UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Andreas Baum

Cost-sensitive classification with deep neural networks

Master's Thesis (30 ECTS)

Supervisor: Meelis Kull, PhD

Tartu 2020

Cost-sensitive classification with deep neural networks

Abstract:
Traditional classification focuses on maximizing the accuracy of predictions. This approach works well if all types of errors have the same cost. Unfortunately, in many real-world applications, the misclassification costs can be different, where some errors may be much worse than others. In such cases, it is useful to consider the costs and build a classifier that minimizes the total cost of all predictions.

Earlier, cost-sensitive learning has received very little research with balanced datasets. Mostly, it has been mostly considered as one of the measures that solves the class imbalance problem. As the basis of the class imbalance problem is similar to cost-sensitive learning, we can mainly rely on the research done regarding the class imbalance problem.

The purpose of this thesis is to experiment on how successful different cost-sensitive techniques are at minimizing the total cost compared to an ordinary neural network. The used techniques involve making neural network cost-sensitive based on the output probabilities. Additionally, oversampling, undersampling and loss functions that consider the class weights are used. The experiments are performed on 3 datasets with different degrees of difficulty and they involve binary and multiclass classification tasks. Also, 3 different cost matrix types are considered. The results show that all the techniques reduce the total prediction cost compared to an ordinary neural network. The best results were achieved using oversampling and cost-sensitive output modifications for both binary and multiclass case.

Keywords: neural networks, cost-sensitive learning, binary classification, multiclass classification

CERCS: P176

Hinnatundlik klassifitseerimine sügavate tehisnärvivõrkutega

Lühikokkuvõte:

Traditsiooniliselt on klassifitseerimise eesmärk võimalikult täpne objektide klassifitseerimine. Selline lähenemine eeldab, et kõikide eksimuste hind on ühesugune. Tegelikult paljudes rakendustes vigade hinnad erinevad. Sellistes olukordades tuleb kasuks valede ennustuste hindasid arvestada ja luua klassifitseerija, mis üritaks teha ennustusi nii, et ennustuste lõpphind oleks võimalikult väike.

Varasemalt on avaldatud väga vähe selliseid teadusartikleid, mis keskenduksid hinnatundlikule klassifitseerimisele kasutades tasakaalustatud andmestikke. Hinnatundlik treenimine on varasemalt kasutust leidnud eeskätt klasside mittetasakaalulisuse probleemi lahendamisel. Kuna hinnatundliku õppimise algprobleem on väga sarnane klasside mittetasakaalulisuse probleemiga, siis me põhiliselt toetume selle valdkonna varasematele töödele.

Selle lõputöö eesmärgiks on katsetada erinevaid hinnatundlike klassifitseerijaid ja proovida teha ennustusi nii, et nende koguhind oleks võimalikult väike. Lõputöö käigus on proovitud tehisnärvivõrgu ennustusi hinnatundlikuks teha kasutades ennustatud tõenäosusi. Lisaks sellele on proovitud ülevalimist, alavalimist ja klassi kaalude kasutamist kahjufunktsioonis. Neid hinnatundlikke lähenemisi on katsetatud 3 erineva keerukusega andmestiku peal kasutades nii kahte, kui ka kümmet klassi ja kolme erinevat hinnamaatriksi tüüpi. Töö käigus saadud tulemused näitavad et kõik kasutatud hinnatundlikud lähenemised aitavad vähendada ennustuste koguhinda võrreldes tavalise tehisnärvivõrguga. Nii kahe kui kümne klassi puhul on kõige paremad tulemused saadud kasutades ülevalimise ja hinnatundlikku ennustuse muutmise kombinatsiooni.

Võtmesõnad: tehisnärvivõrgud, hinnatundlik õppimine, kahe klassiga klassifitseermine, mitme klassiga klassifitseerimine

CERCS: P176

Contents

1 Introduction

Traditional classification focuses on maximizing the accuracy of predictions. This approach works well if all types of errors have the same cost. Unfortunately, in many real-world applications, the misclassification costs can be different, where some errors may be much worse than others. In such cases, it is useful to consider the costs and build a classifier that minimizes the total cost of all predictions.

For instance, diagnosing a severely sick patient as healthy could result in loss of life, while diagnosing a healthy person as sick has a much smaller consequence. Other common areas that benefit from cost-sensitive predictions are time and money, where the determination of misclassifications cost is relatively straightforward.

Earlier, cost-sensitive learning has found the most use addressing the class imbalance problem, where the dataset contains a few instances from one class and numerous cases from another class. This frequently results in a model that predicts only one class that has many instances in the training dataset. Cost-sensitivity is one of the measures that has been used for forcing the model also to predict minority class.

In the past years, many cost-sensitive techniques have been proposed [BGW03]. Although much research has been dedicated to cost-sensitive decision trees [Tin98], only a few studies discuss cost-sensitive neural networks [ZL06]. Some of the techniques used on decision trees are general-purposed, which can be used with any model, but mostly it is not possible to apply those methods on neural networks. On top of that, cost-sensitive learning has found very little research with balanced datasets [CLY15].

The purpose of this thesis is to do an experimental study with a goal to compare different techniques and give suggestions on how to minimize the total misclassification cost instead of other metrics while using neural networks. The experiments are performed on datasets that contain images because that is a domain where neural networks shine compared to other model types. The experiments involve the usage of 3 datasets with different degrees of difficulty. Each dataset initially contains ten classes that are used for multiclass classification. Additionally, a binary classification task is made from every dataset by removing 8 of the classes. Three different cost matrix types are used for each method and dataset combination. Such experiments settings cover a wide variety of possible circumstances that others may encounter.

Some of the ideas for experiments are frequently used regarding the class imbalance problem. This thesis helps to find out whether they are useful when doing cost-sensitive learning with balanced datasets, where the goal is to minimize total cost. In the end, the results of all the methods from each dataset and cost-matrix type combination are reported separately, which may help others to choose the right approach for their tasks.

This thesis is organised so that the next chapter gives the necessary background and terminology information. Chapter 3 describes what is cost-sensitive learning and where it can be applied. Chapter 4 gives an overview of the methods that are used for minimizing the total cost in this thesis. Chapter 5 describes how the experiments were set up and

how the results are compared. Chapter 6 analysis and compares the results of different techniques.

2 Terminology and background

2.1 Classification

This thesis focuses on minimizing the total cost of predictions in classification tasks using images. **Classification** is an identification task where a category (a **class**) has to be assigned to each input from a fixed set of categories. For example, if we have a dataset that contains images of cats and dogs, then the classification task would be to predict whether there is a cat or a dog in each picture.

Classification task can contain 1,2,...,k categories, where k shows the number of classes. The classes are referenced through the class serial number or they have dedicated names like A, B, C. When the number of different predictable classes is 2, then it is called **binary classification**. If there are more than two different classes, then it is called **multiclass classification**.

2.2 Basics of artificial neural networks

An artificial neural network is a machine learning technique that is capable of solving classification tasks. **Artificial neural networks** are computing systems that are inspired by human and animal brains. They consist of a collection of connected neurons. Like in human brain neurons are connected with synapses that are called **weights** and they show the strength of connections between different neurons.

The structure of artificial neural networks is layered and a human defines the number of used layers. Each **layer** contains a specified amount of neurons and each neuron is connected to all the neurons in the previous and next layer. Figure 1 is an example of a simple **feed-forward neural network** that illustrates how neurons and layers are connected. In that example, the neural network receives an input that contains three numerical values and returns one numerical value.

Like humans, neural networks learn through experience. For neural networks, it means that all the weights (edges) shown in figure 1 are learned in the **training** process. Besides, all the neurons also have a learnable value that is called **bias**. The output of each hidden layer neuron can be calculated

$$o = \sum_{i}^{N} x_i w_i + b,$$

where o is neuron output, N is the number of incoming edges, x_i is the output of previous layer's ith neuron, w_i is the weight of the edge that connects to ith neuron in previous layer and b is the bias value of current neuron. Generally, the output of neuron goes through an **activation function** that can be seen as an arbitrary function that changes all the outputs for a given layer.

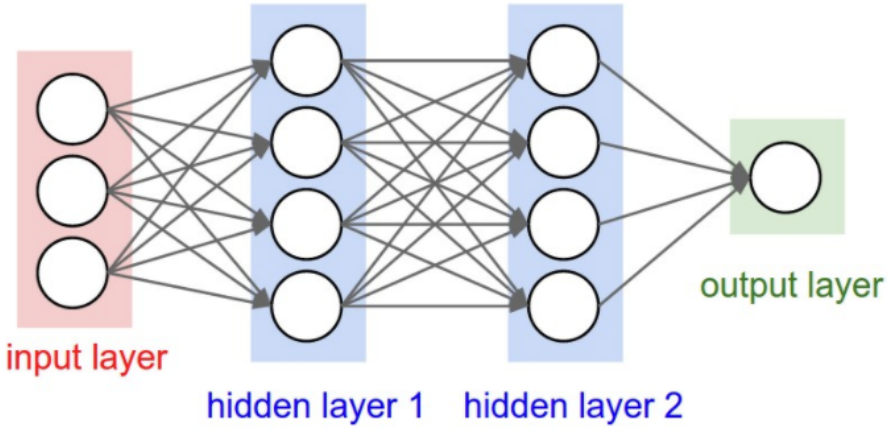

Figure 1. Feed-forward neural network example [cC20]

One commonly used activation function is **ReLU** (Rectified Linear Unit [NH10]). ReLu is mainly used on the output of the hidden layer's neurons and can be expressed as

$$f(x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{otherwise,} \end{cases}$$

where x is the output of a hidden layer neuron.

Another commonly used activation function is **softmax** and it is mostly used on the output of the network to get probabilities. The formula of softmax is

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}},$$

where z is the input vector, z_i is the model output for class i and K is the number of classes. This function applies exponential function for each element and then normalizes it, guaranteeing the sum of output vector $\sigma(\mathbf{z})$ to be 1.

However, the usage of softmax is voluntary. Sometimes we might prefer to get the raw prediction values for each class instead of probabilities. They are called **logits**.

2.3 Training a neural network

Neural networks need data to be able to learn through experience. In this thesis, the datasets consist of images (**instances**) that are shown to the neural network. Each dataset is split into three sub-datasets: training, validation and test.

The **training dataset** is used purely for training and it involves going through the dataset multiple times. One pass through the entire dataset is called an **epoch**. Generally,

neural networks do not go through the dataset one by one. Instead, they form a group of images that are processed simultaneously. The group of images is called a **batch**.

After every batch, neural network updates its weights so that it would make better predictions next time. For that the neural network first calculates the loss, then finds the gradients of the weights and updates the weights. The loss is calculated based on the output of a network and actual labels (more in section 2.4). It returns a scalar value indicating how wrong the predictions were.

After calculating the loss, we have to do **backpropagation**, which is an algorithm that goes backwards from the predicted outcome to given input and on the way calculates the **gradients** of the loss functions with respect to the neural network weights. Gradients show how much and which direction each weight of the neural network should be updated in order to make better predictions next time.

With the information about the gradients, we can update the neural network weights. An **optimization** algorithm is responsible for the updates. In this thesis, **Adam** [KB14] optimizer is used, which combines RMSprop and momentum.

Instead of using only the gradient of the last batch, **momentum** [SMDH13] also accumulates the gradient of the past batches and updates the neural network weights based on accumulated gradients. It makes the used gradient to be more graceful by reducing frequent small gradient direction changes that may appear using one batch at the time.

RMSprop [HSS12] is an optimization algorithm that controls the learning rate. **Learning rate** regulates how much each weight is changed in an optimization step based on the gradient. RMSprop makes the learning rate adaptive based on the magnitude of the gradients seen in the previous batches. It speeds up the learning process as it uses more significant learning rate at the beginning of training. At the same time, it converges well as the training speed is decreased when the gradients are smaller.

Once the learning has started, one has to decide when is the right time to stop training. In this thesis, **early stopping** [Pre97] is used for that. Early stopping needs additional dataset, which is called a **validation dataset**. It contains images which are not included in training and test datasets. Early stopping involves calculating the loss of predictions on the validation set after every epoch. The network will be trained until the validation loss has not improved for some specified number of epochs.

In the end, a neural network that has learned to predict something is called a **model**. After the training has finished, the model can be **evaluated**, which means measuring the performance on the **test dataset**. Evaluation often contains calculating the per cent of images classified correctly.

2.4 Calculating the loss

In this thesis, two different loss functions are used for calculating the loss of given predictions:

1. Cross-entropy
2. Brier score

Both of the loss functions measure the accuracy of probabilistic predictions. After getting the output probabilities, cross-entropy and Brier score can be calculated. The formula for **cross-entropy** is

$$H(p, q) = -\sum_i p_i \log(q_i),$$

where q is the predicted probability, p is the actual outcome (1 if it happened and 0 otherwise), i shows the class number and H is the cross-entropy for a single instance. In a binary case, the cross-entropy can be simplified to

$$L = -\log(\hat{y}),$$

where L is the loss of instance and $\hat{y}$ is the predicted probability of ground-truth class. Shortly, cross-entropy takes the logarithm of actual class probability.

The formula for **Brier score** [Bri50] is

$$BS = \frac{1}{N}\sum_{t=1}^{N}\sum_{i=1}^{R}(f_{ti} - o_{ti})^2,$$

where f_{ti} is the forecast probability, o_{ti} is the outcome (1 if it happened and 0 if it did not), R is the number of possible classes and N is the number of instances that we are calculating Brier score for.

If we are calculating the loss for a single instance in a binary classification task, then we can simplify the Brier score formula to

$$L = (\hat{y} - 1)^2,$$

where L is the loss of instance and $\hat{y}$ is the predicted probability of ground-truth class. Shortly, Brier score calculates the square of the difference between actual class probability and 1.

Another optional factor that can influence the loss is weight decay. **Weight decay** [KH92] is a measure to fight against **overfitting**, which can be seen as a situation, where a neural network produces good predictions on training dataset images, but performers poorly on other images. Weight decay tries to avoid this kind of situations by penalizing high weight values. It means calculating the weights penalty by taking the sum of all the squared weight values and multiplying it with some small number. Later, the weight penalty is added to already received loss.

2.5 Neural network architectures

Earlier, many neural network architectures have been proposed. Network architectures define the structure of the neural network. It includes the number of used layers, number of neurons in each layer and how they are precisely connected.

In this thesis, Residual neural network (ResNet) is used. **ResNet** [HZRS15] is a widely used convolutional neural network architecture that uses residual blocks. A **convolutional neural network** is a special type of neural network that uses convolutional layers and it is commonly applied to images. **Convolutional layers** try to learn some features from images, starting from line detection and ending up with more complex features like a human face.

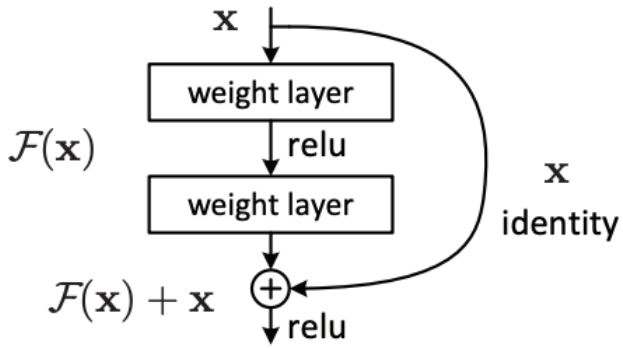

Figure 2. A residual block

In traditional neural networks, each layer feeds its output only to the next layer. With **residual blocks**, the output of one residual block is sent to multiple next residual blocks as their inputs. Figure 2 illustrates one residual block. Fundamentally, residual blocks modify traditional neural network so that they feed some layer's output to multiple layers:

1. To the next layer.
2. To a layer that is 2 or 3 layers later.

Five versions of ResNets have been proposed with a different number of layers starting from 18 hidden layers and ending up with 152 layers. In this thesis, the version with 18 hidden layers is used and the model is called **ResNet18**. This version is chosen because it has the least learnable parameters and therefore it is the fastest to train. At the same time, the model is providing good results.

