Assignment 2

Tutors
Zhuo Huang, Yuhao Wu

Group members

Rajiv Mehta	rmeh0608	540771859
Edward Ji	ziji4098	510477226
Yao Ke	yake9601	510459679
Daniel Kovalenko	dkov2101	500499357

Abstract

Label noise in datasets is a common challenge when building machine learning models. This is commonly due to classes that are similar and easily mistaken for one other, causing classes to be mislabelled. This study explores how a transition matrix can be built containing the overall probabilities that any class was mislabelled as any other class, and use this to compensate for the erroneous classifications in the training data. This transition matrix can be estimated by training the model on a noisy dataset and finding anchor points for that dataset by identifying instances for each class that the model is very confident in predicting. This study explores the efficacy of convolutional neural networks (CNN) and vision transformers (ViT) in producing an accurate transition matrix through the anchor point estimation method and whether they are able to improve their performance using the transition matrix. The ViT was able to estimate a transition matrix that improved its performance whilst the CNN's estimated transition matrix was unusable and decreased its performance. Both models were able to improve their performance using pre-provided, accurate transition matrices, demonstrating the overall robustness of the transition matrix approach.

1 Introduction

Label noise refers to the inaccuracies in the labels assigned to the training data, arising from issues such as human error during data annotation or ambiguities and inconsistency in the labelling guidelines [1]. This report addresses class conditional label noise where the probability of an incorrect label depends on the true class of the sample. This can often cause issues for machine learning models' performance and lead to poor predictions as seen in real world data. Solving this issue can help improve the accuracy of models while simultaneously decreasing real-world costs associated with re-labelling data or hiring external parties to do so. In this study, the authors aim to test the robustness of a *Convolutional Neural Network (CNN)* and *Vision Transformer (ViT)* with three different datasets where the given transition matrices for two datasets and a derived transition matrix for the third will be used to analyse the performance and robustness of the models.

2 Literature Review

Before starting the implementation of the classification models and testing their performance against data with noisy labels, it is first important to analyse past studies to understand the methods and performance of algorithms in the past.

'Learning with Noisy Labels Incorporating Fairness and Privacy Concerns' by Songuia Wu discusses this idea in chapter 3 of her doctoral study *LNL from a Noise Reduction Perspective* [2]. In this chapter, the author proposes the idea to transform the training data labels to be pairwise instead of point wise meaning that each data point is associated with another data point and the label is converted to a '1' if similar and '0' if not. This information is stored in a similarity transition matrix. For this experiment 3 imaged based datasets, *MNIST, CIFAR-10, and CIFAR-100*, were compared with clean and noisy labels where noisy labels were artificially created based on a class transition matrix. These datasets were then trained on a *Deep Neural Network (DNN)* where the means and standard deviations of classification accuracy were measured. This was compared with other baselines such as 'Forward' and 'Reweight' where it was found that the proposed method of a modified Forward loss with the similarity labels was very robust on the 3 datasets with similar to better results than the compared models.

Another study *Improving Robustness of CNN on Noisy Labels with Hinge Loss Function* by Gao et.al. proposed a method of increasing the robustness of CNN models on noisy labels through the implementation of hinge loss with a L_2 norm regularization term [3]. To test the robustness, they experimented on the *Fashion-MNIST* dataset where binary classification was performed on two classes at a time i.e. class 1 and 7 or class 3 and 4. Two other CNNs were introduced, one with a BCE loss and another with MSE loss, to act as a baseline comparison. The study showed that the advantages of hinge loss made the CNN less sensitive to outliers, and though it had a lower training accuracy compared to the other methods, the test accuracy was similar to the other two CNNs at each differing noise level. With a maximum noise of 30%, the hinge loss CNN had a test accuracy of 98.13% on classes 1 and 6, and a test accuracy of 92.57% when comparing classes 7 and 9.

Object Extraction From Very High-Resolution Images Using a Convolutional Neural Network based on a Noisy Large-Scale Dataset by Li et.al, tries to optimise the accuracy of a CNN on noisy data by creating a feature label noise model (FLNM) to model the noisy label distribution, then using a mutli-task deep learning framework (MDLF) to integrate the FLNM into the CNN [4]. For this experiment a large scale Massachusetts dataset is used, where labels are used to identify buildings and roads on the maps of Massachusetts. This dataset is then split to create multiple datasets for different tests in the experiment. To measure the robustness, 'Mass-BR' was created, which was a multi-class subset for multiple object classification. Noise was then artificially generated into a portion of the training data with a probability of $\frac{1}{K-1}$, which was used to compare the CNN-MLDF to the Saito-CNN proposed in *Multiple object extraction from aerial imagery with convolutional neural networks* by Saito et.al [5]. The proposed CNN-MDLF performed marginally better than the Saito-CNN with a large 8.35% model improvement on predicting buildings and a 3.1% increase on predicting roads when 70% noise was added to the training labels.

