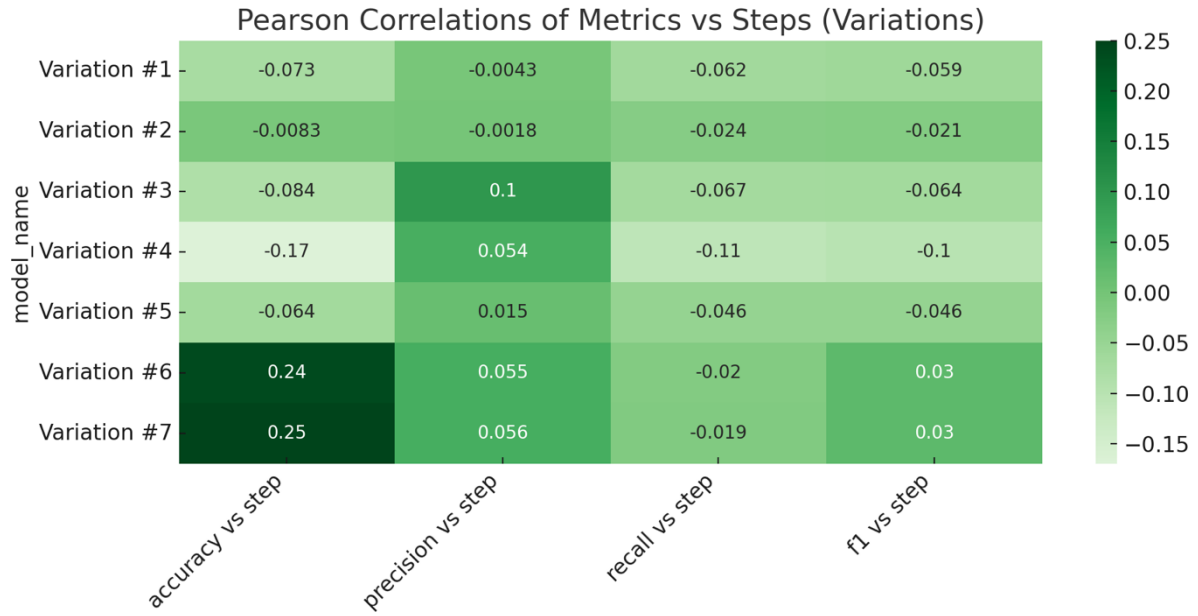


Figure 6.3 Pearson Correlation of Evaluation Metrics in Comparison to Step Size for TENT

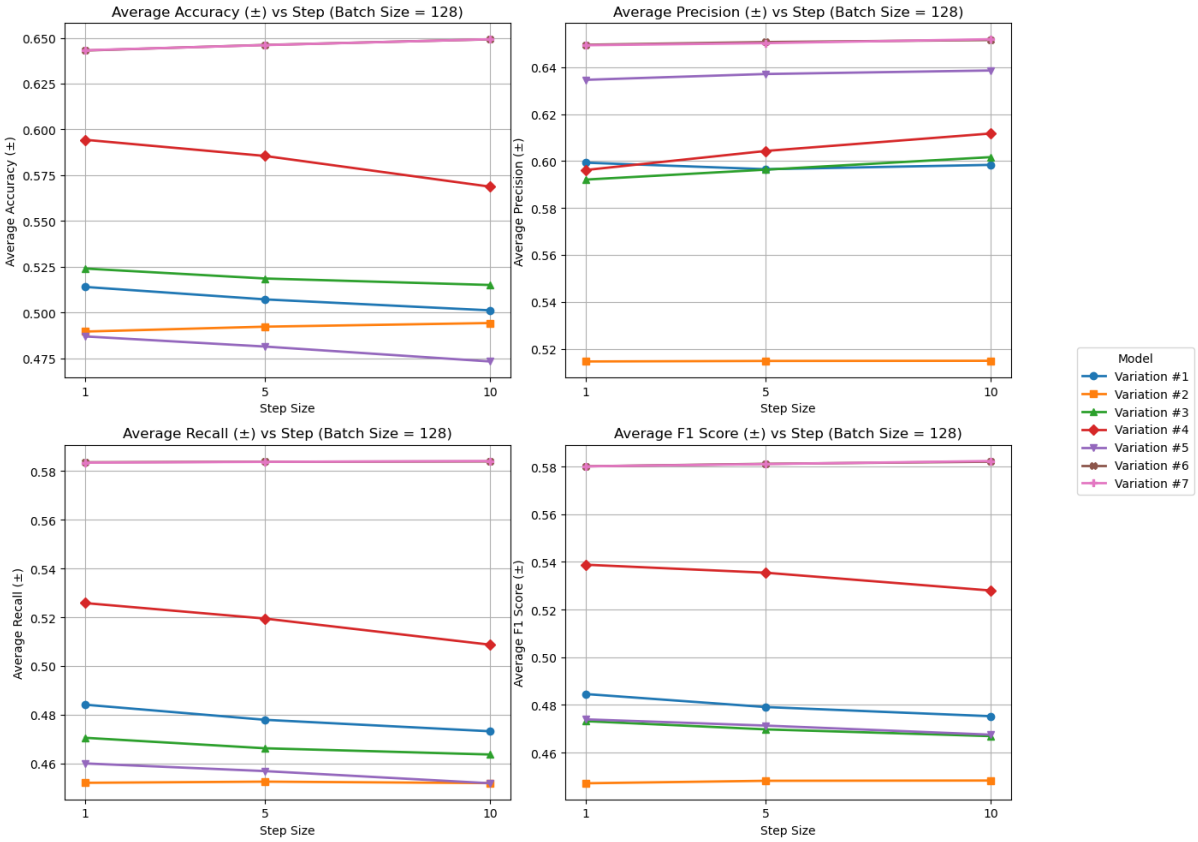


Figure 6.4 Trends of Step Size in Comparison with Evaluation metrics

We investigated further by comparing our Pearson correlation Figure 6.3 to the trends shown by an increasing step size. Figure 6.4 suggested that the increase of step (iterations) in TENT only showed positive correlation in **Variation #6** and **Variation #7** (both overlaps each other indicated by the colour pink and brown). **Variation #6** and **Variation #7** shares a common architecture where there are no BN layers after 2D Conv1 and after Linear1. This architecture choice for these variations in the experiment likely attributed to an increase in performance

trend as step size increases for both variations, whilst other variations showed a downward trend for accuracy as shown in Figure 6.4. Despite this the increase of iterations in our case, does not complement the findings in [34] where more iterations / step increases overall accuracy. Despite this, the Pearson correlation Figure 6.5 and the trends shown in Figure 6.6 showed a positive correlation with significant improvement when batch size are increased.