2.6 Related work for cost-sensitive classification

Traditional neural networks assume that the costs for different misclassification costs are equal as the popular loss functions treat each class equally. Cost-sensitive learning, on the other hand, involves different misclassification costs. Therefore the classes should be treated differently. Hence, the current solutions might not be the best in minimizing the cost [ZL06]. This point is proven by the experiments performed in this thesis.

Another area that is strongly connected to the same problem is the class imbalance problem [LFMTH12a]. Class imbalance problem occurs with a dataset that contains a few instances from one class and many instances from another class. This frequently results in a model, that predicts only the classes that have many instances present in the training dataset. Regardlessly, the classification accuracy could be very high. At the same time, the trained model may never predict the minority class, which makes the quality of the model dubious.

In the past, the class imbalance problem [RMW16] has received a lot of research attention. Cost-sensitive learning, on the other hand, has been mostly considered as one possible technique for solving the class imbalance problem [BMM17]. At the same time, only a little research has been made in minimizing the cost instead of AUC, F-score or other metrics [EE05]. Furthermore, there are not many papers that use balanced datasets but different costs [CLY15]. Therefore the cost-sensitive experiments in this thesis are greatly inspired by the published work regarding the class imbalance problem since the source of the problem is similar for both of them.

Previous research regarding the class imbalance problem has mainly concentrated on two topics for solving the problem:

1. Resampling techniques.
2. Algorithmic level solutions.

Resampling techniques involve balancing the classes by resampling the original dataset. It is done either by oversampling the minority class or undersampling the majority class until the classes are approximately equally represented. Another option is to use both strategies simultaneously.

Both of the resampling techniques have also some clear drawbacks. Undersampling loses some potentially useful data and thus could hurt the model performance. Oversampling increases the size of the dataset, which slows down the training process. Another problem with oversampling is that some learning algorithms such as support vector machines are insensitive to oversampling - the learned classifier remains the same even if some instances are duplicated.

Another proposed technique for sampling is SMOTE (Synthetic Minority Oversampling Technique) [CBHK02]. Instead of replicating minority class instances, SMOTE

generates new synthetic minority class instances by combining features of its close neighbours. This results in new samples.

Algorithmic level solutions involve usage of existing algorithms and techniques and they can be divided into 4 categories:

- One solution is to apply different **weights** to instances from different classes. This strategy has been used for many model types. For instance, Hand and Vinciotti [HV03] have proposed a weighted version of k-NN. Veropoulos, Campbell and Cristianini [VCC99] have proposed two schemes for adjusting the sensitivity of Support Vector Machines. Ting [Tin98] have proposed instance-weighting method to induce cost-sensitive trees.

- Another solution is to create a **cost-sensitive** classification task by adding misclassification costs as shown in section 3.1 and minimizing the cost instead of other measures. Different algorithms can be useful for this task. Threshold-moving is one possible technique and section 3.3 shows an example of how that could be applied.

- Another technique is to build **one-class classifiers** that are trained for detecting one target class. In other words, it just describes objects from class and predicts whether a given instance is from the same class or not. Such classifiers are trained separately for each class. Raskutti and Kowalczyk [RK04] have shown that this technique is particularly useful when the dataset is extremely imbalanced.

- The last technique for algorithmic level solutions is building **ensembles** of different classifiers. For instance, ensembles can contain different resampling methods using the same training settings and/or algorithmic level approaches. It has been observed that ensembles frequently perform better than the models separately.

In the past, much work has been done in addressing the class imbalance problem. Many cost-sensitive learning techniques have been developed on the way, but it's hard to choose the best approach depending on your task. For that several **experimental studies** have been performed that compare different techniques.

For instance, López, Fernández, Moreno-Torres and Herrera [LFMTH12b] have done an in-depth experimental study of different cost-sensitive approaches for imbalanced classification. They tested different sampling variations: Oversampling, undersampling, SMOTE (Synthetic Minority Over-sampling Technique). Additionally, they tried out different modifications for decision-trees, support vector machines and k nearest neighbours algorithms. The experiments were focused on measuring the effectiveness of the approaches using Area Under the ROC curve.

Although many cost-sensitive learning techniques have been developed, only a few studies discuss cost-sensitive learning using neural networks. Zhou and Liu [ZL06] have

done an experimental study that has focused on the usage of neural networks in the class imbalance problem. They tried oversampling, undersampling, SMOTE, ensembles and threshold-moving. Their results suggest that cost-sensitive learning with the multiclass dataset is more difficult than with binary dataset. Additionally, the complexity of cost-sensitive learning is increased by a higher degree of class imbalance. They showed that almost all considered approaches are beneficial for binary classification, but most of them are ineffective in multiclass classification and might even cause a negative effect.

3 Cost-sensitive learning

3.1 Creating a cost matrix

A vital part of cost-sensitive learning is knowing the misclassification costs. Usually, a cost matrix is built based on the costs. Cost matrix shows the costs for all the possible class prediction and actual class combinations.

To create a cost matrix, one must be able to measure the impact on different outcomes. Some common things to measure are time and money. If the outcome is measurable by money (euros, dollars, etc) then it might be quite easy to create a profit matrix first. Profit matrix shows how much money we either win or lose depending on our prediction and the actual outcome.

Consider a business that is offering some kind of service and for each customer, it has to determine the price for a given customer. This kind of situation can happen for instance with phone service providers who offer you personalized prices by calling you. For simplicity we assume that the business has three different options:

- option A - Sell expensive service. Company will get 40 units of money as profit when a customer is willing to buy the service with the given price.
- option B - Sell cheap service. Company will get 30 units of money as profit when a customer is willing to buy the service with the given price. Even if the customer was willing to buy with the higher price, the profit would still be 30 units of money.
- option C - Do not sell service for a given customer. No profit will be gained for the company.

Another important assumption is that if the company fails to sell the product or service, then it loses 5 units of money. It can be considered as a waste of time dealing with a customer that brought no income. Based on the information we can form a profit matrix as shown below:

$$\begin{array}{r} \\ \text{Predicted A} \\ \text{Predicted B} \\ \text{Predicted C} \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 40 & -5 & -5 \\ 30 & 30 & -5 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

Once the profit matrix is ready, it can easily be converted into a *relative cost matrix* [YNWN08]. For that we do

$$c(i,j) \rightarrow -b(i,j) + b(j,j)$$

where c is the cost matrix, b is the profit matrix, i is the row number and j is the column number. In other words, creating a relative cost matrix means subtracting all column values from column largest value. This action assumes that the correct prediction has the highest gain, which should be always the case. Proceeding with the given example we will subtract values 40, 30, 0 respectively from each column and we end up with the following relative cost matrix:

$$\begin{array}{r} \\ \text{Predicted A} \\ \text{Predicted B} \\ \text{Predicted C} \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 35 & 5 \\ 10 & 0 & 5 \\ 40 & 30 & 0 \end{bmatrix} \end{array}$$

This relative cost matrix shows how much money the company would lose with each wrong prediction compared to the perfect prediction. In other words, the cost matrix shows the cost or relative cost of different misclassifications.

Cost matrix can also be made directly without a profit matrix. It is easy to do when all the benefits are negative, meaning that they are costs. This situation probably happens more often when the impact measure is time and then each cost matrix value can be assigned as the time spent on a task. An example case would be if some kind of a device stopped working for some reason and you know a list of things that could fix the problem. Then each possible fix you try that does not work can be considered as a cost.

3.2 Evaluating a cost sensitive model

For evaluating a cost-sensitive model one needs a cost matrix and a model that is capable of making predictions. Assume that we have a neural network that predicts customers' purchase intentions. Based on the predictions we can create a confusion matrix, where each row of the matrix represents the predicted class of the instances and each column represents the actual class of the instance. From a confusion matrix, it is possible to see how many times each prediction and actual class combination occurred. For example, consider the following confusion matrix:

$$\begin{array}{r} \\ \text{Predicted A} \\ \text{Predicted B} \\ \text{Predicted C} \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 110 & 12 & 17 \\ 9 & 115 & 5 \\ 21 & 5 & 132 \end{bmatrix} \end{array}$$

From the given confusion matrix, some readable examples are that option A was predicted correctly 110 times and option C was predicted 5 times as option B.

Confusion matrix could be used for measuring the performance of the trained model. For that, we could do element-wise multiplication between the cost matrix and the

confusion matrix. If we use the same example matrices as shown before we would get the following matrix:

$$\begin{array}{r} \\ \text{Predicted A} \\ \text{Predicted B} \\ \text{Predicted C} \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 420 & 85 \\ 90 & 0 & 25 \\ 840 & 150 & 0 \end{bmatrix} \end{array}$$

This matrix displays how much each misclassification contributes to the total cost, which can be calculated by summing up all the matrix values. In this example, the total cost would be 1610. It means that with the perfect classification model the company would have earned 1610 units of money more.

3.3 Making the optimal decision

It is not always possible to have 100% accurate predictions. Hence, in some cases, it might be beneficial to minimize the total prediction cost instead of accuracy.

To make the theoretically optimal predictions, we first need a model that predicts probabilities for each class. Additionally, a cost matrix is needed that shows how costly each misclassification is. Based on the predicted class probabilities and received cost matrix we can calculate the expected cost for each class. The expected cost [Elk01] can be calculated with the following formula:

$$L(x, i) = \sum_j P(j|x)C(i, j)$$

where x is the instance we predict, C is the used cost matrix, $P(j|x)$ is the predicted probability of jth class, i is the predicted class and j is the actual class. Expected cost shows the estimated average cost obtained over many samples with the same expected probability distribution.

In other words, to find the expected cost for a single class, we have to go through each class and see what can we expect if we predicted the class. It means that we multiply the predicted probability with the actual class cost for a given prediction class. Later we sum up the results for given prediction class and that is the expected cost for a given prediction. The same steps should be repeated for each prediction class and in the end, we want to predict the class that gave us the smallest expected cost.

Assume that we have a model that predicts instance probabilities to be 0.3, 0.2, 0.5 for options A, B, C respectively and we are using the example cost matrix received in section 3.1. In that case, the expected costs for each option are:

$L(x, 1) = 0.3 \cdot 0 + 0.2 \cdot 35 + 0.5 \cdot 5 = 9.5$
$L(x, 2) = 0.3 \cdot 10 + 0.2 \cdot 0 + 0.5 \cdot 5 = 5.5$

$$L(x, 3) = 0.3 \cdot 40 + 0.2 \cdot 30 + 0.5 \cdot 0 = 18.0$$

Based on that the best choice is to predict class 2 because its expected cost is the smallest.

3.4 Calibrating the probabilities

The previous section gives the best possible predictions under the assumption that the predicted probabilities are perfectly calibrated. The probabilities are perfectly calibrated if the probability of each class equals to the per cent of times the class is correct. For instance, if our model predicts something infinitely many times with the confidence of 0.8, then the prediction should be correct exactly 80% of the times.

In reality, the probabilities received after the softmax function are not perfectly calibrated. This affects the effectiveness of the optimal decision. To fix this problem, some calibration algorithms have been proposed. Hence, instead of applying the theoretical optimal decision to probabilities received from softmax, we first calibrate them and then make the optimal decision based on those probabilities as shown in section 3.3.

One method that has been applied for calibrating the neural networks probabilities is Platt scaling. **Platt scaling** [Pla00] is a parametric calibration method that uses logits as features for a logistic regression model with parameters $a, b \in R$. The logistic regression model should not be trained using the same dataset that was used for training the main neural network. Hence, a validation dataset has to be used instead of the training dataset.

Another calibration method is **temperature scaling**, which is an extension to Platt scaling. Guo et al [GPSW17] have done comparisons between different calibration methods using convolutional neural networks and temperature scaling gave the best results for them. On top of that, it is computationally cheap to use. Temperature scaling extension simplifies Platt scaling by using a single scalar parameter $T > 0$ for all classes, and the T is called temperature. Given the logits vector $\mathbf{z}_i$, new class confidences can be calculated with

$$\hat{q}_i = \max \sigma(\mathbf{z}_i / T),$$

where k indicates a single class and σ indicates softmax function.

4 Methods

This chapter gives an overview of all the methods used in the experiments.

4.1 Ordinary neural network

The first experiment approach contains training the most basic neural network for the classification task with no extras. In this thesis, Resnet18 with Adam optimizer is trained using cross-entropy and Brier score as the loss functions. The outcome of this approach can be considered as a baseline. Comparing other approaches with this indicates how effective they are at reducing total cost.

4.2 Cost-sensitive neural network

In this thesis, the neural networks are made cost-sensitive by making the theoretical best decisions based on received probabilities as described in section 3.3. This way we should get the best possible results from the trained model under the assumption that the probabilities are perfectly calibrated. This technique can be applied to every model.

In this thesis, we made two cost-sensitive versions from each model:

1. Uncalibrated version - The optimal decision is made based on the received probabilities after the softmax function.

2. Calibrated version - Temperature scaling model is trained on the validation dataset logits. The prediction probabilities are received after applying the trained Temperature scaling model on predicted logits.

4.3 Resampling

In class imbalance problem the resampling is used for making the classes to have an equal amount of instances. When we apply the same technique on a balanced dataset, while trying to minimize the total cost, we do the opposite. We create an imbalanced dataset based on the class weights that we find from a cost matrix. It changes the decision boundaries so that the model would do less expensive misclassifications.

Figure 3 illustrates the effect of resampling using a balanced dataset. Both of the classes are drawn using a normal distribution with a slightly different standard deviation. The first class is drawn using mean value 0 and the used standard deviation is 10. The second class uses the mean value 12 and standard deviation of 9.1.

The graph in the left has 10 000 instances in both classes, but in the right graph, each class 1 instance is represented 4 times in the dataset. From those graphs, it is easy to see that class 2 is significantly less likely to be predicted based on x value after resampling.

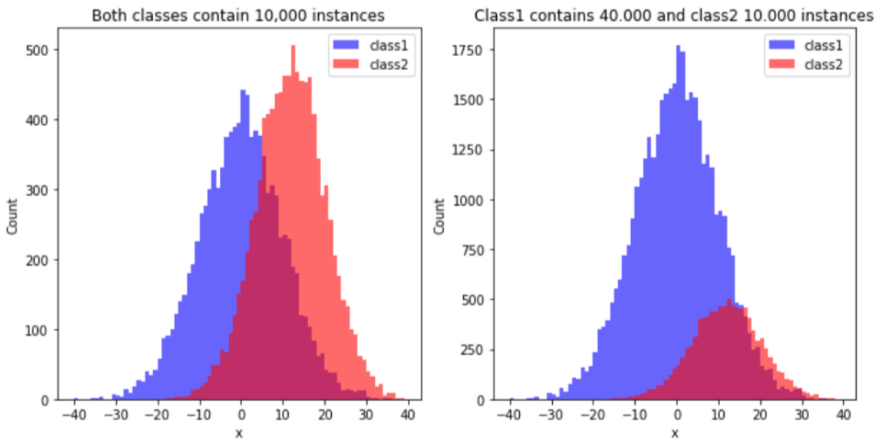

Figure 3. Resampling effect when class 1 instances count is increased 4 times.