In the paper *Jigsaw-ViT: Learning jigsaw puzzles in vision transformer* by Chen et.al the authors aim to implement a vision transformer (ViT) to solve jigsaw puzzles [6]. To do this the authors had to modify a standard ViT so that it does not bias sequential pixels based on its positional embedding by shuffling the positions of patches. The Jigsaw-ViT was then designed to the re-order the shuffled patches back to their original sequence. To test the robustness of this model, 3 datasets were chosen: *Animal -10N, Food-10N and Clothing1M*, where each dataset had a varying level of label noise added (8%, 20% and 38% respectively). A baseline model, Deit-Small/16 from *Training data-efficient image transformers & distillation through attention* by Touvron et.al. was then compared to the Jigsaw-Vit with the same hyper-parameters (AdamW with a weight decay of 0.05, over 400K iterations and a batch size of 128). The baseline model, DeiT-Small/16, achieved a strong performance on datasets with lower noise, suggesting the robustness of Vision Transformers in handling label noise. Looking at the results from the varying noise levels, the Jigsaw-ViT performed marginally better with a 2.5% increase on low noise models and 0.4% increase on high noise level 'Clothin1M' datasets, indicating that the proposed ViT model is robust.

The paper *KMT-PLL: K-Means Cross-Attention Transformer for Partial Label Learning* by Fan et.al also analyses the robustness of vision transformers with label noise through their study of the partial label learning problem, where the model learns a set of candidate labels where only one is correct [7]. In this study the authors implemented a K-Means Cross-Attention Transformer for Partial Label Learning (KMT-PLL) which incorporates a K-means cross-attention mechanism, enabling the model to continuously learn cluster centres. The cluster centres represent the label classes, which assists the model in associating instances with the most relevant labels. The model

then includes a modified cross-entropy loss function to assign weights to the candidate label based on relevance to the instance, improving its overall accuracy. To test this model, the *CIFAR-10, CIFAR-100 and Fashion-MNIST* were used as benchmark datasets for testing robustness while the other image based datasets in the experiment were used to test the performance of the KMT-PLL against non deep learning models such as KNN. It was found that when compared to the other proposed models, the KMT-PLL model performed similar in terms of prediction accuracy and computation time. However, it was also found that when there was a large increase in noise labels, the clustering centre of each was be affected by the noise labels during the learning process and produce large fluctuations, thereby reducing the convergence speed and increasing the computational expense of training. Though, as this was the case for the other partial label learning models the proposed KMT-PLL model was still relatively robust.

3 Methodology

3.1 Datasets

3.1.1 Fashion MNIST 0.3 Dataset

The Fashion MNIST dataset is a popular benchmark for evaluating computer vision algorithms in the field of machine learning. It is a collection of minimalistic and flat images of fashion items such as T-shirts and dresses. The images are (28×28) in size and gray-scale as seen in Figure 1. This experiment used a processed version with only 24000 samples in the training (and validation) set and 4000 in the testing set. The training set has class-conditional noise such that each sample has a 30% chance to be wrongly labelled ($P(\hat{y} \neq y) = 0.3$), hence the name. More specifically, the transition matrix T given the classes is as follows.

$$T = \begin{bmatrix} 0.7 & 0.3 & 0 & 0 \\ 0 & 0.7 & 0.3 & 0 \\ 0 & 0 & 0.7 & 0.3 \\ 0.3 & 0 & 0 & 0.7 \end{bmatrix}.$$

Figure 1: random samples from Fashion MNIST 0.3 dataset

3.1.2 Fashion MNIST 0.6

The Fashion MNIST 0.6 dataset (seen in Figure 2) is also derived from the Fashion MNIST dataset, similar to the 0.3 variant in 3.1.1, but the label has a 60% chance of being wrongly labelled. More specifically, it has a transition matrix as follows:

$$T = \begin{bmatrix} 0.4 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.4 & 0.2 & 0.2 \\ 0.2 & 0.2 & 0.4 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.4 \end{bmatrix}.$$

Figure 2: random samples from Fashion MNIST 0.6 dataset

3.1.3 CIFAR10 Dataset

The CIFAR-10 dataset consists of coloured images of airplanes, birds, cars and cats (see Figure 3). Our version is a reduced dataset consisting of 20000 training (and validation) samples and 4000 testing samples. The dimension of the images are (32×32) in RGB. The training set has class-conditional noise, but the transition matrix T is unknown, and was estimated in this experiment.