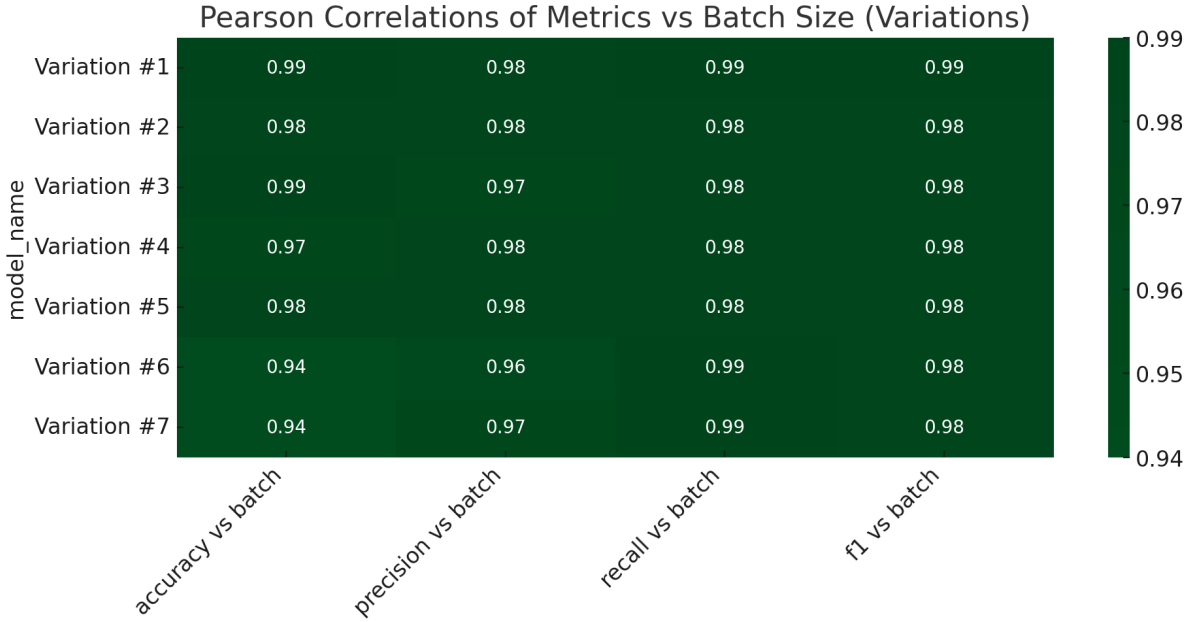


Figure 6.5 Pearson Correlation of Evaluation Metrics in Comparison to Batch Size for TENT

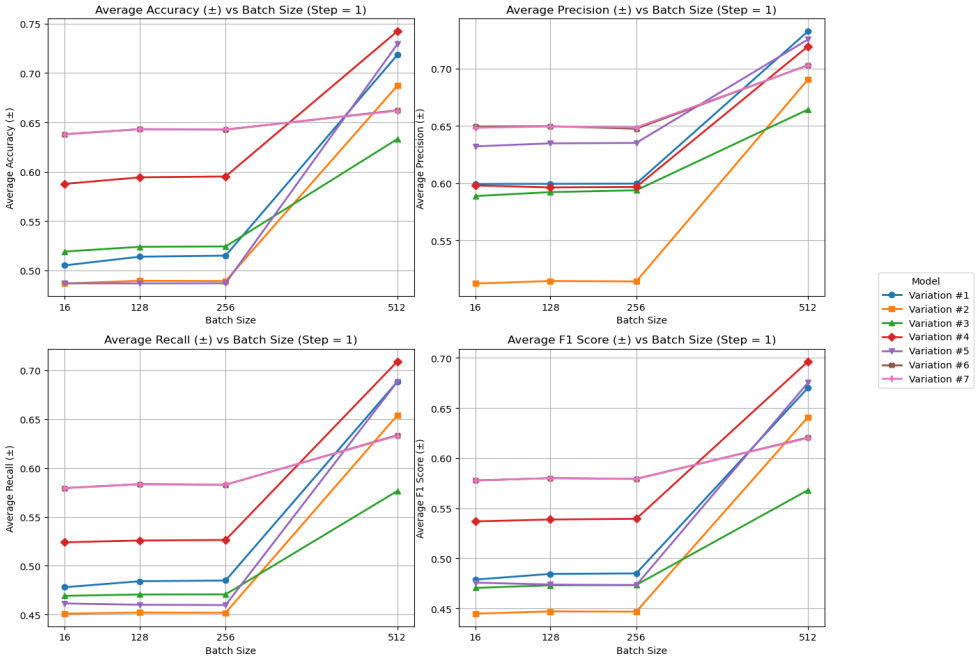


Figure 6.6 Trends of Batch Size in Comparison with Evaluation metrics

This increase in performance for all metrics, as batch size gets bigger from 256 to 512, suggests that TENT performs better with larger batch sizes. Figure 6.6 suggests that all variation had exponential growth in batch size 256 which indicated that it might surpass the baseline performance shown in Figure 6.1 and surpass its pre-adapted models shown in Figure 6.2. However, increasing the step size yields different results and unpredictable behaviour in contrast to the findings in [34]. These volatility is supported further in the paper [35] where

they evaluated batch norm in TTA methods as a crucial obstacle that suffers from many failure cases they tested in comparison to the more stable layer norm and group norm.

6.1.2.2 Dataset Size Variation and Overfitting Evaluation

In the following section we investigate the effects on varying amount of training dataset that TENT receives whilst performing adaptation as seen in Figure 6.7. As training dataset increases for adaptation, we expected TENT to allow the model to learn better improving all metrics when evaluated against the target domain election dataset, however this approach also comes with overfitting concerns.

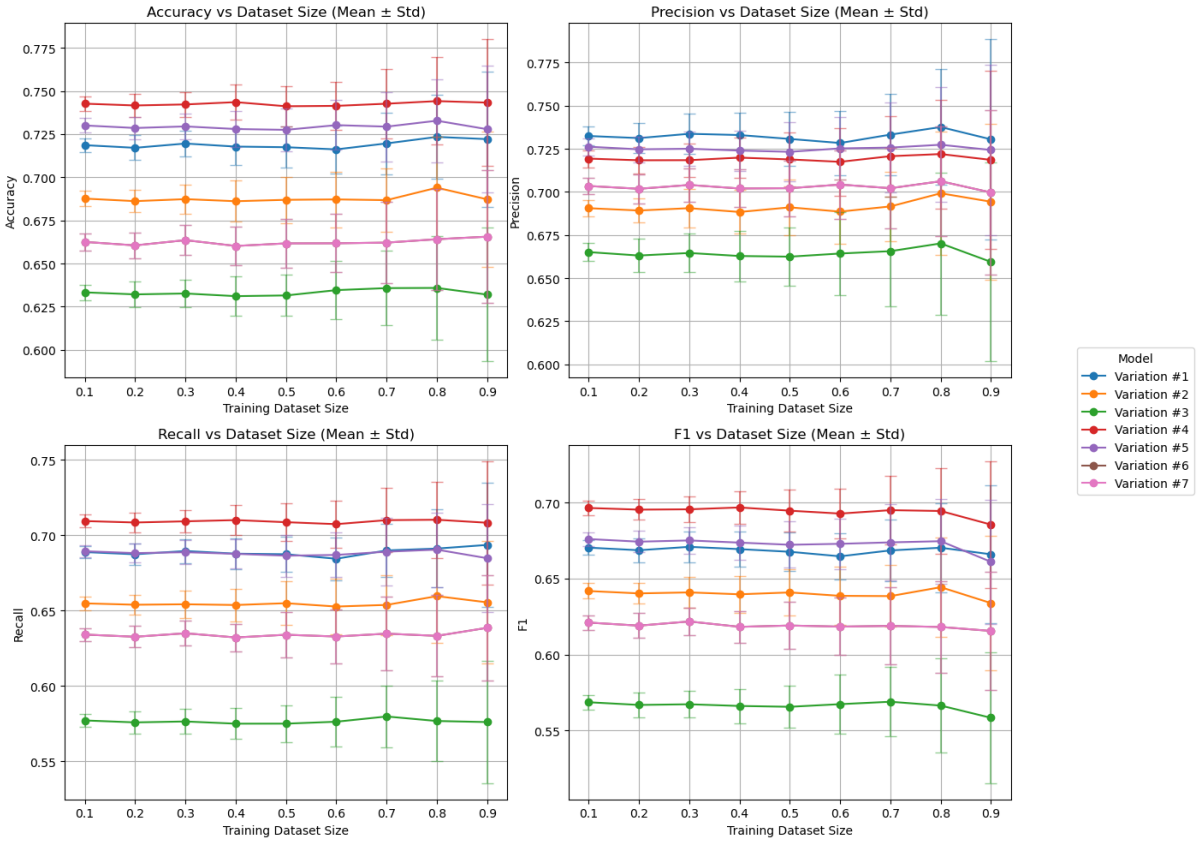


Figure 6.7 Trends of Increasing Adaptation Dataset Size (See Appendix 1 for the code implementation to turning on and off TENT)

Our findings from Figure 6.7 suggest that adding more source domain for adaptation with TENT, does not increase its overall performance in fact it shows a stable trend across all metrics. This suggest that TENT may not need large dataset to adapt to the target domain.
To confirm our previous assumption that TENT may cause overfitting, we conducted evaluation using on all TENT variations.