In this thesis, two different resampling techniques are used for different class weights algorithms (see section 4.3.1):

1. Oversampling
2. Undersampling

and the resampling is done based on the calculated weights.

4.3.1 Assigning class weights

The question is how to assign class weights from cost matrices. To get cost-sensitive improvements we have to choose the resampling weights so that cheaper classes are represented more in the dataset than expensive classes.

Binary classification weights. It is easy to find the class weights from binary classification cost matrix [Elk01]. For example, if we have a matrix

$$\begin{array}{c} \\ \text{predicted} \end{array} \begin{array}{c} \text{actual} \\ \begin{bmatrix} 0 & 8 \\ 1 & 0 \end{bmatrix} \end{array}$$

then the calculated weights would be $w_1 = 8$ and $w_2 = 1$, which can be found by just taking the misclassification cost from each row. In this example, we would sample the training dataset so that the training dataset contains 8 times more class 1 instances than class 2 instances. It is the right way to do it because predicting class 2 instance when they are actually from class 1 is expensive and with such resampling, the model will not predict class 2 very easily.

Multiclass classification weights. On the other hand, choosing the class weights for multiclass cost matrix is not that trivial. In this thesis, two different class weights algorithms are proposed:

1. Label-wise weights.
2. Prediction-wise weights.

For example, if we have a cost matrix

$$\begin{array}{c} \\ \text{predicted} \end{array} \begin{array}{c} \text{actual} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 8 & 0 & 9 & 10 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

then the **label-wise weights** can be found by calculating the mean value from each **column** non-zero elements. In this case, it would mean

$w_1 = (8 + 1 + 1)/3 = 3.3333$
$w_2 = (1 + 1 + 1)/3 = 1$
$w_3 = (1 + 9 + 1)/3 = 3.6667$
$w_4 = (1 + 10 + 1)/3 = 4$

At the same time with **prediction-wise weights**, we first calculate the sum of costs in each **row**. In this example case, the sums for each row would be 3, 27, 3, 3. After that, we divide the maximum column sum (27 in this case) by each column sum. It would result in class weights

$w_1 = 27/3 = 9$
$w_2 = 27/27 = 1$
$w_3 = 27/3 = 9$
$w_4 = 27/3 = 9$

Based on the calculated weights in the example matrix, one can argue, that the prediction-wise weights are better as its weights work more similarly to binary case. But now, if we consider an example cost matrix

$$\begin{array}{c} \\ \text{predicted} \end{array} \begin{array}{c} \text{actual} \\ \begin{bmatrix} 0 & 7 & 9 & 3 \\ 1 & 0 & 9 & 3 \\ 1 & 7 & 0 & 3 \\ 1 & 7 & 9 & 0 \end{bmatrix} \end{array}$$

then the label-wise weights would be $w_1 = 1$, $w_2 = 7$, $w_3 = 9$, $w_4 = 3$, while the prediction-wise weights would be $w_1 = 1$, $w_2 = 1.46$, $w_3 = 1.73$, $w_4 = 1.12$. This time the label-wise weights give more similar weights to the binary case.

4.3.2 Making the sampled models cost-sensitive

In this thesis, every resampled model is also made cost-sensitive with and without calibration by making the optimal decision, like described in section 4.2.

However, there is a problem with making the sampled model cost-sensitive. The found class weights already affect the output of neural networks and it should already be cost-sensitive. For example, if we have two classes that are equally probable before oversampling. Then replicating one of the classes in the training dataset x times should make the class x times more probable than the other class.

To fix this problem, we propose a method that modifies the output of the sampled model so that the class weights are also considered when making the model cost-sensitive. Therefore, two different cost-sensitive modification versions are used and compared:

1. Make the sampled model cost-sensitive the same way as we did with basic neural networks.

2. Modify the sampled model output based on the class weights before making it cost-sensitive.

The second version means that the probabilities after finding softmax are divided by the class weights and then renormalized. The formula for this would be

$$Q(i|x) = \frac{\frac{P(i|x)}{w_i}}{\sum_j \frac{P(j|x)}{w_j}}$$

where $Q(i|x)$ is the new class probability for instance x, $P(i|x)$ is the resampled model's class probability after softmax, w_i shows the sampling weight for class i and $P(j|x)$ is the resampled model's probability for class j.

Later, the new probabilities are used for making the model cost-sensitive. The only problem is that calibration works based on output logits and hence, it cannot be applied there directly.

To solve this problem, the class weight modifications are still done on softmax probabilities, but then the new probabilities are turned back into pseudo-logits by taking the natural logarithm from new probabilities. The found pseudo-logits value is equal to initial logits plus-minus a constant and the calibrated probabilities are now found using the pseudo-logits.

To illustrate the difference between those versions, assume, that we have two instances, with exactly the same input, but the true label is different. In this case, the probability for both labels is 50%. Now, if we oversample one class so that it would be present 3 times in the dataset, the class probabilities should be 75% and 25% and the first cost-sensitive version uses those probabilities. The second version divides those

probabilities with the class weights, which give 25% and 25%, but after renormalizing the probabilities, they are back to 50%. Hence, in theory, the second version is better.

Note that in the example the inputs were exactly the same. At the same time, two exactly the same images cannot have different true labels. It means that the neural network output probabilities may not work exactly as shown. Hence, the second version may not always be better than the first one.

4.4 Weighted loss function

Another technique to train a cost-sensitive neural network is to modify the loss functions [KK98] so that the misclassification costs are taken into consideration when calculating the loss. The easiest way to get a cost-sensitive loss is to first calculate class weights as we did in section 4.3.1. Then for each input, the loss can be calculated by multiplying the instance loss with true class weight.

This process can be seen as replicating the input image in a given batch based on the weight. It causes a problem, where the batch size technically starts to vary based on the weights. Additionally, the average loss of the batch changes a lot between different batches. Those problems are solved by taking the weighted average of different instances' loss instead. As a result, the instances with smaller weights affect the update rule less than the ones with higher class weights.

This technique is very similar to oversampling. The main difference between oversampling and weighted loss is that with oversampling the same instance mostly gets replicated into different batches. Weighted loss on the other hand always replicates the image in one batch. It influences the learning process. When the network has seen an image already within the same epoch, then it already has updated the weights accordingly. It means that the new loss of the same input is probably different from the first one. At the same time, weighted loss functions are faster to train compared to oversampling, where the training dataset size might increase considerably. It can be a significant factor for larger datasets.

As weighted loss works similarly to sampling methods, then the class weight methods are also used the same way (section 4.3.1). Also, two different cost-sensitive methods (section 4.3.2) are used with and without calibration.

5 Experiments setup

This section gives a general overview of how the experiments were carried out. The purpose of all the experiments is to test and analyse ordinary neural networks, cost-sensitive neural networks, calibrated cost-sensitive neural networks, resampled neural networks, neural networks with weighted loss and neural networks with matrix loss for minimizing the total cost of predictions. The results from different techniques are compared and conclusions are made, which may help others to choose the best approach based on their needs. In this thesis, all the experiments are done under the assumption that the cost matrix is known before training a neural network.

The experiments involve the usage of multiple techniques that are compared using three datasets with different degrees of difficulty. Each dataset has a multiclass and binary version of it. Additionally, three different cost matrix types are used for each method and dataset combination. Each cost matrix type contains multiple different cost matrices. Such experiments settings cover a wide variety of possible circumstances that others may run into.

The training process is repeated multiple times for each technique, dataset, cost matrix type and cost matrix combination because neural networks involve a considerable amount of randomness. Due to that, one model can make the best predictions in one run, but in another run, other models may perform better even though the circumstances were the same. Therefore, by repeating the experiments multiple times we can evaluate the models by looking at how well they performed on average.

In the evaluation process, total cost and accuracy are measured for each trained model. Total cost is the most important measure to look at since we are trying to minimize it. Accuracy, on the other hand, is an effortlessly interpretable measure that people mostly care about. It can give a general understanding, how many predictions are affected by the cost-sensitive modifications.

Before every experiment, the dataset is split into training, validation and test sets. The primary train-test split is done by the dataset providers. Received training dataset is divided manually into two datasets: training and validation. Training dataset contains 90% and validation dataset 10% of the original training dataset. The data is split randomly meaning that classes may not have exactly the same proportion in training and validation sets, but it is similar.

In order to make all the experiments comparable, the same settings are used for every experiment. The used neural network architecture is ResNet18 and the optimization algorithm is Adam with weight decay. All the test are done using two loss functions - Brier score and cross entropy. All the models are trained using early stopping, where the stopping is done when the loss on validation dataset has not decreased in five consecutive epochs.

5.1 Datasets

In the experiments, 3 datasets with different degrees of difficulty are used. All datasets contain images from 10 classes that are used for multiclass classification. Additionally, binary datasets are formed from each of the datasets by removing 8 of the classes and keeping the two similar ones. The used datasets are MNIST (section 5.1.1), CIFAR-10 (section 5.1.2) and a modified version of CIFAR-10 (section 5.1.3).

5.1.1 MNIST

The MNIST [LC10] database contains handwritten digits from 0-9. The training set includes 60,000 and the test set includes 10,000 examples. The content of the database is already preprocessed, resulting in centred grayscaled digits with a size of 28x28 pixels (see Figure 4). We made also a binary version of MNIST that contains digits 4 and 9.

Figure 4. MNIST example images

MNIST is the most simple dataset that is used in this thesis. The used neural networks reach accuracy greater than 99% in MNIST. It means that every single misclassification influences the total cost significantly. Therefore, all the possible cost-sensitive modifications might be redundant and the smallest possible total cost might be achieved by maximizing the accuracy.

5.1.2 CIFAR-10

The CIFAR-10 [Kri09] dataset consists of 60000 32x32 colour images in 10 classes, with equal number of images per class. The dataset is split so that there are exactly 1000

randomly-selected images per each class in the test set, resulting in a test set of size 10,000 images. The remaining 50,000 images are placed into a training set in random order.

The classes (see Figure 5) in this dataset are plane, car, bird, cat, deer, dog, frog, horse, ship, truck and they are mutually exclusive. We also made a binary version of CIFAR-10 that contains images with cat and dog. Additionally, there is another version of the CIFAR available. It is called CIFAR-100 and it contains 100 classes instead of 10. Otherwise, the dataset is the same.

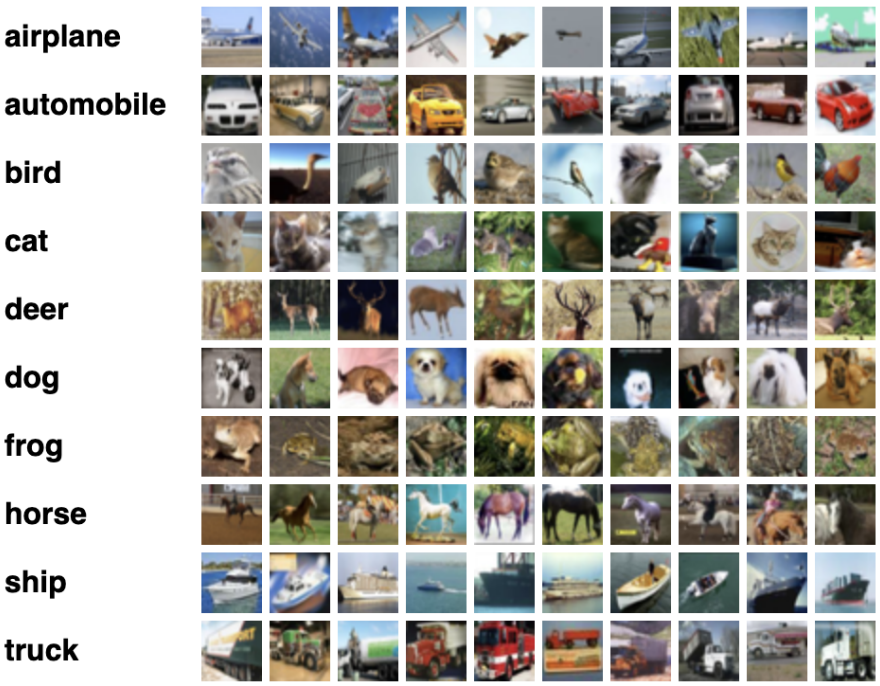

Figure 5. CIFAR-10 classes and their example images

CIFAR-10 is the second chosen dataset in this thesis and it is more complicated than MNIST. In this dataset, the reached accuracies with given experiment settings are around 87% for the multiclass case and 84% for the binary case. This dataset reflects a condition, where the model performs quite well but at the same time, it produces a considerable amount of mistakes. It allows cost-sensitive modifications to reduce the classification cost.

5.1.3 Modified CIFAR-10

More complicated datasets allow the cost-sensitive modifications to carry more importance and potential benefit compared to an ordinary neural network. Therefore, an

additional dataset is created, where the model would perform worse than it did with the previous datasets. At the same time, the image quality was kept good enough for the human eye to be able to make at least some sense of the image content. For that, we made the following changes to CIFAR-10 dataset in the following order:

1. Covering images partly.
2. Adding Gaussian noise.
3. Adding Speckle noise.
4. Adding Salt and Pepper noise.

Covering images partly [ZWZL18] involve adding 5 black boxes with size of 10x10 pixels in the images. One box is added to every corner and the last one is placed in the centre of images. This results in occluded images, where 500 pixels out of 1024 are completely black.

Gaussian noise [BJ15] is statistical noise that disturbs the gray values in digital image. The noise is drawn using the Gaussian probability density function. The noise is given as

$$p(g) = \sqrt{\frac{1}{2\pi\sigma^2}} e^{-\frac{(g-\mu)^2}{2\sigma^2}}$$

where g is grey value, σ is the standard deviation and μ is the mean. In our CIFAR-10 modification process, the mean value is zero and variance is 0.005.

Figure 6. First 32 images in CIFAR-10 training dataset

Speckle noise [BJ15] is multiplicative noise that is similar to Gaussian noise. Its probability density function uses gamma distribution. In this CIFAR-10 modification process, the mean value is one and variance is 0.01.

Salt and Pepper noise [BJ15] takes random pixels and changes the chosen pixel value to be either 0 or 255. In case of grayscaled images, random pixels are either black or white. In case of coloured images, RGB values are independent, meaning that only one of the colours might be affected by Salt and Pepper noise. In this modified CIFAR-10 5% of the pixels are affected by the noise.

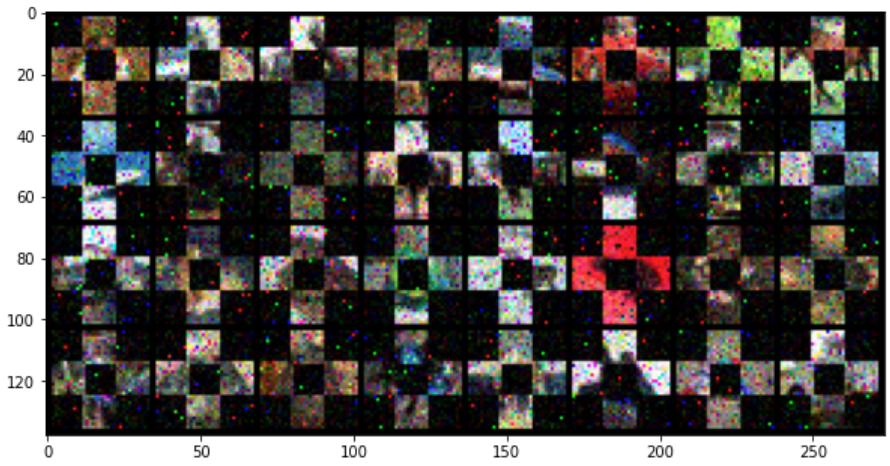

Figure 7. First 32 images in CIFAR-10 training dataset after adding modifications

Figure 6 illustrates first 32 images in the CIFAR-10 training dataset. Exactly the same images are shown in Figure 7 after adding the modifications. After those modifications, the accuracies in the multiclass case are around 67% and in the binary case around 65%.