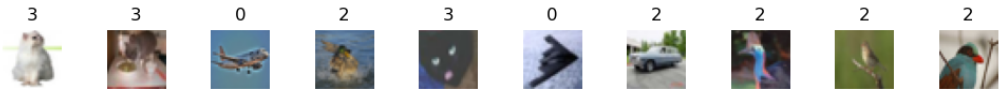

Figure 3: random samples from CIFAR-10 dataset

3.2 Data Preprocessing

As the three datasets are substantial in size it is important to first apply some preprocessing techniques to reduce the computational expense of training. To do this, the data was normalised by dividing all the pixel values by 255, shifting the range from $[0 - 255]$ to $[0 - 1]$. By scaling the data via normalisation, the variance in the pixel values remains the same while also reducing the bias the models can have when training with larger values, overall reducing the computational expense of training.

3.3 The Experiment's Custom Loss Function: Negative Log-Likelihood Loss with Transition Matrix Alteration

For this experiment, a custom loss function was developed that incorporated a transition matrix with negative log-likelihood (NLL) [8]. With normal negative log likelihood, the model's predicted probability for a certain class is used as the input for the natural logarithm, i.e. $\log p_i$, where p_i is the probability that the model assigns to the correct class. The value is then made negative so that the loss can be expressed as a positive value, since p_i is a decimal number between 0 and 1, causing the value of $\log p_i$ to become negative. This creates a final equation of $-\log p_i$.

The closer the value of p_i is to one, the closer the value of $-\log p_i$ is to zero, minimising the loss value. Intuitively this makes sense - as the model becomes more confident that the input is of the correct class, the less it should be penalised. As the value of p_i decreases (i.e. the model is less confident about predicting the correct class), the greater the value of $-\log p_i$ becomes. This is derived from the intuition that the less confident the model is, the more it should be penalised. Additionally, the usage of the natural logarithm allows for very small probability values (i.e. an extremely innacurate model) to be assigned much greater loss values compared to more confident probabilities, which derives from the intuition that more extreme errors should be disproportionately penalised compared to less extreme errors (i.e. the model only being 10% confident of the correct class is a much worse error than the model being 40% confident of the correct class).

In addition to the NLL loss, a transition matrix was applied to the probabilities predicted by the model before they were fed into the NLL formula, due to the fact that the experiment was run on noisy training data. The transition matrix follows the logic that there is a probability relationship between the noisy labels and the true labels:

$$P(\tilde{Y} = j|x) = \sum_{i=1}^{C} P(\tilde{Y} = j|Y = i)P(Y = i|x)$$

Where $\tilde{Y}$ refers to a noisy label, j refers to the class that the noisy label takes on, x refers to a training example, C refers to the number of classes, Y refers to a clean label, and i refers to the specific class that a clean label takes on. Expressing this in matrix form for all the classes gives us the following:

$$\begin{bmatrix} P(\tilde{Y}=1|x) \\ P(\tilde{Y}=2|x) \\ \vdots \\ P(\tilde{Y}=C-1|x) \\ P(\tilde{Y}=C|x) \end{bmatrix} = \begin{bmatrix} P(\tilde{Y}=1|Y=1) & \dots & P(\tilde{Y}=1|Y=C) \\ \vdots & \ddots & \vdots \\ P(\tilde{Y}=C|Y=1) & \dots & P(\tilde{Y}=C|Y=C) \end{bmatrix} \begin{bmatrix} P(Y=1|x) \\ P(Y=2|x) \\ \vdots \\ P(Y=C-1|x) \\ P(Y=C|x) \end{bmatrix}$$

The middle matrix (the one with no x terms) is the transition matrix, and describes the values that the probabilities output by a model need to be multiplied by in order to reflect the skewed probabilities of the noisy data. This is because when a model is learning from the data, it is learning and differentiating features from the input data, and outputting probabilities based on the input features of the data. However, when it outputs the class probabilities, this assumes that the labels are correct. Multiplying by the transition matrix allows the outputs to reflect the probability that the label is wrong in the training data, and thus adjust the weights accordingly. Once the training session is complete, the transition matrix can be removed from the model's prediction process, creating unaltered outputs, and thus give predictions on new, unseen data, despite being trained on noisily-labeled data. Assuming the transition matrix is denoted as follows:

$$T = \begin{bmatrix} P(\tilde{Y}=1|Y=1) & \dots & P(\tilde{Y}=1|Y=C) \\ \vdots & \ddots & \vdots \\ P(\tilde{Y}=C|Y=1) & \dots & P(\tilde{Y}=C|Y=C) \end{bmatrix}$$

The loss function used therefore becomes:

$$\arg\min \frac{1}{N} \sum_{i=0}^{N} -\log(Tf(x_i))$$

Where N is the total number of training examples in a batch, T is the transition matrix, f is the model being used, and x_i is a training example.

For the two FashionMNIST datasets, the transition matrix was pre-provided. However, for the CIFAR10 dataset, the transition matrix was estimated. The section below details how the transition matrices were estimated.