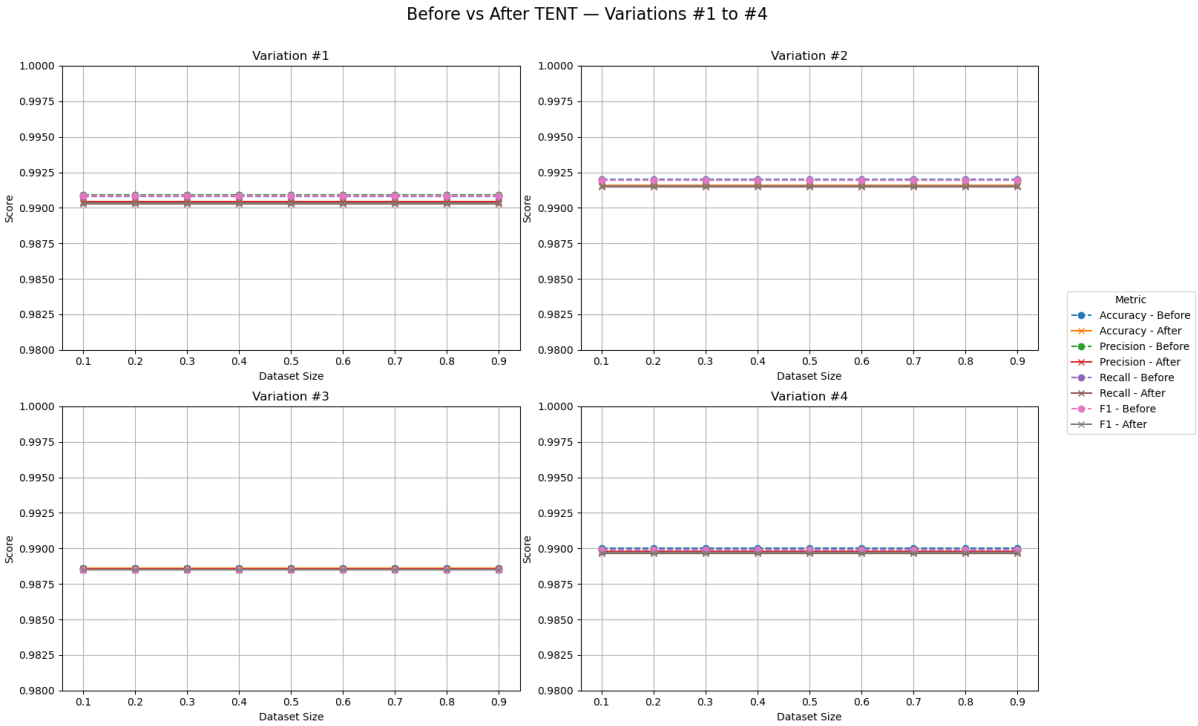


Figure 6.8 Trends of all Evaluation Metrics for Model Variations (1-4) Before and After TENT

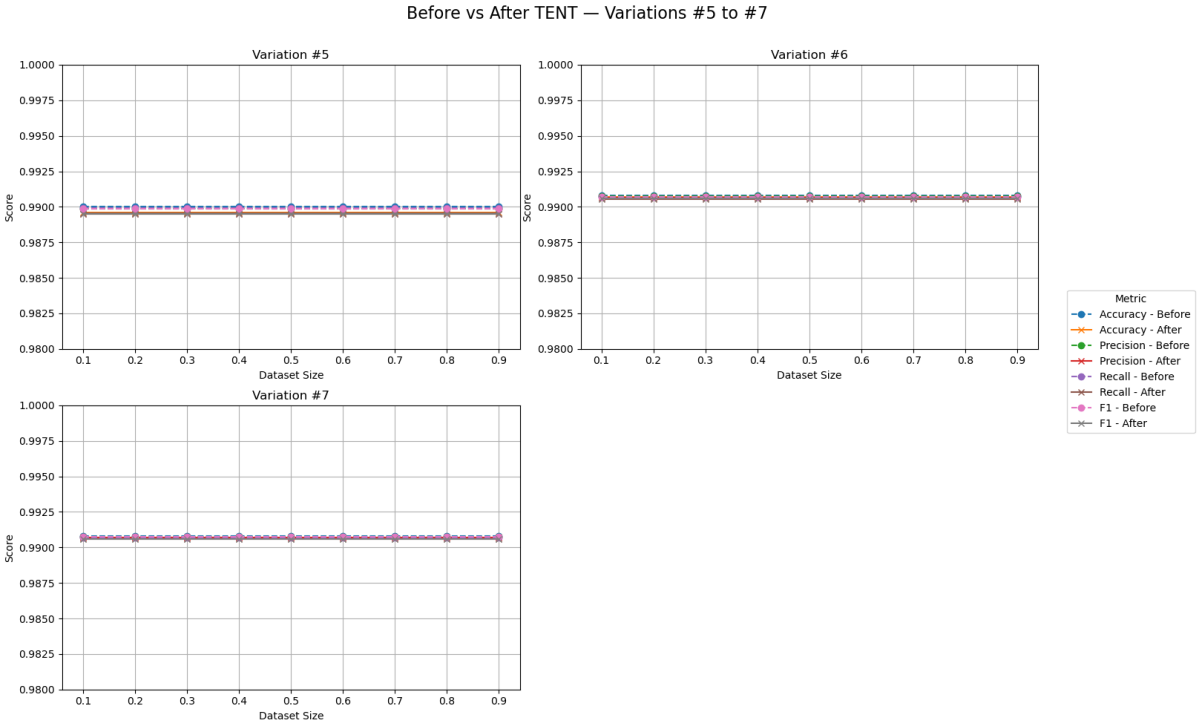


Figure 6.9 Trends of all Evaluation Metrics for Model Variations (Model 5-7) Before and After TENT

Our findings indicate that TENT does not cause overfitting, as Figure 6.8 and Figure 6.9 showed there was no significant drop for all metrics, still staying within the 99% range, in contrast to our previous assumption. This is a positive aspect, suggesting TENT can adjust for domain shift without overwriting features learned on source domain.

6.2 SHOT

6.2.1 Performance Evaluation (Before Adaptation)

We established a baseline performance as a reference point to evaluate the performance of SHOT in subsequent experiments.

- **Source Domain and Target Domain Performance:**

LeNet Performance Before Applying SHOT				
	Accuracy	**Precision**	**Recall**	**F1**
LeNet->MNIST	99.19	0.99	0.99	0.99
LeNet->Election Dataset	26.16	0.39	0.26	0.13

Table 6.4 SHOT LeNet Model Performance Before SHOT Adaptation on MNIST and Election Dataset

As with TENT, the custom LeNet model [14] exhibited poor performance where evaluation metrics drop to the range of 13% to 39 %, range, due to the effect of domain shift, indicated by a significant performance drop on all evaluation metrics. This is likely to be attributed to the difference of base architecture that TENT and SHOT used, explained in Section 5.2