5.2 Cost types

In real life situation, many different cost matrices can be formed. In this thesis, three different types of cost matrices are used for both binary and multiclass case. In multiclass case, the types cover different matrix scenarios and their formation is inspired by a paper from Zhou and Liu [ZL06]. Binary classification, on the other hand, does not offer many opportunities. Hence, each type just contained different misclassification costs.

5.2.1 Multiclass matrices

Type 1 - Predicting one class mistakenly is a lot more expensive than wrongly predicting other classes. Here one row contains cost values from 8-10. All the other wrong

predictions have equal cost 1. Ten different matrices are generated from this type for experiments (dataset with 10 classes) so that the expensive predicted class would be always different. An example matrix for this cost type would be

$$\begin{array}{c} \\ \text{predicted} \end{array} \begin{array}{c} \text{actual} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 8 & 0 & 9 & 10 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

In this case, the model does not want to predict one class, unless it is sure about the predictions. All the other classes are equally costly, which allows maximizing the accuracy on them. In real life, this kind of matrix represents scenarios, where one prediction has high risk compared to other choices. It can happen for example when companies invest in something larger or extend themselves somehow, while other decisions can have the same short-term impact with a smaller risk.

Type 2 - The misclassification cost is the same for each actual class. Here one column contains only value 1 and all the other columns have a random cost value from 2 to 9, which is not depending on the prediction. Ten different matrices are generated so that each time different row contains values 1. An example matrix for this cost type would be

$$\begin{array}{c} \\ \text{predicted} \end{array} \begin{array}{c} \text{actual} \\ \begin{bmatrix} 0 & 7 & 9 & 3 \\ 1 & 0 & 9 & 3 \\ 1 & 7 & 0 & 3 \\ 1 & 7 & 9 & 0 \end{bmatrix} \end{array}$$

In this case, the model should learn to predict costly actual classes slightly more often than the cheaper classes to make less expensive mistakes. In real life, this kind of matrix can be used for example in medical diagnoses, where a wrong treatment may not influence the cure of the patient and the misclassification cost depends purely on the severity of illness.

Type 3 - All the misclassification costs are selected randomly from range 1-9. Ten cost matrices are randomly generated from this type for the experiments. An example matrix from this type would be

$$\begin{array}{c} \\ \text{predicted} \end{array} \begin{array}{c} \text{actual} \\ \begin{bmatrix} 0 & 6 & 4 & 4 \\ 6 & 0 & 2 & 7 \\ 7 & 4 & 0 & 7 \\ 4 & 2 & 8 & 0 \end{bmatrix} \end{array}$$

In this case, the model has to learn different prediction and outcome combinations that should be avoided. Similar real-life cost matrices can be formed from different benefit matrices. For instance, we got a similar example matrix in section 3.1.

5.2.2 Binary matrices

Binary classification task does not provide many possibilities for cost matrices. Nevertheless, we made three types with different misclassification costs. There are two cost matrices for each type, where the second one is the first matrix transposed. Type 1 matrix misclassifications costs for the different classes are 1 and 8, type 2 costs are 3 and 4, type 3 costs are 3 and 8.

5.3 Introducing short names for experiments comparison

This sections recaps the used techniques and gives them short names that will be used while analysing and comparing the results. Table 1 shows the chosen name for each technique, how the dataset is sampled, what is the loss function and in which section it was described. Resampling cells, that are filled with "-" mean that no sampling is used.

Table 1. Introducing short names for experiments

Short name	**Resampling**	**Loss function**	**Section**
CE base	-	cross-entropy	4.1
Brier base	-	Brier score	4.1
Under CE base	undersampled	cross-entropy	4.3
Under Brier base	undersampled	Brier score	4.3
Over CE base	oversampled	cross-entropy	4.3
Over Brier base	oversampled	Brier score	4.3
Weighted CE base	-	weighted cross-entropy	4.4
Weighted Brier base	-	weighted Brier score	4.4

In addition to the models shown in table 1, cost-sensitive and calibrated cost-sensitive models are created from each of them as shown in section 4.2. The cost-sensitive models are built on top of the base model by just modifying the output of the neural networks. Therefore, no additional training is required. In the upcoming sections, those models replace the "base" with "**cs**" or "**cal cs**" depending on if the calibration was used or not.

If the method involves the usage of class weights, then also reverse-transformed cost-sensitive models and reverse-transformed calibrated cost-sensitive models are built as described in section 4.3.2 second point. In the upcoming sections, those models use endings "**rev cs**" or "**rev cal cs**" instead of "base".

In the case of multiclass analysis, two additional abbreviations are used. Abbreviation "**pw**" means prediction-wise weights and "**lw**" means label-wise weights. Both of the weights algorithms are explained in section 4.3.1. In the end, if we have an example model name "Under CE rev cal cs lw" then it would reference a reverse-transformed calibrated cost-sensitive undersampling model with label-wise weights and it was trained using cross-entropy as the loss function.

5.4 Evaluation process

The main goal of the evaluation is to understand whether the used techniques improve the total prediction cost compared to the most ordinary neural network. Another question is, which of the used methods achieve the best results and how the usage of different dataset and cost matrices influence the results.

To answer these questions, each technique, dataset and cost matrix combination has been trained multiple times. Table 2 shows the total number of trained models for each technique and the number of models that could be evaluated based on them.

Table 2. How many models were trained and how many results were received for each tecnique.

Model	**Binary models trained**	**Binary results received**	**Multiclass models trained**	**Multiclass results received**
CE base	10	60	10	300
Brier base	10	60	10	300
Under CE base	18	18	60	60
Under Brier base	18	18	60	60
Over CE base	18	18	60	60
Over Brier base	18	18	60	60
Weighted CE base	18	18	60	60
Weighted Brier base	18	18	60	60

The number of trained baseline models in this thesis is 10. The same baseline models can be applied for all the different cost matrices. As there are 3 cost types and each cost type contains 2 cost matrices in the binary case, we can get $10 \cdot 3 \cdot 2 = 60$ results from those models. In the multiclass case, we have 10 matrices in each cost type and hence the number of results in the multiclass case is $10 \cdot 3 \cdot 10 = 300$.

Other methods, on the other hand, are trained directly for a certain cost-matrix. Hence, each trained model can be applied to one cost-matrix. In the binary case, 3 models were trained for each cost matrix. As there are 2 cost matrices per type we end up with $3 \cdot 3 \cdot 2 = 18$ trained models. In the multiclass case, the number of trained models

is 2 for each cost matrix. As there are 10 cost matrices per type we get $2 \cdot 3 \cdot 10 = 60$ multiclass models.

The number of trained models per matrix was chosen so that it would be large enough to provide somewhat reliable results after averaging the single results while keeping the training times reasonable. CIFAR-10 and modified CIFAR-10 oversampling models took the longest to train. It took around 50-60 GPU hours to train 60 CIFAR-10 multiclass oversampling models for one class weights algorithm.

The received results are shown in the next chapter. They are reported so that similarly performing datasets/techniques are grouped. There are multiple tables in each group to illustrate the results. Each table describes the results of all the cost matrix types for a list of approaches. For each technique, the **accuracy** and **total cost** are reported in a format of **average +- standard deviation**.

In addition to accuracy and cost, each table compares the approaches by ordering their total costs for every trained model. Based on that the number of **times best** is reported, which shows how many times the model made the best predictions among shown models. In some cases, multiple models have the same total cost. In these cases, all of them get the winning point if they are the best ones.

Based on this information, winning results are chosen from each shown table. To choose the winner we look at the total cost and times best columns. Smaller average values are better for total cost column and higher values are better for times best column. Based on those values it is possible to make some conclusion. However, it can be hard to say if the difference between the results is significant or not. To determine the significance statistical tests could be performed.

At the end of the results section, all the winners are compared to each other and based on the comparisons the best techniques are chosen. One thing to note there is that the baseline models include more trained models than other approaches, but we can compare only the equal amount of models from every technique. Therefore the extra baseline models are not considered in the comparisons.

6 Results

6.1 Binary classification results

6.1.1 Ordinary neural networks comparison with cost-sensitive modifications

This section compares the result of ordinary neural networks using cross-entropy and Brier score as the loss function for all used dataset with different degrees of difficulty. They also have received theoretical optimal output modifications with and without calibration. Tables 3, 4 and 5 show the results using binary MNIST, CIFAR-10 and modified CIFAR-10.

Table 3. MNIST binary baseline results

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	99.35 +- 0.1	58.5 +- 21.04	0	99.35 +- 0.1	45.5 +- 7.66	3	99.35 +- 0.1	71.5 +- 17.59	3
CE cs	99.25 +- 0.15	41.5 +- 10.21	2	99.34 +- 0.09	45.1 +- 6.05	2	99.33 +- 0.12	66.2 +- 13.4	1
CE cal cs	99.24 +- 0.15	43.1 +- 12.66	3	99.34 +- 0.09	45.45 +- 6.08	2	99.33 +- 0.12	65.75 +- 12.88	1
Brier base	99.48 +- 0.1	46.8 +- 16.5	2	99.48 +- 0.1	36.4 +- 6.97	10	99.48 +- 0.1	57.2 +- 14.54	6
Brier cs	99.21 +- 0.21	**31.55 +- 8.49**	**9**	99.48 +- 0.11	**35.6 +- 7.92**	**14**	99.42 +- 0.13	51.9 +- 11.61	7
Brier cal cs	99.34 +- 0.14	33.45 +- 8.61	4	99.48 +- 0.11	35.75 +- 7.92	**14**	99.45 +- 0.11	**51.7 +- 13.51**	**9**

From the MNIST results, we can see that the Brier score gave better results than cross-entropy with all cost matrice types. On the other hand, cross-entropy gave slightly better costs on average with both versions of CIFAR-10. It might suggest that Brier score may be better with very easy datasets, while cross-entropy has a tiny advantage with more complicated datasets.

Table 4. CIFAR-10 binary baseline results

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	84.52 +- 1.11	1393.2 +- 185.68	0	84.52 +- 1.11	1083.6 +- 81.0	**5**	84.52 +- 1.11	1702.8 +- 165.67	0
CE cs	77.83 +- 3.66	714.4 +- 78.35	4	84.26 +- 1.25	1075.95 +- 83.8	2	82.88 +- 1.91	1452.05 +- 121.28	**6**
CE cal cs	74.5 +- 4.84	**681.55 +- 79.19**	**11**	84.23 +- 1.34	**1071.3 +- 87.75**	**5**	81.8 +- 2.63	**1446.35 +- 158.51**	**6**
Brier base	83.87 +- 0.93	1451.7 +- 209.46	0	83.87 +- 0.93	1129.1 +- 70.68	2	83.87 +- 0.93	1774.3 +- 171.13	0
Brier cs	77.82 +- 3.47	764.5 +- 83.86	1	83.8 +- 1.12	1110.3 +- 74.78	3	82.41 +- 1.95	1527.7 +- 134.33	2
Brier cal cs	74.83 +- 4.17	717.3 +- 83.62	4	83.73 +- 1.16	1109.2 +- 75.82	3	81.65 +- 2.11	1502.85 +- 146.41	**6**

Modifying the neural network output to be cost-sensitive improved the results for every type for every dataset for both CE and Brier. A good indicator for that is the average costs column and times best column. time The times best column from the second cost type also shows that the difference is very small. Hence, the cost-sensitive output modification effect had the most impact on type 1 and the least impact on type 2. It is expected behaviour as the differences between the costs within the misclassification cost matrices are the largest in type 1 and the smallest in type 2.

If we look at the calibration effect compared to softmax probabilities, then we can see that it only had a slight advantage in average total cost. At the same time, it mostly ends up being the best choice as shown in "times best" columns.

Table 5. CIFAR-10 modified binary baseline results

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	66.97 +- 1.27	2972.25 +- 524.29	0	66.97 +- 1.27	2311.75 +- 114.98	3	66.97 +- 1.27	3632.75 +- 391.22	3
CE cs	53.53 +- 2.77	993.0 +- 19.13	2	66.52 +- 1.43	2250.25 +- 87.72	2	61.75 +- 3.13	2742.35 +- 173.25	1
CE cal cs	52.05 +- 1.96	**983.85 +- 16.95**	**11**	66.37 +- 1.49	**2242.55 +- 95.89**	**5**	59.9 +- 2.86	**2715.3 +- 154.1**	**7**
Brier base	66.57 +- 1.18	3008.7 +- 565.97	0	66.57 +- 1.18	2340.1 +- 114.37	**5**	66.57 +- 1.18	3677.3 +- 417.66	0
Brier cs	52.54 +- 2.5	993.95 +- 26.95	2	65.89 +- 1.31	2290.9 +- 89.31	2	60.95 +- 2.59	2760.8 +- 152.82	3
Brier cal cs	51.52 +- 1.65	990.65 +- 16.19	10	65.67 +- 1.39	2291.0 +- 101.01	3	59.3 +- 2.65	2747.65 +- 129.31	6

Later, in section 6.1.3, the best results from different techniques are compared to each other. From this section calibrated cost-sensitive cross-entropy is chosen from CIFAR-10 and modified CIFAR-10 as it gives the best results in every type. In MNIST dataset it is hard to choose the best one between cost-sensitive Brier score and calibrated cost-sensitive Brier score. We just decided to go with calibrated cost-sensitive Brier score without any particular reason.

6.1.2 Class weights techniques comparison

Tables 6, 7 and 8 show the results using binary undersampling, oversampling and weighted loss functions on **MNIST dataset**.

Table 6. Binary MNIST undersampling results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	98.6 +- 0.54	46.5 +- 14.17	0	99.37 +- 0.12	42.0 +- 6.19	**3**	99.25 +- 0.12	65.83 +- 18.6	1
CE cs	97.67 +- 0.75	52.17 +- 13.66	1	99.37 +- 0.11	41.5 +- 5.19	1	99.16 +- 0.07	65.0 +- 8.39	0
CE rev cs	98.6 +- 0.54	46.5 +- 14.17	0	99.37 +- 0.12	42.0 +- 6.19	**3**	99.25 +- 0.12	65.83 +- 18.6	1
CE cal cs	97.38 +- 0.95	55.67 +- 19.78	1	99.37 +- 0.11	41.5 +- 5.19	**3**	99.18 +- 0.07	**63.17 +- 6.96**	2
CE rev cal cs	98.69 +- 0.6	44.83 +- 13.86	1	99.37 +- 0.12	42.0 +- 6.19	**3**	99.26 +- 0.13	64.83 +- 19.25	0
Brier base	98.79 +- 0.35	**44.0 +- 11.28**	0	99.46 +- 0.18	36.83 +- 11.54	**3**	99.29 +- 0.11	69.17 +- 15.24	**3**
Brier cs	97.82 +- 0.54	48.17 +- 11.47	0	99.42 +- 0.21	39.17 +- 13.38	2	99.14 +- 0.19	67.33 +- 6.72	0
Brier rev cs	98.79 +- 0.35	**44.0 +- 11.28**	0	99.46 +- 0.18	36.83 +- 11.54	**3**	99.29 +- 0.11	69.17 +- 15.24	**3**
Brier cal cs	97.91 +- 0.66	47.5 +- 13.38	0	99.41 +- 0.22	39.83 +- 13.9	2	99.2 +- 0.18	65.83 +- 11.13	0
Brier rev cal cs	99.01 +- 0.35	58.17 +- 40.08	**3**	99.48 +- 0.16	**36.0 +- 11.06**	**3**	99.28 +- 0.14	73.83 +- 26.71	2

In general Brier score seems to have a slight advantage over cross-entropy in type 1 predictions, but a slight disadvantage in type 2 (undersampling is an exception). It may suggest the usage of Brier score when the misclassification cost differences are large, but otherwise, cross-entropy seems better.