3.4 Transition Matrix Estimation Method

To estimate a transition matrix, we first train a model using the original, unmodified NLL loss function. Given a sample x_i in the dataset, the model can predict the probability distribution of x_i being in each class $\hat{y}_i = f(x_i)$. Once the model is trained, we identify **anchor points** in the output probabilities $\hat{y}_i$'s. Anchor points are instances in the data where the model's output has the highest confidence in a specific class, meaning the i-th component of the output probability vector has the greatest value compared to the other components. Formally, we define

$$p_i = \hat{y}_{\arg\max_j(\hat{y}_{j,i})}$$

where $\text{argmax}_j(\hat{y}_{j,i})$ is the index j of the sample x_j that maximizes the i-th component of the output probability vector $\hat{y}_j$.

These anchor points are crucial for accurately estimating the transition matrix. The estimated transition matrix, denoted as $\hat{T}$, is constructed using these output probability vectors. Specifically, $\hat{T}$ is represented as $[p_1, p_2, \dots, p_C]$, where p_i is the output probability vector at the anchor point corresponding to class i. This approach leverages the model's confidence in predicting certain classes to infer the noise structure and approximate the transition matrix effectively.

To assess the performance of the noise rate estimation methods, the employed metric is the mean squared error (MSE). The MSE between the true transition matrix T and the estimated transition

matrix $\hat{T}$ is computed as:

$$\text{MSE} = \frac{1}{C \times C} \sum_{i=1}^{C} \sum_{j=1}^{C} (T_{ij} - \hat{T}_{ij})^2,$$

where C is the number of classes, T_{ij} represents the element in the i-th row and j-th column of the true transition matrix, and $\hat{T}_{ij}$ represents the corresponding element in the estimated transition matrix. The MSE provides a quantitative measure of how accurately the noise rate estimation method approximates the true transition matrix, with lower values indicating better estimation performance.

3.5 The Adam Optimiser

The optimisation algorithm used for minimising the above loss function was the Adam optimiser [9]. Traditional gradient descent methods have fixed learning rates in their update rules, which can cause problems if the selected learning rate is too large and overshoots the minimum, or is too small and takes too long to converge. The Adam optimiser attempts to rectify this by scaling the magnitude and direction of the update step for every parameter in a model with regards to the magnitude and direction of previous update steps.

The moving average of the gradient mean is: $m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$. m_t represents the value of this moving average of gradients at iteration t, with m_{t-1} representing the estimate of the previous iteration. g_t represents the gradient of the loss function with respect to the parameter being evaluated, while β_1 is the regularisation term for the moving averages. β_1 is a hyperparameter, and larger values of β_1 will prioritise the overall moving average compared to the computed gradient at time t, while smaller values will prioritise the value of the gradient at the current iteration g_t.

The squared gradient element v_t of the optimiser represents the moving average of the uncentered variance of the gradient, influencing the magnitude of the change in gradient. It is given by: $v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$. v_{t-1} represents the previous moving average of squared gradients at iteration $t - 1$, while g_t^2 represents the current squared gradient of the loss function with respect to a certain parameter. Like in the previous formula, β_2 is a regularisation term that gives priority to previous squared gradients if it is higher, and higher priority to the current squared gradient if it is lower.

Because both m_t and v_t are initialised to 0, and because the regularisation terms β_1 and β_2 are usually close to 1, this means they are biased towards 0 at the beginning of the optimisation process (i.e. the optimisation process will not take very large steps). This can be seen from the formulas: for example, with $m_{t-1} = 0$, we can see that $m_t = \beta_1 \cdot 0 + (1 - \beta_1) \cdot g_t$. Since $\beta_1 > 1 - \beta_1$ by the earlier assumption, it is clear that the model weighs the initial 0 more than the initial gradient. To counteract this problem, both the mean and uncentered variance terms of the gradient have a bias correction term applied to them: $\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$ and $\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$ respectively. When t is small (i.e. the optimisation algorithm is in its early iterations), the value of $(1 - \beta^t)$ will be smaller, as $\beta^t < 0$, which means that smaller values of t lead to larger values of β, and hence a smaller value when subtracting from 1. Since $(1 - \beta)$ is smaller, this means that the value of $\frac{m_t}{1 - \beta_1^t}$ (and without loss of generality for v_t) becomes larger, since you have the same numerator over a smaller denominator. This helps to increase the value of the initial gradient mean and variance calculations, and mitigate the bias towards zero. Conversely, as t increases over many iterations, this correction effect is diminished, since the β^t term becomes smaller (as a number between 0 and 1 multiplied by itself multiple times becomes smaller), thus causing the denominator of this term to move closer to one, and thus get closer to the unaltered value of m_t.

After the above formulas have been calculated, the update rule to a parameter is given by: $\theta_t = \theta_{t-1} - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$, where θ_t is the parameter to be updated at iteration t, α is a constant base for the learning rate, and ϵ is an error term added to the variance term to remove any chance of division by zero. ϵ is usually set to a very small value, such as 10^{-8}.