6.2.2 SHOT Adaptation Results

6.2.2.1 Adaptation Data Size

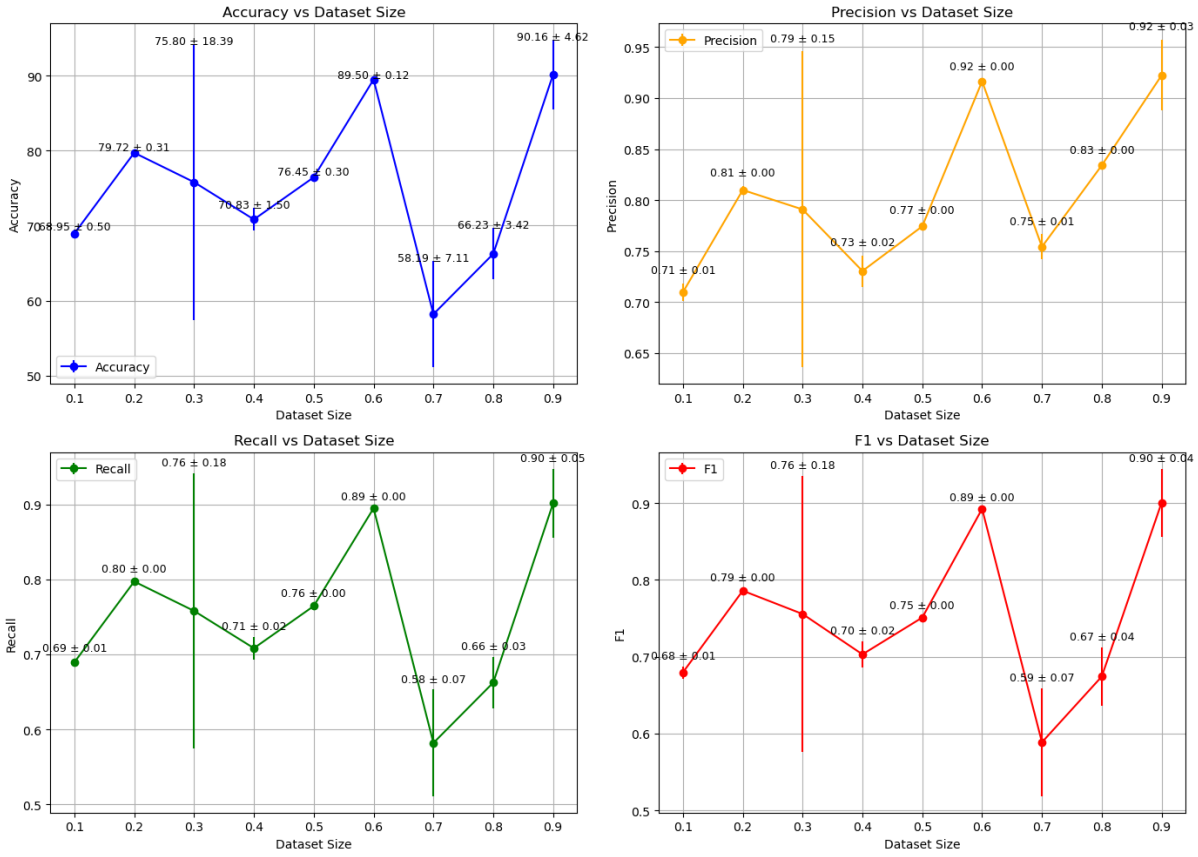


Figure 6.10 Evaluation Metrics for SHOT with differing Dataset Size

Unlike TENT our adaptation for SHOT showed that SHOT adaptation significantly influenced performance with the with the same fluctuating trend across all evaluation metrics. Unlike typical learning curves, performance did not consistently increase with more adaptation data. All metrics showed signs of repeated “rise and drop” periods. This period happens when training dataset size for SHOT is 0.2-0.5 and 0.5-0.9. This is unexpected, as we expected the

curve to increase despite overfitting concerns. Instability was shown when dataset size for SHOT is 0.3, this suggested that SHOT is sensitive on specific data subset used during adaptation, where it's likely exacerbated by the dataset's imbalanced, and the fixed random seed (202) used. Figure 6.10 suggest that using more dataset size for training would not guarantee better results on our source domain dataset. We hypothesized that a fixed random seed (2020) to be the likely cause of the fluctuating trend.

6.2.2.2 Investigating Fluctuating Trends

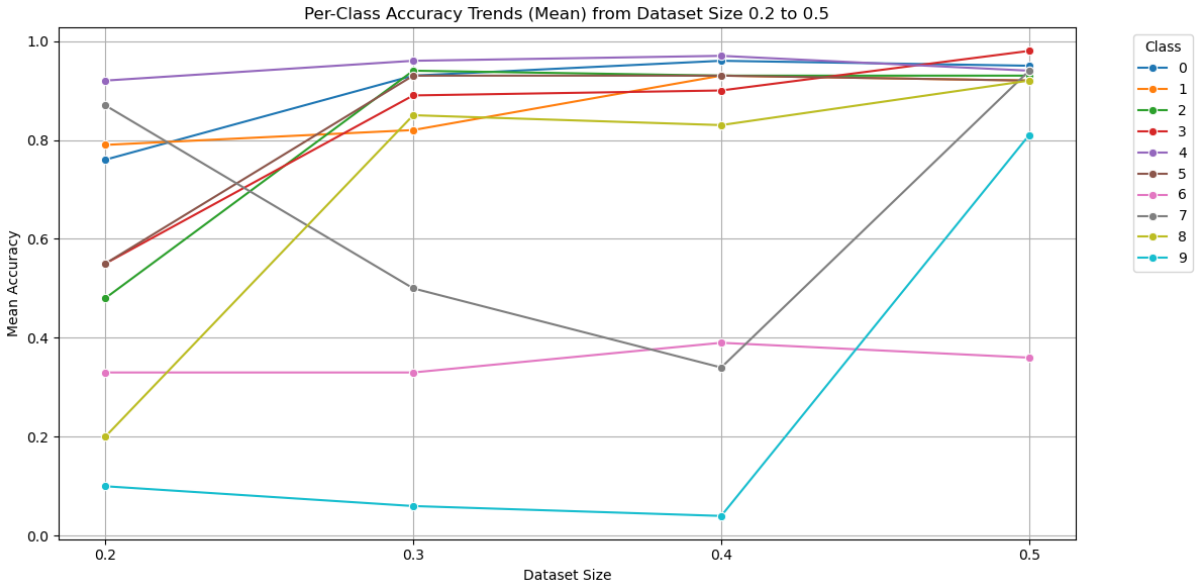


Figure 6.11 Per Class Accuracy on MNIST Dataset Size 0.2-0.5

We investigated the fluctuating trends further by examining SHOT per class accuracy. We observed 2 fluctuating trends in Figure 6.11 and Figure 6.12. In the graph Figure 6.11. We observed that per class accuracy of class 7 to be the only class dropping from ~90% accuracy to the range of ~30% accuracy. Class 7 is the only class shown to have a steep significant performance drop; thus, we attribute the fluctuating performance drop to it.

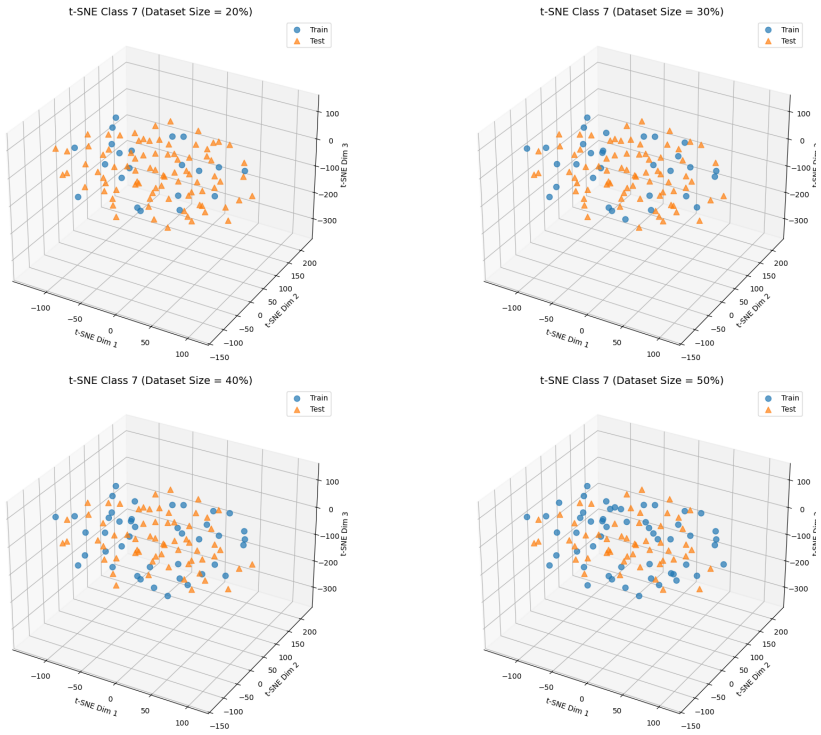


Figure 6.12 3D T-SNE Visualization for Train and Test Split in Fixed Seed 2020 from Dataset Size 0.2-0.5 for Class 7

To confirm our hypothesis that a fixed random seed (2020) is the cause of the fluctuation. We observed whether the training dataset is representative of the target test distribution. We visualised the feature space of Class 7 using 3D t-SNE across four different dataset size splits where the fluctuations happened (20%, 30%, 40%, 50%). (Refer to Table 5.16 for definition of dataset size split)

As shown in Figure 6.12, the 3D T-SNE we observed that there are increasing overlaps between the training and target dataset which indicated the model was supposed to be able to differentiate the classes.