In these experiments, cost-sensitive and calibrated cost-sensitive modification did not have many benefits, as their results are quite equal to technique baseline. Also,

reverse-transformed cost-sensitive modifications are better than ordinary cost-sensitive modifications only in some cases, although it should happen more often in theory.

Table 7. Binary MNIST oversampling results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	99.21 +- 0.28	28.5 +- 11.35	1	99.49 +- 0.09	34.5 +- 5.88	**4**	99.38 +- 0.16	**47.83 +- 10.3**	1
CE cs	98.7 +- 0.52	30.5 +- 9.46	0	99.49 +- 0.03	**34.0 +- 2.65**	3	99.25 +- 0.14	50.83 +- 12.39	0
CE rev cs	99.21 +- 0.28	28.5 +- 11.35	1	99.49 +- 0.09	34.5 +- 5.88	**4**	99.38 +- 0.16	**47.83 +- 10.3**	1
CE cal cs	98.61 +- 0.5	31.17 +- 8.86	0	99.49 +- 0.03	**34.0 +- 2.65**	3	99.29 +- 0.18	49.17 +- 12.76	2
CE rev cal cs	99.26 +- 0.25	28.67 +- 13.26	2	99.48 +- 0.1	35.17 +- 6.64	**4**	99.36 +- 0.16	50.5 +- 9.95	0
Brier base	99.09 +- 0.43	**24.0 +- 7.46**	**3**	99.41 +- 0.12	40.5 +- 9.52	1	99.37 +- 0.22	48.33 +- 14.02	**3**
Brier cs	97.79 +- 1.26	46.33 +- 22.39	0	99.33 +- 0.09	44.83 +- 7.56	0	99.15 +- 0.3	58.5 +- 18.93	0
Brier rev cs	99.09 +- 0.43	**24.0 +- 7.46**	**3**	99.41 +- 0.12	40.5 +- 9.52	1	99.37 +- 0.22	48.33 +- 14.02	**3**
Brier cal cs	98.0 +- 1.15	42.17 +- 20.39	1	99.36 +- 0.11	43.5 +- 8.28	0	99.2 +- 0.25	55.0 +- 16.43	0
Brier rev cal cs	99.25 +- 0.32	31.33 +- 7.48	1	99.41 +- 0.12	40.5 +- 9.52	1	99.36 +- 0.2	52.17 +- 13.61	2

One thing to note about the binary sampling models is that the sampling baseline and reverse-transformed cost-sensitive model gives always the same results. It shows that the sampling methods affect the output probabilities like theoretically expected in section 4.3.2.

Table 8. Binary MNIST weighted loss results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	99.36 +- 0.13	33.83 +- 11.1	1	99.41 +- 0.13	38.67 +- 7.06	0	99.42 +- 0.12	53.67 +- 8.83	0
CE cs	98.57 +- 0.55	39.0 +- 13.18	0	99.41 +- 0.14	**38.17 +- 8.37**	**3**	99.32 +- 0.18	**53.0 +- 10.89**	3
CE rev cs	99.36 +- 0.13	33.83 +- 11.1	1	99.41 +- 0.13	38.67 +- 7.06	0	99.42 +- 0.12	53.67 +- 8.83	0
CE cal cs	98.75 +- 0.44	34.17 +- 11.58	1	99.41 +- 0.14	**38.17 +- 8.37**	**3**	99.32 +- 0.24	**53.0 +- 15.25**	**4**
CE rev cal cs	99.41 +- 0.13	34.0 +- 10.26	1	99.41 +- 0.14	39.33 +- 7.95	0	99.42 +- 0.1	55.33 +- 6.39	0
Brier base	99.28 +- 0.2	**30.67 +- 11.07**	1	99.38 +- 0.16	44.17 +- 11.01	1	99.32 +- 0.07	57.17 +- 7.03	0
Brier cs	98.4 +- 0.48	37.67 +- 8.86	0	99.4 +- 0.15	42.33 +- 10.37	2	99.15 +- 0.16	63.83 +- 10.57	0
Brier rev cs	99.28 +- 0.2	**30.67 +- 11.07**	1	99.38 +- 0.16	44.17 +- 11.01	1	99.32 +- 0.07	57.17 +- 7.03	0
Brier cal cs	98.75 +- 0.39	33.0 +- 7.85	2	99.41 +- 0.16	41.83 +- 10.68	**3**	99.2 +- 0.15	60.83 +- 11.22	1
Brier rev cal cs	99.46 +- 0.08	35.33 +- 16.59	**3**	99.37 +- 0.17	45.0 +- 12.62	1	99.35 +- 0.05	57.33 +- 5.88	1

Later, in section 6.1.3, undersampling is represented by reverse-transformed calibrated cost-sensitive Brier score, which results for type 1 and type 3 are quite unstable, as their standard deviation is large, but at the same time, they did the best predictions almost in every type. The representer of the oversampling technique is basic cross-entropy as it performed quite well in type 2 and type 3. In the first type, it was also one of the best cross-entropy models. The representer of the weighted loss function is calibrated cost-sensitive cross-entropy as it gave the best results in two types.

Tables 9, 10 and 11 show the results using binary undersampling, oversampling and weighted loss functions on **CIFAR-10 dataset**.

All techniques seem to slightly favour CE over Brier score in type 3. At the same time, type 1 and type 2 work differently in every approach. Brier score is slightly favoured by

Table 9. Binary CIFAR-10 undersampled results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	64.3 +- 6.74	899.5 +- 50.01	1	84.21 +- 0.9	1083.33 +- 51.75	1	78.67 +- 1.33	**1582.0 +- 129.87**	**3**
CE cs	54.55 +- 3.85	923.0 +- 59.85	0	83.94 +- 1.09	1078.17 +- 61.6	0	72.85 +- 1.78	1749.83 +- 131.52	0
CE cal cs	64.3 +- 6.74	899.5 +- 50.01	1	84.21 +- 0.9	1083.33 +- 51.75	1	78.67 +- 1.33	**1582.0 +- 129.87**	**3**
CE cal cs	50.17 +- 0.25	996.5 +- 5.02	0	83.92 +- 1.11	**1070.5 +- 62.7**	2	68.7 +- 2.35	1952.17 +- 124.57	0
CE rev cal cs	60.38 +- 5.72	876.5 +- 76.33	**3**	84.12 +- 0.94	1081.83 +- 55.37	0	77.43 +- 1.28	1609.83 +- 130.69	2
Brier base	64.43 +- 9.07	**847.83 +- 90.76**	0	84.0 +- 0.91	1101.83 +- 70.31	0	77.88 +- 3.04	1639.5 +- 207.76	1
Brier cs	56.41 +- 5.31	891.67 +- 91.13	0	83.88 +- 1.06	1085.17 +- 70.57	0	72.53 +- 5.29	1792.67 +- 302.4	0
Brier rev cs	64.43 +- 9.07	**847.83 +- 90.76**	0	84.0 +- 0.91	1101.83 +- 70.31	0	77.88 +- 3.04	1639.5 +- 207.76	1
Brier cal cs	50.33 +- 0.36	993.33 +- 7.2	0	83.86 +- 1.09	1080.83 +- 72.18	**3**	68.63 +- 4.73	1965.33 +- 282.93	0
Brier rev cal cs	61.16 +- 9.15	857.33 +- 110.17	2	84.16 +- 0.96	1084.5 +- 71.48	1	76.72 +- 2.81	1647.0 +- 203.14	2

undersampling and oversampling in type 1 and by oversampling and weighted loss in type 2. Based on those results it is hard to suggest one loss function over another.

Table 10. Binary CIFAR-10 oversampled results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	75.25 +- 6.42	702.67 +- 94.94	**2**	81.17 +- 0.7	1257.5 +- 46.74	0	81.92 +- 1.35	1465.83 +- 82.59	1
CE cs	67.04 +- 8.6	719.83 +- 149.39	1	80.43 +- 0.81	1278.17 +- 46.37	0	77.67 +- 2.79	1538.67 +- 158.37	1
CE rev cs	75.25 +- 6.42	702.67 +- 94.94	**2**	81.17 +- 0.7	1257.5 +- 46.74	0	81.92 +- 1.35	1465.83 +- 82.59	1
CE cal cs	58.81 +- 7.02	835.5 +- 132.53	0	80.19 +- 0.83	1286.17 +- 45.02	0	74.96 +- 3.22	1640.0 +- 187.43	0
CE rev cal cs	72.92 +- 5.95	675.83 +- 90.49	1	81.02 +- 0.77	1261.0 +- 50.03	0	80.88 +- 1.44	**1458.67 +- 97.08**	1
Brier base	74.67 +- 4.26	**648.83 +- 82.14**	**2**	82.97 +- 0.33	**1167.67 +- 54.92**	**2**	81.38 +- 2.34	1471.67 +- 93.06	1
Brier cs	65.34 +- 4.72	732.83 +- 72.27	0	82.39 +- 0.62	1182.33 +- 34.17	0	77.69 +- 2.96	1557.67 +- 136.75	0
Brier rev cs	74.67 +- 4.26	**648.83 +- 82.14**	**2**	82.97 +- 0.33	**1167.67 +- 54.92**	**2**	81.38 +- 2.34	1471.67 +- 93.06	1
Brier cal cs	55.87 +- 4.23	892.0 +- 69.72	0	82.27 +- 0.84	1184.0 +- 28.86	**2**	74.88 +- 3.84	1658.33 +- 200.33	0
Brier rev cal cs	72.58 +- 3.29	672.0 +- 67.65	0	82.83 +- 0.35	1170.17 +- 53.32	**2**	80.04 +- 2.67	1485.83 +- 120.8	**2**

Similarly to MNIST, the reverse-transformed cost-sensitive modifications keep the results quite equal to basic results. It is hard to prefer one over other. At the same time, the CIFAR-10 shows that reverse-transformed cost-sensitive modifications clearly perform better than ordinary cost-sensitive modifications.

Table 11. Binary CIFAR-10 weighted loss results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	76.02 +- 3.9	710.67 +- 78.33	1	83.07 +- 2.38	1144.67 +- 171.39	1	81.2 +- 2.28	**1468.83 +- 116.79**	**3**
CE cs	65.78 +- 6.11	720.5 +- 101.55	**2**	82.51 +- 2.46	1157.83 +- 166.56	0	76.67 +- 3.88	1581.67 +- 199.39	0
CE rev cs	76.02 +- 3.9	710.67 +- 78.33	1	83.07 +- 2.38	1144.67 +- 171.39	1	81.2 +- 2.28	**1468.83 +- 116.79**	**3**
CE cal cs	56.97 +- 5.33	861.83 +- 104.78	0	82.43 +- 2.45	1155.5 +- 165.96	0	74.09 +- 3.83	1681.17 +- 200.33	0
CE rev cal cs	73.71 +- 4.07	**689.17 +- 76.76**	**2**	82.9 +- 2.37	1149.17 +- 169.44	1	80.04 +- 2.01	1492.5 +- 116.53	2
Brier base	68.97 +- 5.04	732.67 +- 54.03	0	84.25 +- 0.73	1078.5 +- 43.55	**2**	79.84 +- 2.88	1552.83 +- 135.68	0
Brier cs	59.97 +- 5.18	818.17 +- 87.61	0	83.95 +- 1.17	1077.5 +- 68.61	0	74.62 +- 4.69	1695.0 +- 219.83	1
Brier rev cs	68.97 +- 5.04	732.67 +- 54.03	0	84.25 +- 0.73	1078.5 +- 43.55	**2**	79.84 +- 2.88	1552.83 +- 135.68	0
Brier cal cs	52.46 +- 3.19	952.0 +- 61.37	0	83.91 +- 1.19	**1073.33 +- 68.88**	**2**	72.15 +- 5.04	1788.5 +- 255.16	0
Brier rev cal cs	68.26 +- 4.57	727.0 +- 55.95	1	84.17 +- 0.74	1077.5 +- 41.88	0	79.12 +- 2.82	1551.33 +- 149.74	0

Later, in section 6.1.3, undersampling and weighted loss are represented by CE rev cal cs. It is hard to prefer one model over others in both undersampling and weighted

loss results. Hence the decision was made based on the average costs in all types. CE rev cal cs performs decently in all the matrix types. The representer of oversampling is basic Brier score model. It performed the best in two matrix types and in the third one it was quite close to the best models.

Tables 12, 13 and 14 show the results using binary undersampling, oversampling and weighted loss functions on **CIFAR-10 modified dataset**.

Similarly to CIFAR-10 results, cross-entropy can be slightly prefered over Brier score in type 3. At the same time, it is not possible to prefer one loss function in other types as each technique perform differently.

Table 12. Binary modified CIFAR-10 undersampling results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	51.47 +- 2.09	**994.0 +- 8.96**	3	63.97 +- 1.96	2372.67 +- 101.28	1	59.17 +- 2.24	2866.67 +- 142.42	2
CE cs	50.13 +- 0.28	997.33 +- 5.53	2	62.34 +- 3.14	2394.33 +- 156.54	1	52.13 +- 2.53	2908.67 +- 105.68	0
CE rev cs	51.47 +- 2.09	**994.0 +- 8.96**	3	63.97 +- 1.96	2372.67 +- 101.28	1	59.17 +- 2.24	2866.67 +- 142.42	2
CE cal cs	50.0 +- 0.0	1000.0 +- 0.0	2	61.51 +- 3.01	2428.5 +- 154.21	0	50.17 +- 0.29	2991.17 +- 17.61	0
CE rev cal cs	50.37 +- 0.68	995.0 +- 8.45	3	64.09 +- 1.75	2352.17 +- 102.63	1	57.28 +- 2.61	**2838.83 +- 161.86**	**3**
Brier base	50.13 +- 0.2	997.33 +- 3.94	3	65.62 +- 1.7	2338.17 +- 109.41	0	54.49 +- 3.01	2905.5 +- 115.93	1
Brier cs	50.0 +- 0.0	1000.0 +- 0.0	2	64.21 +- 2.33	2337.0 +- 130.55	**2**	50.52 +- 0.89	2976.5 +- 38.96	0
Brier rev cs	50.13 +- 0.2	997.33 +- 3.94	3	65.62 +- 1.7	2338.17 +- 109.41	0	54.49 +- 3.01	2905.5 +- 115.93	1
Brier cal cs	50.0 +- 0.0	1000.0 +- 0.0	2	63.54 +- 2.4	2364.33 +- 133.49	1	50.0 +- 0.0	3000.0 +- 0.0	0
Brier rev cal cs	50.21 +- 0.29	999.33 +- 6.42	**4**	65.48 +- 1.73	**2333.67 +- 117.67**	1	54.47 +- 3.1	2888.67 +- 73.0	0

One thing to note here is that with type 1 where the class misclassifications costs are the largest, models mostly predicts only the less costly class resulting in accuracy around 50%. Undersampling and weighted loss learned such behaviour without any cost-sensitive modifications and hence the modifications did not improve the results. With oversampling, on the other hand, the calibration has a large impact as basic oversampling and cost-sensitive changes based on softmax made many expensive mistakes. A lot better accuracy with higher cost is a clear indicator for that.