3.6 Classification Models

3.6.1 Convolutional Neural Network

As seen in the previous studies mentioned in the literature review the Convolutional Neural Network (CNN) is a popular machine learning model for image classification tasks. This is due to its strengths such as automatic feature learning and robustness to training noise [10]. Although CNNs can be prone to over-fitting as they require large amounts of data for optimal predictions, the datasets provided have large sample sizes indicating that this model is an optimal choice for testing the ability to handle noisy labels.

In our implementation of the CNN, we opted for a custom setup that first runs the training images through convolutional layers with a LeakyReLu activation function to capture the spatial hierarchies of the images. These outputs are then flattened into a one dimensional vector which is passed through a series of fully connected layers (with the exact number of layers determined as a hyperparameter) with LeakyReLu activation functions. The output of these fully connected layers is then finalised through a softmax function which outputs how likely the training example is to belong to each class.

During the training process the above custom NLL loss function is used to measure the models ability to predict new data with the *Adam Optimizer* as discussed in Section 3.5. The model then performs batch training where it is repeated for the number of specified epochs. During each training loop, the model compares the actual labels to its predicted labels and modifies the weights and errors for the next iteration to obtain higher accuracies.

In the training process grid search is used to iterate through the following hyper parameters to determine the best parameters leading to the highest accuracy:

- **conv_layers** = [1, 2, 5, 10]: This represents the number of convolutional layers in the CNN. Each layer helps the model learn different levels of image features, starting from simple edges to more complex patterns as layers increase. A higher number of convolutional layers allows the model to capture more intricate patterns but can increase the model's complexity [11].
- **conv_dimension_size** = [2, 4, 6, 8, 10]: This is the size of the filters (kernels) in each convolutional layer. The filter size how much of the image each filter scans at once. Larger filters capture broader patterns, while smaller filters detect fine-grained details [11].
- **fc_layers** = [1, 2, 5, 10]: This denotes the number of fully connected (FC) layers at the end of the network. These layers help in combining the extracted features to make final predictions. More FC layers can increase the model's ability to learn complex relationships, but too many can lead to overfitting [11].
- **fc_neurons** = [100, 500, 1000]: This represents the number of neurons in each fully connected layer. Higher values allow the model to store more information and capture complex relationships in the data but also increase the computational load and potential for overfitting [11].
- **num_epochs** = [10, 100]: This is the number of times the entire training dataset passes through the model during training. Higher epochs allow the model to learn more deeply from the data, but too many epochs can cause overfitting, where the model performs well on training data but poorly on new data [11].

3.6.2 Vision Transformer

Vision Transformers were first proposed by Dosovitskiy et al. [12] in 2020 to leverage the power of transformers in the domain of computer vision. The attention module offers a key advantage over traditional deep learning modules: the capability of global attention. Vision transformers split an image into patches, create embeddings from the patches and apply attention across the patches. This allows pixels of objects that are related semantically in different parts of the image to modify each other's embeddings through the attention mechanism and better capture the real world.

In our implementation of the vision transformer, we opted for a custom architecture based loosely around the original proposed paper. We split the images into 4 x 4 patches and converted each patch into an embedding of size 32 using a simple Linear layer. We then applied a positional embedding

to encode the original position of each patch using `posemb_sincos_2d` as taken from the existing vit-pytorch repository [13]. This was not modified as it is just a helper primitive that does not significantly impact model architecture or performance. Then, a custom Transformer architecture with two heads and the same dimension as the embeddings was used to apply the attention mechanism over the different patches. Finally a linear layer was used to transform the output into a one-hot encoding vector with the same dimension as the number of classes.

All of the above hyperparameters were chosen to build a model with the same approximate size as the optimal hyperparameters CNN that was tested.

4 Experiment and Results

Hyperparameter testing was undertaken to identify which set of hyperparameters would perform best for each dataset. For the CNN and ViT, the best hyperparameter sets are shown below in Table 1 and 2. The hyperparameter training was mainly conducted on the CNN, and we did not perform any explicit hyperparameter tuning on the ViT model to maximise fairness in comparing the two architectures. The ViT hyper parameters were instead derived from the size of the optimal CNN. By creating an architecture of roughly the same size as the CNNs, the ViTs were comparable in their complexity and allowed us to draw conclusions based on the architectural differences, not the model's size or complexity.