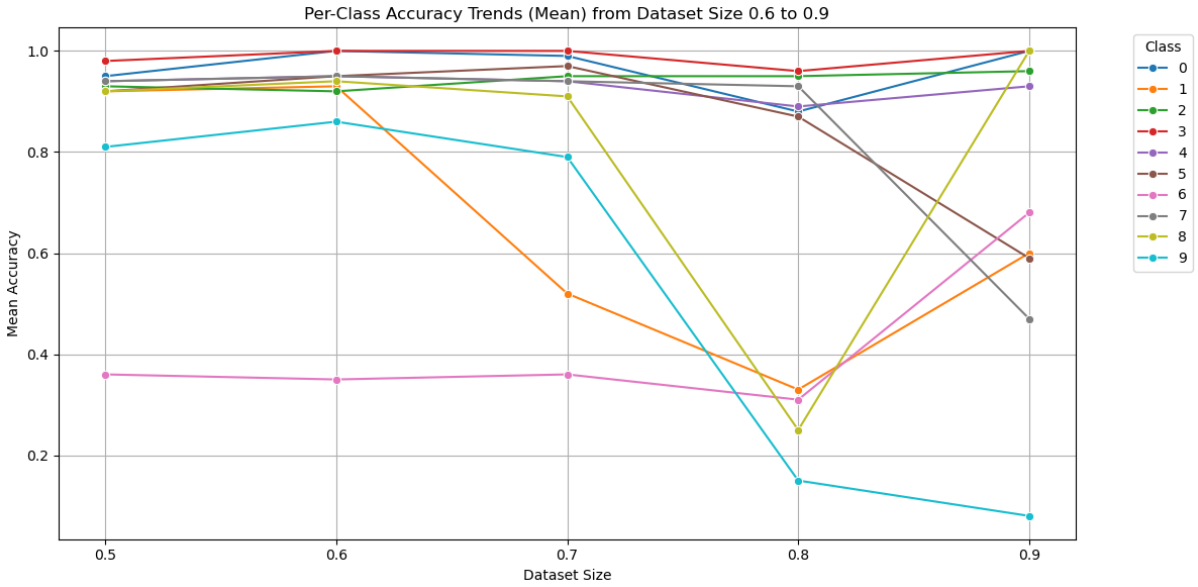


Figure 6.13 Per Class Accuracy on MNIST Dataset Size 0.5-0.9

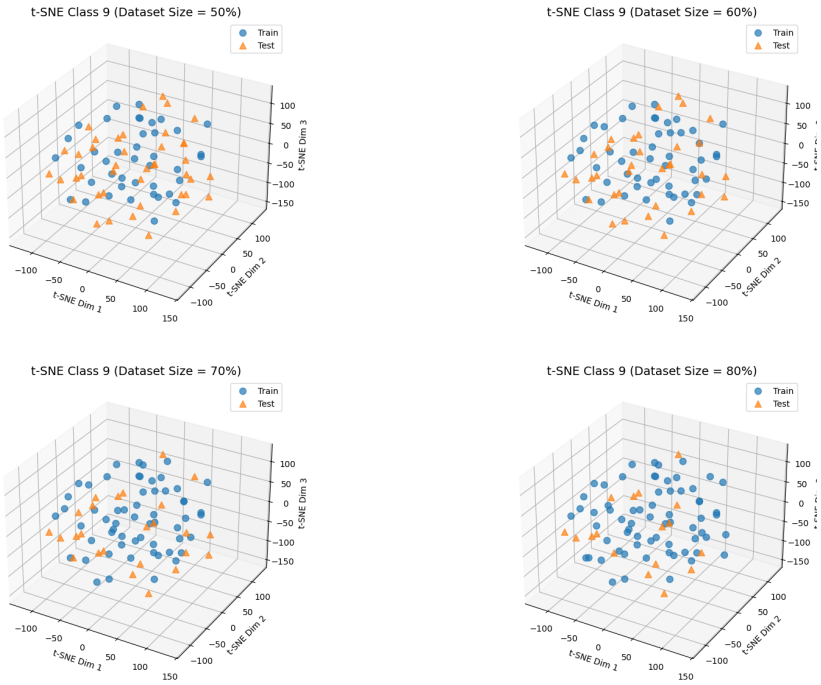


Figure 6.14 3D T-SNE Visualization for Train and Test Split in Fixed Seed 2020 from Dataset Size 0.2-0.8 for Class 7

The same effect is also shown when investigating the per class analysis result for the fluctuation in 50%-90% dataset size, where it revealed a downward trend with the most prominent on class 9 on Figure 6.13. Thus, we investigated further with a 3D T-SNE in Figure 6.14 showing its train and test split. The overlapping split shown should enable SHOT to differentiate the class 9. Our investigation with the 3D T-SNE is first initiated by the assumption that the random fixed seed (2020) is the likely cause the variations and fluctuating performance generating test and training set that doesn't converge. However as shown, this proves to not be true when we

observed class with significant performance drop on both dataset size instance. This highly suggested SHOT's pseudo-label loss (3) and information maximization loss (4) refinement mechanism maybe unstable or sensitive to class imbalance we presented in Section 4 or possibly requiring a hyperparameter adjustment for β in (5).

Performance Evaluation on SHOT After Adaptation		
Dataset Size	**Average MNIST Accuracy (±)**	**Average Election Accuracy (±)**
0.1	97.281 ± 0.26	68.95 ± 0.5
0.2	94.914 ± 0.6	79.72 ± 0.31
0.3	94.68 ± 0.21	75.8 ± 18.39
0.4	93.588 ± 0.45	70.83 ± 1.5
0.5	90.941 ± 2.41	76.45 ± 0.3
0.6	89.638 ± 0.46	89.5 ± 0.12
0.7	86.44 ± 0.44	58.19 ± 7.11
0.8	80.583 ± 1.52	66.23 ± 3.42
0.9	85.798 ± 0.87	90.16 ± 4.62

Table 6.5 Performance of LeNet-5 Variants After SHOT Adaptation on MNIST and Election

Table 6.5 showed SHOT performance on MNIST on a downward trend in comparison to the Fluctuating trend that is presented on Election Dataset. A downward trend on the MNIST accuracy after adaptation suggested that SHOT causes overfitting as dataset size increases. This decrease in performance is expected as SHOT adapts/retrains more layers in comparison to TENT. However, we acknowledge that SHOT is able to increase performance significantly in evaluating target domain using a small dataset size as claimed in [14]

Chapter 7 Conclusion and Future Works

This project aims to investigate the effectiveness of 2 novel *test time adaptation* methodology, Test Time Adaptation by Entropy Minimization (TENT) and Source Hypothesis Transfer for Unsupervised Domain Adaptation (SHOT) on MNIST pretrained model (LeNet), to real world target domain of handwritten digits extracted from Indonesian 2024 Presidential Election tally form. The study focuses on addressing the significant domain shift observed between the clean benchmark dataset and imbalanced election data, under constraints that access to source training data during adaptation is not feasible.