Table 13. Binary modified CIFAR-10 oversampling results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	66.02 +- 1.23	2282.67 +- 554.58	0	65.93 +- 0.57	**2274.33 +- 58.7**	0	61.77 +- 5.01	2881.0 +- 183.74	1
CE cs	62.9 +- 1.57	1699.83 +- 495.61	0	64.38 +- 1.08	2293.17 +- 65.53	0	57.08 +- 4.93	2788.83 +- 124.92	1
CE rev cs	66.02 +- 1.23	2282.67 +- 554.58	0	65.93 +- 0.57	**2274.33 +- 58.7**	0	61.77 +- 5.01	2881.0 +- 183.74	1
CE cal cs	51.01 +- 1.04	988.0 +- 14.01	1	63.27 +- 1.15	2340.5 +- 72.12	0	53.22 +- 4.07	2858.67 +- 176.33	1
CE rev cal cs	52.54 +- 2.52	979.5 +- 17.66	3	65.62 +- 0.63	2279.17 +- 64.73	**2**	59.91 +- 3.35	2747.17 +- 131.69	0
Brier base	63.99 +- 0.91	1717.67 +- 325.56	0	65.88 +- 1.54	2313.0 +- 47.96	1	61.87 +- 2.79	**2722.17 +- 158.09**	**2**
Brier cs	60.29 +- 1.32	1302.83 +- 231.42	0	65.24 +- 2.39	2276.83 +- 97.39	1	56.49 +- 3.63	2762.17 +- 140.27	0
Brier rev cs	63.99 +- 0.91	1717.67 +- 325.56	0	65.88 +- 1.54	2313.0 +- 47.96	1	61.87 +- 2.79	**2722.17 +- 158.09**	**2**
Brier cal cs	50.15 +- 0.2	997.0 +- 4.0	1	64.72 +- 2.66	2287.33 +- 117.69	**2**	52.0 +- 1.95	2899.17 +- 98.55	0
Brier rev cal cs	52.77 +- 3.08	**971.5 +- 30.26**	**4**	66.08 +- 1.89	2280.0 +- 98.49	0	59.16 +- 2.58	2731.33 +- 134.71	1

While comparing types 2 and 3 it is possible to see that cost-sensitive output modifications perform very similarly to basic technique version. Also, only a slight difference

is visible between cost-sensitive version 1 and version 2 modifications. In some cases, they seem to perform equally.

Table 14. Binary modified CIFAR-10 weighted loss results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	53.17 +- 2.13	**972.67 +- 31.49**	2	66.42 +- 2.36	2285.83 +- 127.39	0	62.27 +- 4.1	2755.17 +- 170.12	1
CE cs	50.22 +- 0.23	995.67 +- 4.57	0	65.42 +- 2.85	2273.17 +- 137.83	1	55.43 +- 3.52	2789.0 +- 150.91	1
CE rev cs	53.17 +- 2.13	**972.67 +- 31.49**	2	66.42 +- 2.36	2285.83 +- 127.39	0	62.27 +- 4.1	2755.17 +- 170.12	1
CE cal cs	50.0 +- 0.0	1000.0 +- 0.0	0	64.95 +- 2.97	2282.17 +- 144.28	**2**	52.02 +- 2.02	2896.83 +- 106.11	0
CE rev cal cs	52.08 +- 2.01	972.5 +- 26.91	1	66.34 +- 2.29	**2269.0 +- 131.57**	1	61.17 +- 3.21	**2726.67 +- 192.42**	1
Brier base	52.33 +- 2.22	983.83 +- 14.36	**3**	65.87 +- 1.67	2324.83 +- 78.05	1	58.62 +- 4.91	2761.67 +- 191.68	**2**
Brier cs	50.12 +- 0.12	997.5 +- 2.5	0	65.08 +- 1.81	2286.33 +- 80.75	0	53.62 +- 3.64	2821.67 +- 177.77	0
Brier rev cs	52.33 +- 2.22	983.83 +- 14.36	**3**	65.87 +- 1.67	2324.83 +- 78.05	1	58.62 +- 4.91	2761.67 +- 191.68	**2**
Brier cal cs	50.0 +- 0.0	1000.0 +- 0.0	0	64.73 +- 2.09	2294.5 +- 91.75	0	50.98 +- 1.42	2947.67 +- 73.38	0
Brier rev cal cs	52.42 +- 2.49	990.0 +- 14.84	0	65.87 +- 1.74	2311.0 +- 91.93	1	57.92 +- 4.62	2764.17 +- 191.67	1

From CIFAR-10 modified dataset, the chosen representers for undersampling and oversampling are Brier rev cal cs models. With undersampling, the model performed well in the first and second matrix type. At the same time, it performed decently in type 3. With oversampling, on the other hand, it performed the best in type 1 and in other types its average cost was quite close to the winning models.

The representer of the weighted loss functions is a basic Brier score model. It did the most best predictions in two matrix types. At the same time, its average costs were close to the best averages.

6.1.3 Comparing the best results from every approach

It is time to compare undersampling, oversampling, weighted loss models and give an overview which of the techniques to use. Tables 15, 17 and 18 show the chosen best models results for MNIST, CIFAR-10 and modified CIFAR-10 datasets.

Table 15. MNIST chosen best models.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
Base Brier cal cs	99.34 +- 0.14	33.45 +- 8.61	1	99.48 +- 0.11	35.75 +- 7.92	0	99.45 +- 0.11	51.7 +- 13.51	0
Under Brier rev cal cs	99.01 +- 0.35	58.17 +- 40.08	1	99.48 +- 0.16	36.0 +- 11.06	2	99.28 +- 0.14	73.83 +- 26.71	1
Over CE base	99.21 +- 0.28	**28.5 +- 11.35**	**3**	99.49 +- 0.09	**34.5 +- 5.88**	**3**	99.38 +- 0.16	**47.83 +- 10.3**	**4**
Weighted CE cal cs	98.75 +- 0.44	34.17 +- 11.58	1	99.41 +- 0.14	38.17 +- 8.37	2	99.32 +- 0.24	53.0 +- 15.25	1

Table 16 shows the best mean cost for every dataset, matrix type and technique combination. Note that there are two different baselines. The ordinary baseline shows the smallest average cost received using the most ordinary neural network either with Brier score or cross-entropy. The cost-sensitive baseline, on the other hand, shows the best result received after adding the cost-sensitive output modifications.

Table 16. Smallest binary average total costs from every technique

Technique	**Type 1 cost**	**Type2 cost**	**Type 3 result**	**Dataset**
Ordinary baseline	46.8	36.4	57.2	MNIST
Cost-sensitive baseline	31.55	35.6	51.7	MNIST
Undersampled	44.0	36.0	63.17	MNIST
Oversampled	**24.0**	**34.0**	**47.83**	MNIST
Weighted loss	30.67	38.17	53.0	MNIST
Ordinary baseline	1393.2	1083.6	1702.8	CIFAR-10
Cost-sensitive baseline	681.55	1071.3	**1446.35**	CIFAR-10
Undersampled	847.83	**1070.5**	1582.0	CIFAR-10
Oversampled	**648.83**	1167.67	1458.67	CIFAR-10
Weighted loss	689.17	1073.33	1468.83	CIFAR-10
Ordinary baseline	2972.25	2311.75	3632.75	mod CIFAR-10
Cost-sensitive baseline	983.85	**2242.55**	**2715.3**	mod CIFAR-10
Undersampled	994.0	2333.67	2838.83	mod CIFAR-10
Oversampled	**971.5**	2274.33	2722.17	mod CIFAR-10
Weighted loss	972.67	2269.0	2726.67	mod CIFAR-10

From the table 16 we can see that generally, all the cost-sensitive techniques worked better than the ordinary baseline model. The only exception is undersampling on MNIST dataset, where the model accuracies were very high. Another thing that can be noted is that in type 2 the cost-sensitive modifications helped very little, but it can be expected as the misclassification cost differences were small there.

Table 17. CIFAR-10 chosen best models.

Model	**Type 1 accuracy**	**Type 1 cost**	**Type 1 Times best**	**Type 2 accuracy**	**type 2 cost**	**Type 2 Times best**	**Type 3 accuracy**	**type 3 cost**	**Type 3 Times best**
Base CE cal cs	74.5 +- 4.84	681.55 +- 79.19	**2**	84.23 +- 1.34	**1071.3 +- 87.75**	1	81.8 +- 2.63	**1446.35 +- 158.51**	1
Under CE rev cal cs	60.38 +- 5.72	876.5 +- 76.33	0	84.12 +- 0.94	1081.83 +- 55.37	**3**	77.43 +- 1.28	1609.83 +- 130.69	0
Over Brier base	74.67 +- 4.26	**648.83 +- 82.14**	**2**	882.97 +- 0.33	1167.67 +- 54.92	0	81.38 +- 2.34	1471.67 +- 93.06	**3**
Weighted CE rev cal cs	73.71 +- 4.07	689.17 +- 76.76	**2**	82.9 +- 2.37	1149.17 +- 169.44	2	80.04 +- 2.01	1492.5 +- 116.53	2

When we try to find the best technique for binary cost-sensitive learning, then we can note that every single dataset shows that oversampling is clearly the best choice for matrix type 1, where the misclassification costs difference is large. With types 2 and 3 oversampling is still the best choice for MNIST, but both CIFAR-10 dataset versions get better results with calibrated cost-sensitive baseline models. When we compare the oversampling and baseline results differences, then we can also see that in type 2 and 3, the cost differences are smaller than they are in type 1.

Weighted loss can be considered as a decent alternative to oversampling when the

dataset is large. Oversampling can make the training process computationally very expensive. Weighted loss at the same time is computationally less expensive and its results generally are not far from oversampling.

Table 18. Modified CIFAR-10 chosen best models.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
Base CE cal cs	52.05 +- 1.96	983.85 +- 16.95	1	66.37 +- 1.49	**2242.55 +- 95.89**	0	59.9 +- 2.86	**2715.3 +- 154.1**	**3**
Under Brier rev cal cs	50.21 +- 0.29	999.33 +- 6.42	0	65.48 +- 1.73	2333.67 +- 117.67	2	54.47 +- 3.1	2888.67 +- 73.0	0
Over Brier rev cal cs	52.77 +- 3.08	**971.5 +- 30.26**	**3**	66.08 +- 1.89	2280.0 +- 98.49	**3**	59.16 +- 2.58	2731.33 +- 134.71	2
Weighted Brier base	52.33 +- 2.22	983.83 +- 14.36	2	65.87 +- 1.67	2324.83 +- 78.05	1	58.62 +- 4.91	2761.67 +- 191.68	1

Undersampling mostly gives the worst results, but at the same time, it achieved the best mean cost in type 2 CIFAR-10 dataset. In general, it seems to have the best results, when the misclassification costs have only a slight difference and where the dataset remains quite balanced. However, even in the winning case, the resulting cost is extremely similar to ordinary cost-sensitive network. Hence, it might be a better idea to use the baseline model instead.

6.2 Multiclass results

6.2.1 Ordinary neural networks comparison with cost-sensitive modifications

Tables 19, 20, 21 show the baseline model results with and without cost-sensitive output modification. Each table is dedicated to a different dataset.

Table 19. Multiclass MNIST baseline results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	99.25 +- 0.1	134.25 +- 40.58	6	99.25 +- 0.1	350.22 +- 66.9	14	99.25 +- 0.1	340.54 +- 57.02	2
CE cs	99.16 +- 0.15	**100.78 +- 18.61**	21	99.18 +- 0.14	329.46 +- 58.92	25	99.03 +- 0.17	**288.35 +- 43.82**	**33**
CE cal cs	99.17 +- 0.14	101.52 +- 19.29	**39**	99.19 +- 0.13	**329.35 +- 59.67**	**30**	99.04 +- 0.16	289.73 +- 43.56	32
Brier base	99.25 +- 0.14	134.09 +- 47.63	7	99.25 +- 0.14	354.04 +- 78.94	6	99.25 +- 0.14	337.12 +- 68.73	2
Brier cs	99.07 +- 0.23	104.21 +- 25.86	25	99.15 +- 0.18	335.13 +- 71.26	15	98.92 +- 0.24	294.81 +- 61.94	30
Brier cal cs	99.18 +- 0.17	103.66 +- 26.35	21	99.2 +- 0.16	334.71 +- 71.55	28	99.06 +- 0.19	296.0 +- 58.4	13

The results from all the datasets reveal that cross-entropy gives smaller costs on average than Brier score in every dataset for each type. Nevertheless, the cost differences aren not large. If we look at the columns that show how many times each model was the best, then we can observe that Brier score performs equally to cross-entropy in that sense. The only exception is CIFAR-10, where cross-entropy is certainly a better choice.

The tables also show that making the neural network cost-sensitive gives us obvious improvements in every case. Usage of calibration gives additional positive effect only in slightly complicated datasets. In MNIST, where the accuracies are very high, the

Table 20. Multiclass CIFAR-10 baseline results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	87.46 +- 0.97	2248.6 +- 589.5	0	87.46 +- 0.97	6304.26 +- 1136.73	0	87.46 +- 0.97	5512.25 +- 602.85	0
CE cs	86.75 +- 1.24	1572.53 +- 160.64	19	86.86 +- 1.25	5862.45 +- 1094.73	13	84.88 +- 1.64	4671.71 +- 495.78	4
CE cal cs	86.37 +- 1.22	**1528.49 +- 141.84**	**51**	86.59 +- 1.26	**5822.34 +- 1098.24**	**59**	83.97 +- 1.54	**4601.63 +- 512.97**	**71**
Brier base	86.74 +- 0.82	2370.69 +- 638.37	0	86.74 +- 0.82	6643.32 +- 1077.75	2	86.74 +- 0.82	5816.09 +- 540.95	0
Brier cs	86.03 +- 1.02	1691.22 +- 200.32	5	86.22 +- 0.99	6201.32 +- 1086.39	7	84.22 +- 1.17	4990.17 +- 496.52	1
Brier cal cs	85.61 +- 1.12	1631.97 +- 143.22	27	85.95 +- 1.07	6168.09 +- 1095.47	19	83.33 +- 1.38	4915.55 +- 488.48	24

calibrated model performs equally or even slightly worse than the uncalibrated cost-sensitive model.