	Number of Training Epochs	Convolution Layers	Kernel Size	FC[1] Layers	FC Dimension
CIFAR10	100	1	8	2	1000
FashionMNIST0.3	10	1	6	5	1000
FashionMNIST0.6	10	1	7	2	1000

Table 1: Hyperparameters for CNN

Number of Patches	Vocabulary Size	Depth	Number of Heads	MLP Dimension
4×4	32	4	2	64

Table 2: Hyperparameters for ViT

Once the best hyperparameter sets were found, both models were built using these hyperparameters. Of the training data, a random 80:20 split would be taken of the training data and turned into separate training and validation sets respectively. The models would then be trained on the noisy data for 10 epochs in order to find anchor points, and then once the anchor points were found, transition matrices were estimated. For the datasets where a transition matrix was already supplied, they were trained twice in two different experiments: one with the given transition matrix and one with the estimated transition matrix. After the models were trained, they were tested on the respective test sets, and their accuracy values were recorded. This process was repeated ten times with a different 80:20 random sampling of the training data each time, providing a range of results from which the average test accuracy and standard deviation was calculated.

[1] FC stands for fully connected and refers to last few layers in the CNN.

4.1 CNN Results

Dataset	Transition Matrix	Test Accuracy Mean	Std.
CIFAR10	None	0.613725	0.017334
	Estimated	0.611975	0.016921
FashionMNIST0.3	None	0.90565	0.027129
	Estimated	0.89190	0.045729
	Given	0.93530	0.005889
FashionMNIST0.6	None	0.67660	0.064457
	Estimated	0.66985	0.064027
	Given	0.86840	0.020501

Table 3: Experiment results of CNN

The results for the CNN architecture show that overall, when using an accurate transition matrix (such as the given ones) in training, the CNN architecture does perform better in the classification task. However, there is a stark contrast in the difference between the level of improvement between the FashionMNIST0.3 dataset and the FashionMNIST0.6 dataset when using an accurate transition matrix. While the FashionMNIST0.6 dataset improved by 19.18%, the FashionMNIST0.3 dataset only improved by 2.965%, indicating that the benefits from using a transition matrix may not apply equally to all datasets, and may require further development of the underlying models to see further results. Additionally, even without using a transition matrix, the CNN architecture still achieved a 90.565% accuracy on the FashionMNIST0.3 dataset, despite training on noisy data, leaving less potential room for improvement from a transition matrix compared to the FashionMNIST0.6 dataset, which started out from a 67.66% accuracy before jumping to 86.84% accuracy using the supplied transition matrix. This is most likely explained by the lower rate of noise in the FashionMNIST0.3 dataset.

Dataset	Estimate $\hat{T}$	MSE
CIFAR10	$\begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 1.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 1.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 1.000 \end{bmatrix}$	T not given
FashionMNIST0.3	$\begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 \\ 0.001 & 0.999 & 0.000 & 0.000 \\ 0.000 & 0.001 & 0.990 & 0.008 \\ 0.065 & 0.000 & 0.000 & 0.935 \end{bmatrix}$	0.04
FashionMNIST0.6	$\begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 \\ 0.001 & 0.996 & 0.002 & 0.001 \\ 0.000 & 0.001 & 0.999 & 0.001 \\ 0.000 & 0.000 & 0.000 & 1.000 \end{bmatrix}$	0.1195

Table 4: Estimated transition matrices $\hat{T}$'s using CNN (note rows may not sum to 1 due to rounding errors)

What is consistent across all three datasets is that the anchor point method for estimating transition matricies does not seem to be a very effective one when using a CNN as the underlying model. In all three of the datasets, the average accuracy of the models actually *decreased* after applying an estimated transition matrix. Given the fact that the supplied transition matrices gave decent improvements to the performance of the models, this would indicate that methods that are more accurate than the anchor point method should be used for the estimation of the transition matrix when using a CNN. It is clear that the models are not producing very useful matrices given the values within them. This is especially true for the estimated CIFAR-10 transition matrix - it gives a

probability of 0 for the flipped values (i.e. when $\hat{Y} \neq Y$), a factor that is untrue given that it is known the dataset is noisy. The same can be said for the other two estimated transition matrices, although in their cases they have some very small non-zero probabilities assigned to flipped classes. A potential explanation for these errors is that the CNN models are too confident about certain data points, and thus when assigning the data point's probability values to the transition matrix, it provides a false probability estimate of improperly labelled data in the noisy dataset. Future experiments should consider using alternate methods to estimate the transition matrix.

From the perspective of the standard deviations of the experiments, they were all fairly stable. None of the standard deviations exceeded 0.07, and the relatively low values indicate that the model was relatively consistent across the random samplings taken of the training data. This further supports the idea that the accuracy values seen here from these experiments are a result of fundamental, structural issues with both the model and the transition matrix estimation method, rather than just luck of the draw.

4.2 ViT Results

The experimental results for the ViT architecture show that the ViT responds well to the transition matrix estimation method and the noise-robust loss function using the estimated and given transition matrices.