Our study first identifies the problem as a generalization problem, where models struggle to predict unseen data outside of its training dataset. While retraining the model with the additional data is the preferred, this approach is deemed impractical for the specific context of the Indonesian 2024 Presidential Election tallying process which requires immediate processing on mobile devices over a short duration without immediate verified labels to be available. Thus, our study opted for *test time adaptation* a concept of adapting (retraining) selected layer without the need of labelled domain data.

Subsequently, we established our experimental environment. The MNIST dataset served as the source domain and a varying LeNet models in Section customized for their respective methodology were trained on it. The target domain is constructed by manually collecting and pre-processing 1288 digits from 200 official tally form (Form C1) sourced from [15] as explained in Section 5.1.2. Our analysis found that the data collected confirms a significant class imbalance. Experiments from reverse engineering the SIREKAP application following [17] gives us context on the methodologies, pre-processing and post-processing pipeline which is applied to our target dataset to closely resemble real-life scenario. Baseline performance metrics are then established on both source and target domains to quantify the initial domain shift.

We then implemented TENT, on LeNet-5 models with varying Batch Norm Layers placement. Our study systematically observes adaptation on varying Batch Norm Layers placement, adaptations step size, batch size, and inference on varying fraction of the target dataset as explained in Section 5.4. SHOT adaptation on customized LeNet model [14] is also applied with the same process methodologies as explained in Section 5.4.

Our study evaluates the implementation, using a comprehensive and systemic evaluation protocol in Section 5.6. The performance of the adapted models was compared against the non-adapted baselines against the target and source domain with evaluation metrics.

7.1 Findings

7.1.1 Overall Findings

- SIREKAP application pre-processing method is proven to be the most effective yielding good performance in comparison with other pre-processing method experimented, however the opposite is true for the application post-processing method. Despite this, the government ensembled methodology is shown to yield lowest result in comparison to individual LeNet models used in TENT and SHOT experiments.

- **Domain Shift Confirmation**
  A performance gap was found between the MNIST source domain and target domain Election Dataset. All models tested in TENT, SHOT, and government models found experienced significant performance ranging from 20-70% drop in chapter, motivating the need for adaptation. The target domain set was also found to be imbalanced favouring the class label "1" as the highest occurring label.
- **Dataset Size Variation for Adaptation**
  TENT indicates unnoticeable performance as dataset size grows bigger for target source adaptation as seen in Figure 6.7, whereas SHOT introduced fluctuating performance where we investigated it to be caused by dropping performance on average accuracy in class "7" and class "1" seen in Figure 6.11 and Figure 6.13.

7.1.2 TENT-Specific Findings

- TENT methodology proves in preserving source domain (MNIST) (not overfitting) with stable performance across different batch size, step size, and target dataset size.
- TENT's performance was highly sensitive to placement of BN architecture and required large batch size.
- Step size changes in TENT is shown to be complex across different LeNet-5 variations tested.
- Increasing batch size shows positive correlation with performance metrics, indicating a need for larges batches of images (e.g.,512) for stable and effective TENT adaptation.

7.1.3 SHOT-Specific Findings

- SHOT shows significant improvement of target domain performance over the non - adapted baseline. However, it's performance shows instability depending on the fraction of target domain set received, with fluctuating trends and high variance. This performance was unexpected, subsequent experiments were experimented, where the assumption that the fixed seed (2020) cause diverging test and train cases for specific classes was revealed to not be true. Thus, we made the conclusion that SHOT fluctuating performance is highly likely attributed to our imbalanced dataset,
- SHOT adaptation resulted in noticeable overfitting as target data size it receives increases. This result is expected as SHOT tunes more layers in comparison to TENT.
- We found that SHOT [14] claim of using a small iteration for good separation of the data is valid. In this case small iteration is equals to the small dataset size.

The findings revealed, TENT achieving best average accuracy in **Variation #4** achieving 74% average whilst SHOT's claim that a favourable performance is able to be achieved with lower dataset proved to be true [14] as dataset size 0.2 is able to increase it to 79% in target domain dataset, however SHOT suffers more in overfitting as dataset size of the target domain gets bigger whilst TENT shows stable performance in all metrics still within the range of 98%-99%. Each method is shown to improve performance with increasing evaluations metrics by over 50% in comparison to the SIREKAP ensembled models performances on the target domain data explained in Section 5.2.2

However, neither TENT nor SHOT, is found to suit the requirements under the Indonesian 2024 Presidential Election settings where the model updates it's performance locally on a mobile device immediately after a photo of the C1 Form is taken. This because both methods proved to require a large and balanced dataset possibly above the current count of 1288 images

to work with favourable and predictable performance that increases initial performance before adaptation.

The study also found that using LeNet-5 models' variants in TENT experiments, without the usage of ensembling method or test time adaptation, proves to yield higher result upon evaluating with target dataset [Refer to Table with Inference on Target dataset] suggesting that a simple change to LeNet-5 architecture might yield better performance for the SIREKAP application.

7.2 Limitation

Bigger Dataset and Balanced Dataset:
The relatively small and imbalance nature of the custom Election Dataset is shown to negatively impact SHOT adaptation performance and cause instability with large variants. The usage of sampling techniques to make a balanced dataset (e.g. stratification) might prove to work in bigger dataset. Future work should explore work with a bigger and balanced dataset, potentially employing sampling techniques to fully utilize the potential of TENT and SHOT.

Usage with centralized or decentralized (federated) learning:
The study also revealed that both methods might work under the setting of a centralized model where source domain adaptation happens server side thus allowing the model to receive larger datasets that is crucial for TENT and SHOT adaptation found in the study. Federated learning [36] might also potentially work as a decentralized approach, where trained models via SHOT or TENT lives on a server and adapted weights are then updated to models on end devices(e.g. smartphone, tablets, and personal computers). According to McMahan et.al [36] this approach would reduce privacy and security risk by limiting *attack surface* to one device. Future work should study this approach as it might be more perfectly suited for this.

Other Architectures and Hyperparameters:
The study is only able to reveal the effects of each methods adaptation within certain scope of hyperparameters and varying dataset sizes of target domain. LeNet based models are used due to its simplicity and low computation cost with time constraints consideration in constructing the experiments for this report. As such, future works should incorporate advanced models (e.g. ResNet [37] , GoogLeNet [38], VGG [39]), and explore more on the hyperparameters that can effect this process either on a balanced or imbalanced dataset as [13] highlights the volatility of test-time adaptations under different settings reviewed in Section 3.3

Other Methods of Test Time Adaptation:
Future work should incorporate more methods of Test Time Adaptation, research [13] shows promising results in other test time adaptation method (e.g. MEMO, NOTE, T3A, Co-TTA) respectively.
Combining TENT and SHOT:
The study is only able to observe how TENT and SHOT performs individually. Future work should incorporate work on both algorithm observing how doing SHOT in an offline matter first and subsequently combining TENT in an online manner after would improve stability over imbalanced dataset and avoiding overfitting.

References

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, 'Gradient-based learning applied to document recognition', *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.

[2] T. Kumari, Y. Vardan, P. Giridhar Shambharkar, and Y. Gandhi, 'Comparative Study on Handwritten Digit Recognition Classifier Using CNN and Machine Learning Algorithms', in *2022 6th International Conference on Computing Methodologies and Communication (ICCMC)*, Mar. 2022, pp. 882–888. doi: 10.1109/ICCMC53470.2022.9753756.