Table 21. Modified CIFAR-10 baseline results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base	65.68 +- 1.55	6139.47 +- 1117.93	2	65.68 +- 1.55	17154.95 +- 2474.41	12	65.68 +- 1.55	15059.19 +- 1089.13	0
CE cs	63.54 +- 1.8	3820.43 +- 168.89	16	63.41 +- 2.12	15919.9 +- 2596.62	11	57.05 +- 3.17	12209.96 +- 1173.94	0
CE cal cs	63.04 +- 1.67	**3785.63 +- 169.28**	31	62.76 +- 2.06	**15885.0 +- 2601.4**	24	55.02 +- 3.06	**12114.77 +- 1197.45**	47
Brier base	64.55 +- 2.04	6334.63 +- 1300.82	0	64.55 +- 2.04	17705.8 +- 2642.18	6	64.55 +- 2.04	15539.01 +- 1182.99	0
Brier cs	63.35 +- 2.29	4210.05 +- 287.49	15	63.37 +- 2.39	16674.45 +- 2701.25	12	59.42 +- 3.1	13259.2 +- 1170.8	0
Brier cal cs	61.38 +- 2.37	3915.2 +- 218.8	**36**	61.6 +- 2.9	16454.86 +- 2747.58	**35**	53.51 +- 4.4	12606.53 +- 1280.89	**53**

Later, in section 6.2.5, the best results from different techniques are compared to each other. From this section calibrated cost-sensitive cross-entropy is chosen as the best model for each dataset. Additionally, ordinary cross-entropy is taken from this section to see how beneficial different techniques are in minimizing the total cost.

6.2.2 Undersampling comparisons with and without cost-sensitive modifications

Tables 22, 23 and 24 show the undersampling results for each dataset. All tables cover two different weights algorithms usage with two loss functions in a different dataset. Two versions of cost-sensitive and calibrated cost-sensitive models are also made from each undersampled model.

The tables display that the label-wise weights algorithm is better in type 1, slightly better in type 3 and worse in type 2, compared to the prediction-wise weights algorithm. At the same time, their difference volume depends on the dataset. For example in type 1, the label-wise weights version is clearly better in CIFAR-10, but in the modified CIFAR-10 both of the weights algorithms can be considered.

Note that the cost-sensitive modifications play a huge part in deciding the better weights algorithm. For example, in type 1 cost matrices, the basic undersampling models have smaller average costs using the prediction-wise weights algorithm. After adding the cost-sensitive output modifications, suddenly the label-wise algorithm becomes more useful as the cost-sensitive aspect makes its models so much better. At the same time,

Table 22. MNIST undersampling results

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base lw	99.32 +- 0.12	135.5 +- 61.32	4	98.98 +- 0.18	367.6 +- 76.02	2	99.3 +- 0.07	309.35 +- 45.05	0
CE cs lw	99.22 +- 0.13	95.6 +- 27.29	2	98.68 +- 0.28	399.65 +- 79.71	0	99.06 +- 0.17	265.1 +- 31.01	2
CE rev cs lw	99.26 +- 0.12	99.9 +- 33.74	1	98.98 +- 0.18	367.6 +- 76.02	2	99.08 +- 0.16	264.4 +- 30.42	2
CE cal cs lw	99.23 +- 0.13	97.1 +- 27.14	2	98.69 +- 0.28	397.65 +- 78.81	0	99.09 +- 0.15	264.75 +- 31.11	1
CE rev cal cs lw	99.27 +- 0.12	101.75 +- 35.69	3	99.0 +- 0.16	364.95 +- 74.63	1	99.1 +- 0.15	264.6 +- 31.24	0
Brier base lw	99.35 +- 0.1	106.75 +- 31.78	2	98.97 +- 0.25	374.3 +- 83.49	0	99.31 +- 0.09	312.65 +- 42.64	0
Brier cs lw	99.19 +- 0.12	89.85 +- 12.67	2	98.58 +- 0.39	413.45 +- 91.81	0	99.04 +- 0.2	264.25 +- 37.58	1
Brier rev cs lw	99.25 +- 0.1	**89.35 +- 13.86**	1	98.97 +- 0.25	374.3 +- 83.49	0	99.07 +- 0.19	**259.45 +- 39.0**	3
Brier cal cs	99.28 +- 0.1	89.45 +- 16.7	2	98.7 +- 0.34	395.7 +- 86.66	0	99.17 +- 0.13	265.0 +- 37.56	2
Brier rev cal cs lw	99.32 +- 0.09	90.75 +- 19.33	**4**	99.01 +- 0.23	376.4 +- 83.76	1	99.18 +- 0.12	265.7 +- 38.93	1
CE base pw	99.19 +- 0.15	96.25 +- 17.76	2	99.34 +- 0.08	306.0 +- 47.53	1	99.22 +- 0.11	336.5 +- 63.77	0
CE cs pw	98.78 +- 0.35	124.4 +- 33.42	0	99.27 +- 0.13	287.65 +- 50.28	3	98.94 +- 0.25	293.35 +- 41.79	0
CE rev cs pw	99.17 +- 0.16	95.1 +- 17.58	1	99.28 +- 0.12	286.6 +- 46.4	1	98.97 +- 0.24	287.8 +- 41.92	1
CE cal cs pw	98.78 +- 0.34	124.25 +- 32.16	0	99.27 +- 0.13	287.7 +- 48.43	2	98.94 +- 0.26	293.45 +- 41.96	0
CE rev cal cs pw	99.19 +- 0.15	95.7 +- 18.2	0	99.29 +- 0.12	**285.55 +- 47.01**	2	98.98 +- 0.24	286.7 +- 40.25	0
Brier base pw	99.19 +- 0.17	100.4 +- 20.38	1	99.33 +- 0.12	310.55 +- 69.66	3	99.24 +- 0.16	331.8 +- 86.9	0
Brier cs pw	98.51 +- 0.54	152.6 +- 52.1	1	99.24 +- 0.16	**285.55 +- 60.09**	3	98.83 +- 0.31	295.5 +- 67.45	0
Brier rev cs pw	99.13 +- 0.2	101.55 +- 18.86	1	99.25 +- 0.15	288.45 +- 61.38	**4**	98.9 +- 0.31	285.4 +- 64.49	**4**
Brier cal cs pw	98.86 +- 0.36	119.4 +- 33.38	0	99.28 +- 0.14	289.25 +- 61.19	0	99.01 +- 0.26	290.55 +- 64.9	2
Brier rev cal cs pw	99.24 +- 0.14	101.05 +- 22.22	2	99.29 +- 0.13	292.35 +- 61.6	1	99.06 +- 0.25	287.6 +- 68.97	2

the cost-sensitive modifications do not bring any benefit to the prediction-wise weights algorithm.

Table 23. CIFAR-10 undersampling results

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base lw	88.05 +- 0.76	1890.3 +- 335.25	0	81.7 +- 2.2	7995.8 +- 1226.24	0	86.85 +- 1.08	5779.05 +- 648.54	0
CE cs lw	86.99 +- 1.14	1494.75 +- 111.33	1	79.59 +- 3.47	7838.0 +- 1243.82	0	83.98 +- 1.86	4909.25 +- 606.06	0
CE rev cs lw	87.35 +- 1.02	1536.25 +- 152.94	4	81.7 +- 2.2	7995.8 +- 1226.24	0	84.1 +- 1.87	4925.7 +- 606.89	0
CE cal cs lw	86.55 +- 1.17	**1471.7 +- 96.97**	**6**	79.03 +- 3.61	7845.55 +- 1233.58	0	83.06 +- 1.79	**4823.1 +- 624.25**	4
CE rev cal cs lw	86.97 +- 1.05	1487.6 +- 113.4	1	81.45 +- 2.24	7909.65 +- 1234.62	0	83.15 +- 1.83	4838.4 +- 635.95	**5**
Brier base lw	86.84 +- 1.14	2013.0 +- 338.09	0	78.46 +- 3.65	7712.5 +- 1268.91	0	86.44 +- 1.12	5855.6 +- 731.18	0
Brier cs lw	85.81 +- 1.28	1614.6 +- 129.48	0	76.66 +- 4.03	7841.85 +- 1272.73	0	83.89 +- 1.48	5064.55 +- 720.03	0
Brier rev cs lw	86.17 +- 1.24	1662.8 +- 157.93	1	78.46 +- 3.65	7712.5 +- 1268.91	0	84.05 +- 1.49	5057.15 +- 736.35	0
Brier cal cs	85.34 +- 1.32	1592.2 +- 118.52	1	75.4 +- 4.61	8041.05 +- 1292.76	0	82.8 +- 1.61	4988.95 +- 711.03	0
Brier rev cal cs lw	85.79 +- 1.26	1611.65 +- 124.84	0	77.81 +- 3.95	7738.05 +- 1263.88	1	83.0 +- 1.58	4974.95 +- 725.98	4
CE base pw	86.62 +- 1.13	1605.35 +- 196.21	1	88.1 +- 0.89	5861.7 +- 1110.5	2	85.77 +- 1.75	6085.2 +- 835.54	0
CE cs pw	84.9 +- 1.51	1559.85 +- 140.14	2	87.36 +- 1.01	5518.4 +- 1109.61	1	82.47 +- 2.8	5242.2 +- 648.22	0
CE rev cs pw	86.45 +- 1.18	1568.6 +- 164.98	2	87.47 +- 0.99	5528.35 +- 1121.73	3	82.78 +- 2.66	5209.0 +- 647.84	0
CE cal cs pw	84.23 +- 1.47	1603.7 +- 141.16	1	87.03 +- 0.99	**5504.35 +- 1113.92**	**4**	81.47 +- 3.14	5180.85 +- 645.19	0
CE rev cal cs pw	86.04 +- 1.2	1537.85 +- 141.21	0	87.17 +- 0.96	5510.75 +- 1123.58	2	81.81 +- 2.93	5137.6 +- 640.31	2
Brier base pw	80.74 +- 3.45	1962.75 +- 285.46	0	87.76 +- 1.14	6065.9 +- 1258.94	0	85.77 +- 2.04	6154.7 +- 729.62	0
Brier cs pw	80.38 +- 2.87	1970.75 +- 272.05	0	87.09 +- 1.29	5751.5 +- 1302.56	0	83.12 +- 3.0	5309.15 +- 553.18	0
Brier rev cs pw	80.71 +- 3.4	1960.65 +- 288.9	0	87.19 +- 1.27	5763.85 +- 1296.72	1	83.32 +- 2.9	5318.95 +- 539.55	0
Brier cal cs pw	80.23 +- 2.67	1982.45 +- 257.88	0	86.84 +- 1.37	5724.55 +- 1306.15	4	81.93 +- 3.62	5213.3 +- 564.91	3
Brier rev cal cs pw	80.63 +- 3.27	1959.45 +- 289.71	0	86.95 +- 1.35	5734.75 +- 1306.11	2	82.25 +- 3.43	5199.0 +- 548.31	2

A similar situation happens in type 2 cost matrices, where the prediction-wise weights algorithm gives better results with basic undersampling. This time, after adding cost-sensitive modifications the prediction-wise weights algorithm become even better. At the same time, cost-sensitive modifications do not improve the label-wise algorithm results quality. Therefore, in the end, the prediction-wise weights algorithm ends up being noticeably better.

Table 24. Modified CIFAR-10 undersampling results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base lw	66.11 +- 0.93	4954.7 +- 499.49	0	58.51 +- 2.63	17481.7 +- 2633.42	1	64.86 +- 1.47	14966.2 +- 890.93	0
CE cs lw	63.8 +- 1.19	3739.05 +- 106.26	0	54.41 +- 3.64	17640.55 +- 2729.3	2	56.19 +- 3.18	12383.9 +- 1093.17	0
CE rev cs lw	64.36 +- 1.15	3779.85 +- 122.98	2	58.51 +- 2.63	17481.7 +- 2633.42	1	56.56 +- 3.13	12347.2 +- 1108.78	0
CE cal cs lw	63.31 +- 1.13	**3720.45 +- 105.12**	3	52.91 +- 3.83	17888.85 +- 2753.95	0	53.76 +- 3.3	12289.45 +- 1127.43	3
CE rev cal cs lw	63.82 +- 1.1	3732.9 +- 103.46	4	57.68 +- 2.6	17424.2 +- 2627.42	0	54.35 +- 3.0	**12230.3 +- 1125.67**	**7**
Brier base lw	64.91 +- 1.31	5153.75 +- 566.62	0	53.35 +- 2.22	18428.25 +- 2624.14	0	64.12 +- 1.68	15465.4 +- 1203.58	0
Brier cs lw	63.61 +- 1.71	3969.45 +- 147.08	0	51.48 +- 2.85	18591.05 +- 2647.82	0	59.02 +- 2.44	13366.05 +- 1268.83	0
Brier rev cs lw	64.02 +- 1.63	4114.8 +- 177.92	0	53.35 +- 2.22	18428.25 +- 2624.14	0	59.32 +- 2.36	13344.2 +- 1250.63	0
Brier cal cs	61.97 +- 1.71	3843.0 +- 157.22	3	48.2 +- 3.92	19258.85 +- 2707.74	0	52.99 +- 3.34	12768.4 +- 1326.48	1
Brier rev cal cs lw	62.4 +- 1.72	3842.75 +- 143.43	0	51.18 +- 2.98	18631.15 +- 2650.38	1	53.44 +- 3.23	12708.25 +- 1299.26	2
CE base pw	63.72 +- 1.23	3836.45 +- 144.78	1	66.64 +- 1.29	16443.4 +- 2601.73	0	63.51 +- 2.37	15548.35 +- 904.86	0
CE cs pw	62.17 +- 1.45	3791.95 +- 144.14	3	64.32 +- 2.07	15439.25 +- 2592.85	0	55.04 +- 4.45	12839.55 +- 813.11	1
CE rev cs pw	63.31 +- 1.32	3784.25 +- 126.9	2	64.7 +- 1.89	15446.4 +- 2595.49	3	55.64 +- 4.22	12820.65 +- 844.9	0
CE cal cs pw	61.95 +- 1.48	3809.3 +- 148.01	2	63.56 +- 2.16	**15398.0 +- 2588.39**	**8**	52.15 +- 5.28	12753.65 +- 831.29	1
CE rev cal cs pw	62.89 +- 1.28	3761.55 +- 122.29	**6**	63.9 +- 1.96	15416.85 +- 2599.88	1	53.09 +- 4.68	12648.75 +- 860.8	3
Brier base pw	60.45 +- 2.04	3955.35 +- 203.73	0	65.53 +- 1.49	16864.45 +- 2719.98	0	61.88 +- 3.33	16056.75 +- 1048.81	0
Brier cs pw	60.45 +- 2.04	3955.35 +- 203.73	0	64.1 +- 1.97	16040.65 +- 2689.39	1	56.2 +- 5.12	13988.8 +- 809.52	0
Brier rev cs pw	60.45 +- 2.04	3955.35 +- 203.73	0	64.33 +- 1.89	16074.5 +- 2702.43	1	56.75 +- 5.05	13964.7 +- 784.52	0
Brier cal cs pw	60.45 +- 2.04	3955.35 +- 203.73	0	62.2 +- 2.47	15935.4 +- 2662.24	1	49.59 +- 7.09	13428.15 +- 927.44	0
Brier rev cal cs lw	60.45 +- 2.04	3955.35 +- 203.73	0	62.49 +- 2.35	15936.05 +- 2661.16	1	50.36 +- 6.97	13349.95 +- 924.74	2

Type 3 is different from previous types. Its cost-sensitive modifications improve all the basic undersampled models. Due to that, the final difference between two class weights algorithms is not as clear as with previous types.

These results show that cost-sensitive output modifications help only if the cost matrix's costs direction is different from the weights algorithm direction. For example with label-wise weights algorithm, the cost-sensitive aspect works in type 1, where the misclassification costs are similar for each prediction. At the same time with prediction-wise weights algorithm, the cost-sensitive aspect improves the results in type 2, where the misclassification costs are the same label-wise. On top of that, cost-sensitive output modifications improve the results for both weights algorithm in type 3, where the cost values are chosen randomly and therefore they contain no prediction or prediction-wise pattern.