		Test Accuracy	
		Mean	Std.
Dataset	Transition Matrix		
CIFAR10	None	0.637109	0.004399
	Estimated	0.641431	0.005934
FashionMNIST0.3	None	0.881958	0.005779
	Estimated	0.895679	0.007114
	Given	0.913867	0.004950
FashionMNIST0.6	None	0.855225	0.008783
	Estimated	0.855615	0.013559
	Given	0.865039	0.009240

Table 5: Experiment results on the performance of ViT

The experimental results on ViT in Table 5 shows a consistent improvement in test accuracy across all three datasets when using the loss function adjusted for class conditional noise. The model performs the best on the simplest FashionMNIST0.3 dataset with less label noise than the 0.6 dataset. When using the estimated transition matrix, the model demonstrates most significant boost in performance of more than 1.3% for the FashionMNIST0.3 dataset. The performance gain on the other two more complicated datasets are marginal at less than 1%. This signifies the limitation of the anchor point method in transition matrix estimation in harsher situations. Given the true transition matrix, the ViT's perform well against FashionMNIST datasets with a 3.2% boost in test accuracy for the 0.3 variation, and a close to 1% gain in test accuracy for the 0.6 variation.

Dataset	Estimate $\hat{T}$	MSE
CIFAR10	$\begin{bmatrix} 0.888 & 0.046 & 0.052 & 0.013 \\ 0.039 & 0.899 & 0.042 & 0.020 \\ 0.053 & 0.023 & 0.838 & 0.086 \\ 0.055 & 0.019 & 0.076 & 0.850 \end{bmatrix}$	T not given
FashionMNIST0.3	$\begin{bmatrix} 0.844 & 0.108 & 0.007 & 0.040 \\ 0.036 & 0.879 & 0.079 & 0.006 \\ 0.008 & 0.033 & 0.873 & 0.086 \\ 0.087 & 0.008 & 0.038 & 0.866 \end{bmatrix}$	0.01827
FashionMNIST0.6	$\begin{bmatrix} 0.542 & 0.159 & 0.145 & 0.155 \\ 0.173 & 0.486 & 0.182 & 0.160 \\ 0.129 & 0.182 & 0.522 & 0.167 \\ 0.160 & 0.142 & 0.158 & 0.540 \end{bmatrix}$	0.00530

Table 6: Estimated transition matrices $\hat{T}$'s using ViT (note rows may not sum to 1 due to rounding errors)

The anchor point method to estimating transition matrices performs well on the ViT architecture. The MSE in Table 6 when compared to the true transition matrices in 3.1 is significantly smaller than the CNN architecture. The method demonstrates potential for estimating class conditional label noise where the T is unknown, and can be used for learning on the noise robust loss function. However, the method is still overestimating the probability that the noisy label is the true label (represented by the diagonal of the transition matrix). The anchor point method is prone to over-confidence in the model's predictions since the algorithm chooses the most confident predictions in each class by definition.

4.3 Discussion

Overall, these results point to Vision Transformers generally being more accurate than CNNs for the task of learning with label noise. In many cases, the ViT architecture achieves a higher accuracy than the CNN, such as for the CIFAR10 dataset where ViT is approximately 2-3% more accurate, or with the FashionMNIST0.6 dataset where it is approximately 18% more accurate when training without a transition matrix or estimating a transition matrix. The exception to this is the FashionMNIST0.3 dataset, where the CNN performs slightly better than the ViT architecture, performing approximately 2% better than the ViT when not using a transition matrix or when using the given one, although the performance is very similar when estimating the transition matrices. Despite this however, the results still indicate that even in these scenarios the ViT performs the task better, as the standard deviations of the accuracies across all the experiments are much lower, with the average of standard deviations for the CNN being approximately 0.03274 while the average of standard deviations for the ViT being approximately 0.00746. This further supports the notion that the ViT is more accurate, and is also more consistent in its accuracy. An explanation for this could be that the ViT architecture has generally been proven to perform similarly to if not better than CNNs in previous experiments [12], leading to higher accuracy values in this experiment.

In terms of estimating the transition matrices, the ViT architecture is once again the clear winner. When comparing the estimated transition matrices to the given ones for the FashionMNIST0.3 and FashionMNIST0.6 datasets, the ViT's MSEs are 0.01827 and 0.00530, compared to the CNN's 0.04 and 0.1195 respectively. This means that for the FashionMNIST0.3 dataset, the ViT's loss is 54.325% lower compared to the CNN, while for the FashionMNIST0.6 the ViT's loss is 95.56% lower. The respective transition matrices produced for the CIFAR10 dataset further indicate that the ViT performs better, as the CNN's transition matrix for the CIFAR10 dataset is just an identity matrix, which is not representative of the noisy dataset. The explanation for this variation in accuracy could again come down to the better historical performance of ViT architectures over CNNs - because CNNs historically [12] tend to perform worse than ViTs on image tasks such as this one, the CNN could erroneously be extremely confident about a certain training example and give a 100% chance of predicting a certain class for it. This could lead to this erroneous training example being selected as an anchor point, leading to faulty probabilities being selected for the

rows of the estimated transition matrix. Meanwhile, the ViT architecture could have more realistic probabilities for an anchor point (given noisy training data) due to its better performance, and thus those more realistic probabilities would instead be used for the transition matrix, giving the lower MSE scores we see above.