[3] 'Papers with Code - USPS Benchmark (Image Clustering)'. Accessed: Apr. 10, 2025. [Online]. Available: https://paperswithcode.com/sota/image-clustering-on-usps

[4] 'Papers with Code - MNIST Benchmark (Handwritten Digit Recognition)'. Accessed: Apr. 10, 2025. [Online]. Available: https://paperswithcode.com/sota/handwritten-digit-recognition-on-mnist

[5] 'Technical Issues Cause Discrepancies in Vote Counting Machine Data', Jakarta Globe. Accessed: Apr. 10, 2025. [Online]. Available: https://jakartaglobe.id/news/technical-issues-cause-discrepancies-in-vote-counting-machine-data

[6] Salma, 'Indonesia's 2024 Elections: Balancing Technology and Trust in Digital Age', Universitas Gadjah Mada. Accessed: Apr. 10, 2025. [Online]. Available: https://ugm.ac.id/en/news/indonesias-2024-elections-balancing-technology-and-trust-in-digital-age/

[7] 'IT Expert Reveals Three Sources of Sirekap Issues', en.mkri.id. Accessed: Apr. 10, 2025. [Online]. Available: https://en.mkri.id/news/details/2024-04-03/IT_Expert_Reveals_Three_Sources_of_Sirekap_Issues

[8] P. W. Koh *et al.*, 'WILDS: A Benchmark of in-the-Wild Distribution Shifts', in *Proceedings of the 38th International Conference on Machine Learning*, PMLR, Jul. 2021, pp. 5637–5664. Accessed: Apr. 10, 2025. [Online]. Available: https://proceedings.mlr.press/v139/koh21a.html

[9] M. Ramazanova, A. Pardo, B. Ghanem, and M. Alfarra, 'Test-Time Adaptation for Combating Missing Modalities in Egocentric Videos', Mar. 02, 2025, *arXiv*: arXiv:2404.15161. doi: 10.48550/arXiv.2404.15161.

[10] 'Beyond Model Adaptation at Test Time: A Survey'. [Online]. Available: https://arxiv.org/html/2411.03687v1#bib.bib293

[11] D. Wang, E. Shelhamer, S. Liu, B. Olshausen, and T. Darrell, 'Tent: Fully Test-Time Adaptation by Entropy Minimization', presented at the International Conference on Learning Representations, Oct. 2020. Accessed: Apr. 06, 2025. [Online]. Available: https://openreview.net/forum?id=uXl3bZLkr3c

[12] Y. Sun, X. Wang, Z. Liu, J. Miller, A. Efros, and M. Hardt, 'Test-Time Training with Self-Supervision for Generalization under Distribution Shifts', in *Proceedings of the 37th International Conference on Machine Learning*, PMLR, Nov. 2020, pp. 9229–9248. Accessed: Apr. 10, 2025. [Online]. Available: https://proceedings.mlr.press/v119/sun20b.html

[13] H. Zhao, Y. Liu, A. Alahi, and T. Lin, 'On Pitfalls of Test-Time Adaptation', in *Proceedings of the 40th International Conference on Machine Learning*, PMLR, Jul. 2023, pp. 42058–42080. Accessed: Apr. 10, 2025. [Online]. Available: https://proceedings.mlr.press/v202/zhao23d.html

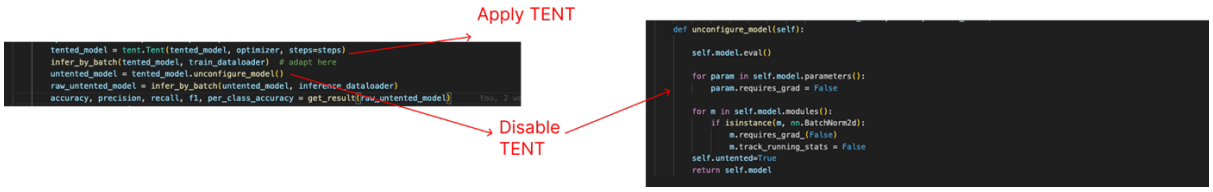
[14] J. Liang, D. Hu, and J. Feng, 'Do We Really Need to Access the Source Data? Source Hypothesis Transfer for Unsupervised Domain Adaptation', in *Proceedings of the 37th International Conference on Machine Learning*, PMLR, Nov. 2020, pp. 6028–6039.

Accessed: Apr. 06, 2025. [Online]. Available: https://proceedings.mlr.press/v119/liang20a.html
[15] Komisi Pemilihan Umum Republik Indonesia, 'Pemilu 2024 – Hasil Perhitungan Suara'. Accessed: Jan. 10, 2025. [Online]. Available: https://pemilu2024.kpu.go.id/
[16] 'Download SIREKAP 2024 2.48 Android APK File', APKPure.com. Accessed: Apr. 13, 2025. [Online]. Available: https://apkpure.com/sirekap-2024/id.go.kpu.sirekap2024/download/2.48
[17] yohanes, 'Reverse Engineering Model Handwritten Digits Recognition Aplikasi Mobile SIREKAP', Amazing Grace. Accessed: Apr. 07, 2025. [Online]. Available: https://blog.compactbyte.com/2024/02/23/reverse-engineering-model-handwritten-digits-recognition-aplikasi-mobile-sirekap/
[18] A. Farahani, S. Voghoei, K. Rasheed, and H. R. Arabnia, 'A Brief Review of Domain Adaptation', Oct. 07, 2020, *arXiv*: arXiv:2010.03978. doi: 10.48550/arXiv.2010.03978.
[19] J. Liang, R. He, and T. Tan, 'A Comprehensive Survey on Test-Time Adaptation under Distribution Shifts', Mar. 27, 2023, *arXiv*: arXiv:2303.15361. doi: 10.48550/arXiv.2303.15361.
[20] Y. Ganin and V. Lempitsky, 'Unsupervised Domain Adaptation by Backpropagation', in *Proceedings of the 32nd International Conference on Machine Learning*, PMLR, Jun. 2015, pp. 1180–1189. Accessed: Apr. 16, 2025. [Online]. Available: https://proceedings.mlr.press/v37/ganin15.html
[21] Y. Sun, E. Tzeng, T. Darrell, and A. A. Efros, 'Unsupervised Domain Adaptation through Self-Supervision', Sep. 29, 2019, *arXiv*: arXiv:1909.11825. doi: 10.48550/arXiv.1909.11825.
[22] Y. Sun *et al.*, 'Learning to (Learn at Test Time): RNNs with Expressive Hidden States', Apr. 03, 2025, *arXiv*: arXiv:2407.04620. doi: 10.48550/arXiv.2407.04620.
[23] S. Ioffe and C. Szegedy, 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift', Mar. 02, 2015, *arXiv*: arXiv:1502.03167. doi: 10.48550/arXiv.1502.03167.
[24] A. Krause, P. Perona, and R. Gomes, 'Discriminative Clustering by Regularized Information Maximization', in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2010. Accessed: Apr. 16, 2025. [Online]. Available: https://papers.nips.cc/paper_files/paper/2010/hash/42998cf32d552343bc8e460416382dca-Abstract.html
[25] '2024_Elections_FAQ_Final.pdf'. Accessed: Apr. 10, 2025. [Online]. Available: https://www.ifes.org/sites/default/files/2024-02/2024_Elections_FAQ_Final.pdf
[26] 'Kawal Pemilu - Real Time Election Monitoring and Data Verification'. Accessed: Dec. 06, 2024. [Online]. Available: https://kawalpemilu.org/
[27] 'APK decompiler - decompile Android .apk ✓ ONLINE ✓'. Accessed: Apr. 13, 2025. [Online]. Available: http://www.javadecompilers.com/apk
[28] 'Netron'. Accessed: Apr. 13, 2025. [Online]. Available: https://netron.app/
[29] 'TensorFlow', TensorFlow. Accessed: Apr. 16, 2025. [Online]. Available: https://www.tensorflow.org/
[30] 'PyTorch', PyTorch. Accessed: Apr. 16, 2025. [Online]. Available: https://pytorch.org/
[31] 'ONNX | Home'. Accessed: Apr. 16, 2025. [Online]. Available: https://onnx.ai/
[32] 'tim-learn/SHOT: code released for our ICML 2020 paper "Do We Really Need to Access the Source Data? Source Hypothesis Transfer for Unsupervised Domain Adaptation"'. Accessed: Apr. 13, 2025. [Online]. Available: https://github.com/tim-learn/SHOT/