5 Conclusion

In conclusion, this experiment finds that overall, the transition matrix method for training with noisily-labelled data does help increase model accuracy, although ViT architectures tend to be better suited for training with noisy data than CNNs. This conclusion was obtained by creating a CNN model and a ViT model, and training them on three image classification tasks with noisily-labelled training data, and attempting to estimate transition matrices in doing so. The results of the experiments showed that using an accurate transition matrix helped increase model accuracy, although when a poorly estimated transition matrix was used, accuracy would slightly decrease. On average, the ViT had a higher accuracy for most of the datasets, except for the FashionMNIST0.3 dataset where the CNN was slightly more accurate. It also found that the ViT was more consistent in its accuracies, and that it was much better at predicting a transition matrix for the given datasets. This potentially could have been due to the historically better performance that ViT's see over CNNs. Future experiments should consider using alternate methods to the anchor point method for estimating transition matrices when using CNNs, in order to see whether ViTs perform better across other transition matrix estimation methods, which in turn would shed further insight as to whether ViTs are consistently better than CNNs for tasks requiring learning with label noise, or if they are only superior for the anchor point estimation method.

6 Appendix

6.1 Instructions

First, please populate the `data` folder with the `.npz` files corresponding to the three datasets used for this experiment.

You can view the source code for the models we used implemented in PyTorch in the `custom_cnn.py` and `custom_vit.py` files. These files were not used directly but instead imported into `experiments.py`.

Experiments took place in `experiments.py`. First the datasets were loaded and converted to PyTorch tensors. Then a main loop was used to run combinations of different datasets, different estimation matrices and models. Finally all metrics were recorded in the `experiments` folder. To repeat the experiments, please run `python experiments.py`.

Several core dependencies are listed below. It is recommended that you install them using an environment such as VENV or Conda.

- matplotlib
- pytorch
- numpy
- pandas
- tqdm
- einops
- fastai
- torchvision
- scikit-learn

6.2 Individual Contribution

Student ID	**Contribution**
540771859	Introduction, Literature Review, Data Preprocessing, CNN Methodology
510477226	Datasets, Experiments, ViT Results
510459679	ViT architecture/code, ViT report, abstract
500499357	CNN Code, architecture, hyperparameter testing, and results. Results discussion, conclusion, Adam optimiser and custom loss report description.

Table 7: Team Members and Contributions

References

[1] P. Lu, "A homophilous and dense graph is robust to label noise," 2022.

[2] S. Wu, "Learning with noisy labels incorporating fairness and privacy concerns," 2023.

[3] F. Gao, C.-H. Chen, H. Zhang, and S.-Y. Hsu, "Improving robustness of cnn on noisy labels with hinge loss function," in *2023 15th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*. IEEE, 2023, pp. 32–35.

[4] P. Li, X. He, X. Cheng, X. Gao, R. Li, M. Qiao, D. Li, F. Qiu, and Z. Li, "Object extraction from very high-resolution images using a convolutional neural network based on a noisy large-scale dataset," *IEEE access*, vol. 7, pp. 122 784–122 795, 2019.

[5] S. Saito, T. Yamashita, and Y. Aoki, "Multiple object extraction from aerial imagery with convolutional neural networks," *Electronic Imaging*, vol. 60, no. 10, pp. 1–9, Feb 2016.

[6] Y. Chen, X. Shen, Y. Liu, Q. Tao, and J. A. Suykens, "Jigsaw-vit: Learning jigsaw puzzles in vision transformer," *Pattern recognition letters*, vol. 166, pp. 53–60, 2023.

[7] J. Fan, L. Huang, C. Gong, Y. You, M. Gan, and Z. Wang, "Kmt-pll: K-means cross-attention transformer for partial label learning," *IEEE transaction on neural networks and learning systems*, vol. PP, pp. 1–12, 2024.

[8] J. Howard and S. Gugger, *Deep Learning for Coders with Fastai and PyTorch: AI Applications Without a PhD*. O'Reilly Media, 2020.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

[10] Tamanna, "Exploring convolutional neural networks: Architecture, steps, use cases, and pros and cons," *Medium*, 2023, available at: https://medium.com/@tam.tamanna18/exploring-convolutional-neural-networks-architecture-steps-use-cases-and-pros-and-cons-b0d3b7d46c71.

[11] Rendyk, "Tuning the hyperparameters and layers of neural network deep learning," *Analytics Vidhya*, 2021, available at: https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/.

[12] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021. [Online]. Available: https://arxiv.org/abs/2010.11929

[13] P. Wang, "vit-pytorch," https://github.com/lucidrains/vit-pytorch, 2024.