[33] D. Wang, *DequanWang/tent*. (Apr. 14, 2025). Python. Accessed: Apr. 15, 2025. [Online]. Available: https://github.com/DequanWang/tent
[34] M. Hasani and H. Khotanlou, 'An Empirical Study on Position of the Batch Normalization Layer in Convolutional Neural Networks', in *2019 5th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS)*, Dec. 2019, pp. 1–4. doi: 10.1109/ICSPIS48872.2019.9066113.
[35] S. Niu *et al.*, 'TOWARDS STABLE TEST-TIME ADAPTATION IN DYNAMIC WILD WORLD', 2023.
[36] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, 'Communication-Efficient Learning of Deep Networks from Decentralized Data', Jan. 26, 2023, *arXiv*: arXiv:1602.05629. doi: 10.48550/arXiv.1602.05629.
[37] K. He, X. Zhang, S. Ren, and J. Sun, 'Deep Residual Learning for Image Recognition', Dec. 10, 2015, *arXiv*: arXiv:1512.03385. doi: 10.48550/arXiv.1512.03385.
[38] C. Szegedy *et al.*, 'Going Deeper with Convolutions', Sep. 17, 2014, *arXiv*: arXiv:1409.4842. doi: 10.48550/arXiv.1409.4842.
[39] K. Simonyan and A. Zisserman, 'Very Deep Convolutional Networks for Large-Scale Image Recognition', Apr. 10, 2015, *arXiv*: arXiv:1409.1556. doi: 10.48550/arXiv.1409.1556.
[40] C.-L. Liu, K. Nakashima, H. Sako, and H. Fujisawa, 'Handwritten digit recognition: benchmarking of state-of-the-art techniques', *Pattern Recognit.*, vol. 36, no. 10, pp. 2271–2285, Oct. 2003, doi: 10.1016/S0031-3203(03)00085-2.

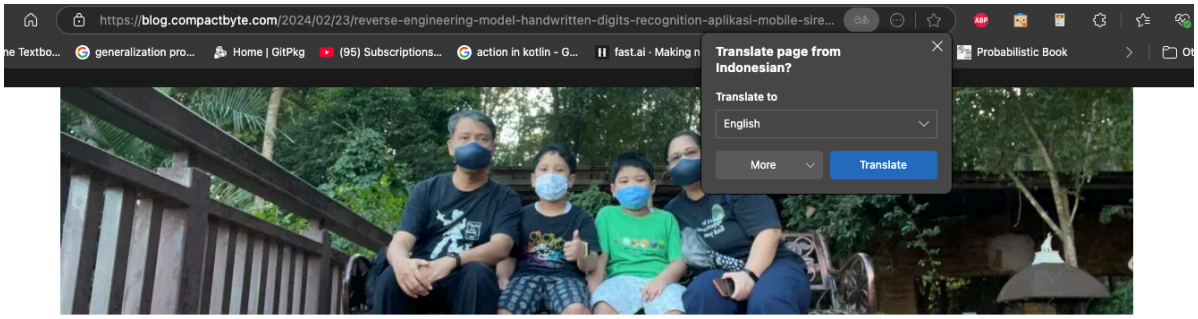
Appendix

1. Code For TENT Evaluation in Adaptation Size Evaluation



2. Code For SHOT Evaluation in Adaptation Size Evaluation

3. Translating [17] website (tested on Microsoft Edge Version 135.0.3179.73 (Official build) (arm64) Mac OS)



Reverse Engineering Model Handwritten Digits Recognition Aplikasi Mobile SIREKAP

yohanes

23/02/2024

ai, android, reverse-

Di tulisan ini saya akan melakukan reverse engineering aplikasi SIREKAP 2024, mengekstrak model TensorFlow Lite untuk inference, lalu membuat aplikasi Android untuk mengetes model tersebut. Saya bandingkan juga dengan model sederhana yang jadi contoh tutorial tensorflow lite.

CARI KONTEN

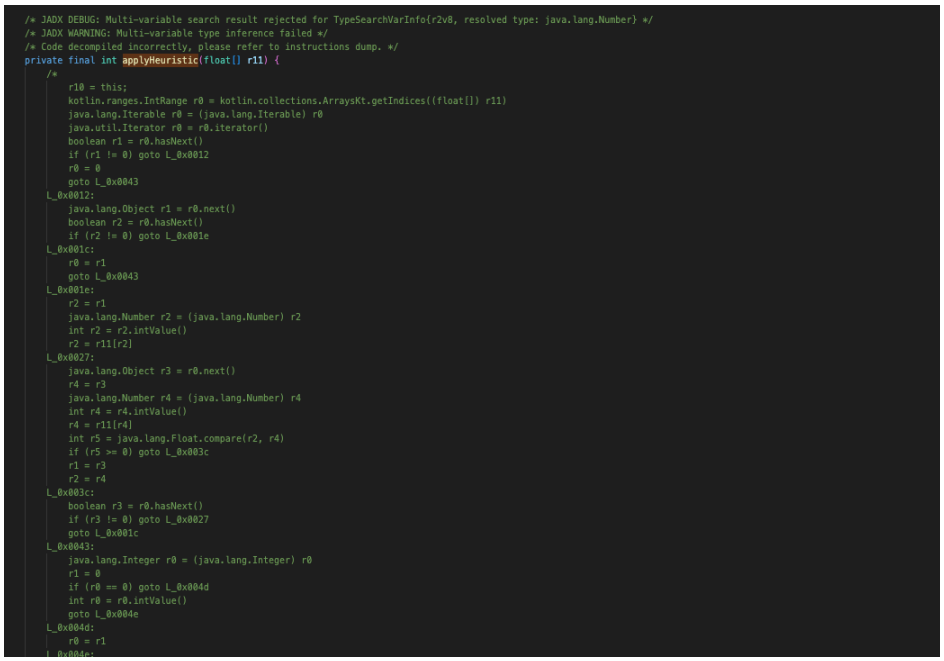
ENHANCED BY Google

SEARCH

Blog ini bertenaga surya

PAGES

4. **Gitlab Link:**
   https://github.com/Dwikavindra/Final_Year (with details written in README.MD)
5. **Post-processing ‘applyHeuristic’ code gained from decompilation using [27]**



```
/* JADX DEBUG: Multi-variable search result rejected for TypeSearchVarInfo{r2v8, resolved type: java.lang.Number} */
/* JADX WARNING: Multi-variable type inference failed */
/* Code decompiled incorrectly, please refer to instructions dump. */
private final int applyHeuristic(float[] r11) {
    /*
        r10 = this;
        kotlin.ranges.IntRange r0 = kotlin.collections.ArraysKt.getIndices((float[]) r11)
        java.lang.Iterable r0 = (java.lang.Iterable) r0
        java.util.Iterator r0 = r0.iterator()
        boolean r1 = r0.hasNext()
        if (r1 != 0) goto L_0x0012
        r0 = 0
        goto L_0x0043
    L_0x0012:
        java.lang.Object r1 = r0.next()
        boolean r2 = r0.hasNext()
        if (r2 != 0) goto L_0x001e
    L_0x001c:
        r0 = r1
        goto L_0x0043
    L_0x001e:
        r2 = r1
        java.lang.Number r2 = (java.lang.Number) r2
        int r2 = r2.intValue()
        r2 = r11[r2]
    L_0x0027:
        java.lang.Object r3 = r0.next()
        r4 = r3
        java.lang.Number r4 = (java.lang.Number) r4
        int r4 = r4.intValue()
        r4 = r11[r4]
        int r5 = java.lang.Float.compare(r2, r4)
        if (r5 >= 0) goto L_0x003c
        r1 = r3
        r2 = r4
    L_0x003c:
        boolean r3 = r0.hasNext()
        if (r3 != 0) goto L_0x0027
        goto L_0x001c
    L_0x0043:
        java.lang.Integer r0 = (java.lang.Integer) r0
        r1 = 0
        if (r0 == 0) goto L_0x004d
        int r0 = r0.intValue()
        goto L_0x004e
    L_0x004d:
        r0 = r1
    L_0x004e:
```