If we look at the impact of two different cost-sensitivity versions, we cannot see a clear difference between their results. The second version (rev cs), that already considers that the class weights are used for sampling, unquestionably has better accuracies than the first one (cs). However, its costs are sometimes better, but sometimes worse. It shows that in some cases it is beneficial to consider the class weights multiple times.

If we start comparing the calibration and softmax probabilities difference, then we can see that their results are quite equal to each other. The advatanage of calibration seems to increase a little as the dataset gets more complicated, but with a simple dataset, there is no difference at all.

If we compare the loss functions, then we can perceive that Brier score tends to have tinier average costs and does more frequently better predictions than cross-entropy in MNIST. On the other hand, cross-entropy has an advantage in other, more complicated datasets

Later, in section 6.2.5, MNIST undersampling is represented by Brier rev cs lw and Brier cs pw. The representers of CIFAR-10 dataset are CE cal cs lw and CE cal cs pw. The winning methods in modified CIFAR-10 are CE rev cal cs lw and CE cal cs pw. The winning models for all the datasets are chosen so that they cover the best predictions in all 3 cost types.

6.2.3 Oversampling comparisons with and without cost-sensitive modifications

Tables 25, 26 and 27 show the oversampling results for each dataset similarly to the previous section. The results show that oversampling does not work exactly the same way as undersampling even though both of them are resampling approaches with the same class weights.

The similar thing in type 1 is that both of the resampling methods get better base predictions with prediction-wise weights. Also, the cost-sensitive modifications impact is only positively visible with label-wise weights. But the difference between the resampling methods is that with oversampling the cost-sensitivity improvements are not that powerful as with undersampling. It means that the prediction-wise weights algorithm generally remained better in type 1 with oversampling, while with undersampling label-wise weights turned to be better. The only exception with oversampling is modified CIFAR-10, where the label-wise weights algorithm also turned to be better in type 1.

Table 25. MNIST oversampling results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base lw	99.31 +- 0.1	119.9 +- 47.73	0	99.39 +- 0.13	236.35 +- 57.86	**4**	99.27 +- 0.15	321.7 +- 56.1	0
CE cs lw	99.19 +- 0.13	95.1 +- 21.27	0	99.24 +- 0.21	240.5 +- 50.79	2	99.05 +- 0.28	276.55 +- 51.15	2
CE rev cs lw	99.25 +- 0.11	92.6 +- 21.65	1	99.39 +- 0.13	236.35 +- 57.86	4	99.05 +- 0.27	279.9 +- 54.31	2
CE cal cs lw	99.21 +- 0.12	93.9 +- 21.02	0	99.24 +- 0.2	240.05 +- 51.65	3	99.07 +- 0.26	277.15 +- 52.49	3
CE rev cal cs lw	99.25 +- 0.11	93.75 +- 24.5	1	99.39 +- 0.12	236.8 +- 56.63	3	99.07 +- 0.24	279.25 +- 53.63	1
Brier base lw	99.34 +- 0.13	97.95 +- 33.24	1	99.4 +- 0.13	**226.45 +- 50.11**	2	99.28 +- 0.13	315.25 +- 56.1	0
Brier cs lw	99.06 +- 0.28	100.2 +- 28.12	1	99.29 +- 0.22	229.55 +- 54.73	2	98.97 +- 0.23	264.25 +- 50.95	**4**
Brier rev cs lw	99.19 +- 0.22	91.45 +- 23.37	0	99.4 +- 0.13	**226.45 +- 50.11**	2	98.97 +- 0.22	268.0 +- 50.86	1
Brier cal cs	99.23 +- 0.2	89.9 +- 21.9	0	99.33 +- 0.19	226.65 +- 54.55	2	99.11 +- 0.18	270.15 +- 49.25	1
Brier rev cal cs lw	99.29 +- 0.16	88.75 +- 21.95	0	99.42 +- 0.12	230.0 +- 50.43	2	99.11 +- 0.19	270.15 +- 49.16	1
CE base pw	99.45 +- 0.09	79.95 +- 17.37	0	99.3 +- 0.09	326.15 +- 66.01	0	99.31 +- 0.11	311.5 +- 51.97	0
CE cs pw	99.31 +- 0.18	**76.15 +- 20.74**	**6**	99.24 +- 0.11	308.1 +- 54.24	0	99.09 +- 0.18	269.1 +- 49.39	2
CE rev cs pw	99.44 +- 0.1	77.1 +- 17.85	2	99.25 +- 0.1	306.55 +- 54.08	0	99.11 +- 0.17	268.05 +- 43.01	0
CE cal cs pw	99.3 +- 0.18	76.6 +- 20.95	3	99.24 +- 0.11	309.3 +- 55.04	0	99.1 +- 0.17	270.5 +- 48.92	1
CE rev cal cs pw	99.44 +- 0.09	77.65 +- 18.32	1	99.25 +- 0.1	306.15 +- 53.81	0	99.12 +- 0.16	268.5 +- 44.65	1
Brier base pw	99.44 +- 0.08	73.2 +- 14.59	4	99.31 +- 0.11	313.3 +- 66.44	0	99.34 +- 0.11	292.9 +- 51.78	0
Brier cs pw	99.2 +- 0.18	85.45 +- 16.36	0	99.19 +- 0.21	293.4 +- 66.04	1	99.03 +- 0.2	253.0 +- 43.2	2
Brier rev cs pw	99.42 +- 0.09	70.45 +- 13.53	3	99.21 +- 0.19	293.1 +- 64.34	2	99.06 +- 0.2	253.3 +- 44.13	0
Brier cal cs pw	99.34 +- 0.12	75.05 +- 12.68	2	99.24 +- 0.17	291.5 +- 62.15	2	99.17 +- 0.16	**250.75 +- 44.82**	1
Brier rev cal cs pw	99.45 +- 0.07	81.1 +- 15.1	1	99.26 +- 0.16	291.6 +- 61.52	1	99.19 +- 0.16	250.95 +- 44.16	2

Unlike with undersampling, the oversampling basic model makes better predictions using label-wise weights algorithm in type 2. Once again, the cost-sensitive modifications did not improve the results of the label-wise weights. At the same time, the improvements are clear with prediction-wise weights. After the modifications, label-wise weights still

give better results in MNIST and CIFAR-10, while both weighting algorithms give quite similar results in modified CIFAR-10.

Table 26. CIFAR-10 oversampling results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base lw	88.24 +- 0.78	1837.5 +- 313.3	0	88.65 +- 1.21	5028.7 +- 863.23	0	87.59 +- 0.64	5337.1 +- 463.82	0
CE cs lw	87.19 +- 1.17	1451.3 +- 119.42	1	87.57 +- 1.66	4950.7 +- 946.56	1	85.03 +- 1.22	4591.7 +- 415.4	0
CE rev cs lw	87.54 +- 1.04	1487.0 +- 141.34	0	88.65 +- 1.21	5028.7 +- 863.23	0	85.17 +- 1.18	4589.1 +- 412.61	0
CE cal cs lw	86.64 +- 1.22	1432.8 +- 114.66	0	86.92 +- 1.87	4989.8 +- 981.54	**5**	83.93 +- 1.3	4524.25 +- 425.73	1
CE rev cal cs lw	87.06 +- 1.1	1437.45 +- 114.37	0	88.33 +- 1.34	4967.0 +- 897.6	2	84.11 +- 1.28	4507.3 +- 427.48	2
Brier base lw	88.3 +- 0.86	1905.85 +- 488.67	0	88.37 +- 0.79	4912.7 +- 863.35	2	87.19 +- 1.26	5549.45 +- 624.14	0
Brier cs lw	87.49 +- 1.17	1467.7 +- 191.04	0	87.21 +- 1.27	**4864.45 +- 824.25**	4	84.83 +- 1.63	4801.9 +- 654.52	0
Brier rev cs lw	87.8 +- 1.07	1525.3 +- 237.85	0	88.37 +- 0.79	4912.7 +- 863.35	2	85.01 +- 1.56	4785.65 +- 641.86	0
Brier cal cs	87.07 +- 1.27	1442.1 +- 168.13	1	86.89 +- 1.4	4888.55 +- 823.77	1	83.92 +- 1.92	4720.8 +- 670.4	0
Brier rev cal cs lw	87.5 +- 1.18	1468.9 +- 195.29	0	88.24 +- 0.85	4901.65 +- 854.37	0	84.22 +- 1.78	4691.45 +- 668.68	2
CE base pw	88.91 +- 1.46	1436.45 +- 217.45	0	87.48 +- 0.76	6277.0 +- 1108.5	0	87.97 +- 1.12	5220.8 +- 715.01	0
CE cs pw	87.54 +- 1.99	1322.15 +- 185.08	3	87.07 +- 0.76	5809.1 +- 1142.04	0	85.51 +- 1.67	4518.95 +- 540.59	0
CE rev cs pw	88.82 +- 1.53	1390.25 +- 194.41	1	87.14 +- 0.77	5843.5 +- 1145.62	0	85.71 +- 1.57	4528.65 +- 539.73	0
CE cal cs pw	86.57 +- 2.08	1378.35 +- 192.79	1	86.76 +- 0.75	5759.05 +- 1143.16	1	84.38 +- 1.68	4439.35 +- 544.14	**6**
CE rev cal cs pw	88.19 +- 1.61	1314.95 +- 173.5	4	86.87 +- 0.73	5776.85 +- 1153.67	2	84.62 +- 1.61	**4436.75 +- 542.29**	3
Brier base pw	88.83 +- 1.08	1356.6 +- 156.73	1	86.98 +- 1.78	6346.05 +- 1308.53	0	87.52 +- 1.27	5344.85 +- 743.12	0
Brier cs pw	87.6 +- 1.21	**1307.05 +- 117.32**	3	86.21 +- 1.91	6030.2 +- 1232.84	2	85.19 +- 1.47	4687.6 +- 661.33	0
Brier rev cs pw	88.74 +- 1.1	1334.15 +- 146.03	3	86.35 +- 1.9	6029.3 +- 1231.76	0	85.38 +- 1.45	4690.15 +- 662.02	0
Brier cal cs pw	87.08 +- 1.21	1339.4 +- 115.67	0	85.89 +- 2.07	6022.35 +- 1222.0	0	84.37 +- 1.61	4631.35 +- 654.75	3
Brier rev cal cs pw	88.6 +- 1.13	1314.6 +- 133.72	**5**	86.03 +- 2.02	6024.3 +- 1223.81	0	84.62 +- 1.62	4613.85 +- 652.71	3

In the third matrix type, the results between over and undersampling are the most similar. Both of the approaches reduce the cost with cost-sensitive modifications with both weights algorithms. The difference is that undersampling gets somewhat better base results with label-wise weights while oversampling gets better base results with prediction-wise weights. After cost-sensitive changes, the weights algorithms association remains the same, but the difference between them is not enormous.

Table 27. Modified CIFAR-10 oversampling results.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base lw	65.91 +- 0.96	4932.4 +- 640.16	0	62.65 +- 1.89	15917.9 +- 2286.99	1	65.78 +- 0.92	14820.15 +- 880.94	0
CE cs lw	63.59 +- 1.23	3737.55 +- 115.92	5	58.81 +- 2.34	16043.85 +- 2345.31	0	58.19 +- 2.11	12207.95 +- 1138.86	0
CE rev cs lw	64.12 +- 1.2	3776.85 +- 136.07	2	62.65 +- 1.89	15917.9 +- 2286.99	1	58.5 +- 1.98	12178.9 +- 1175.03	0
CE cal cs lw	63.0 +- 1.21	3739.1 +- 115.32	0	57.27 +- 2.47	16246.0 +- 2387.39	0	55.52 +- 2.42	12020.5 +- 1191.43	3
CE rev cal cs lw	63.44 +- 1.12	3734.1 +- 106.2	1	61.63 +- 1.81	15866.6 +- 2301.16	**4**	55.9 +- 2.25	12007.95 +- 1193.08	3
Brier base lw	66.35 +- 1.21	4925.4 +- 520.43	0	60.68 +- 1.73	15757.95 +- 2494.68	2	65.26 +- 1.08	14898.05 +- 1017.44	0
Brier cs lw	64.91 +- 1.33	3838.65 +- 194.36	0	59.27 +- 2.07	15745.15 +- 2493.45	0	60.59 +- 2.03	12962.15 +- 1138.95	0
Brier rev cs lw	65.38 +- 1.26	3993.95 +- 246.07	0	60.68 +- 1.73	15757.95 +- 2494.68	2	60.88 +- 1.97	12964.25 +- 1153.28	0
Brier cal cs	63.15 +- 1.7	3717.05 +- 179.91	3	57.43 +- 2.59	15981.45 +- 2496.14	0	54.97 +- 3.39	12346.8 +- 1221.83	0
Brier rev cal cs lw	63.65 +- 1.56	**3704.65 +- 174.71**	**7**	59.45 +- 1.93	**15734.2 +- 2499.83**	2	55.35 +- 3.18	12315.6 +- 1211.89	3
CE base pw	62.71 +- 1.73	3948.5 +- 177.86	0	65.93 +- 1.35	17097.2 +- 2501.47	0	66.14 +- 1.11	14308.5 +- 787.4	0
CE cs pw	61.13 +- 1.75	3893.9 +- 175.09	1	64.19 +- 1.7	15789.75 +- 2628.77	1	58.73 +- 3.09	11991.95 +- 928.05	0
CE rev cs pw	62.35 +- 1.77	3876.5 +- 150.52	1	64.5 +- 1.62	15830.3 +- 2636.72	1	59.27 +- 3.0	11965.85 +- 945.74	0
CE cal cs pw	60.83 +- 1.77	3917.45 +- 177.32	1	63.35 +- 1.71	15735.0 +- 2658.28	**4**	55.67 +- 3.49	11871.65 +- 941.52	2
CE rev cal cs pw	61.63 +- 1.57	3862.25 +- 155.71	1	63.7 +- 1.62	15757.45 +- 2651.51	1	56.4 +- 3.16	**11802.55 +- 957.82**	**5**
Brier base pw	62.18 +- 1.67	3781.9 +- 167.1	2	65.74 +- 1.73	16989.8 +- 2343.08	0	65.26 +- 1.33	14765.45 +- 1166.12	0
Brier cs pw	62.18 +- 1.67	3781.9 +- 167.1	2	64.69 +- 1.94	16119.65 +- 2416.73	0	60.69 +- 2.35	12864.7 +- 1129.28	0
Brier rev cs pw	62.18 +- 1.67	3781.9 +- 167.1	2	64.83 +- 1.91	16170.7 +- 2406.45	0	61.04 +- 2.26	12932.5 +- 1133.83	0
Brier cal cs pw	62.18 +- 1.67	3781.9 +- 167.1	2	63.02 +- 2.35	15898.9 +- 2423.87	3	54.86 +- 3.83	12305.1 +- 1255.51	2
Brier rev cal cs pw	62.18 +- 1.67	3781.9 +- 167.1	2	63.26 +- 2.25	15916.3 +- 2434.93	1	55.56 +- 3.57	12269.4 +- 1239.2	2

Similarly to undersampling, none of the cost-sensitive modification algorithms can

be prefered. Also, calibration mostly did not give many benefits over the uncalibrated model, but if it did, then the difference is more noticeable in datasets where the accuracy is low. If we compare the loss functions, then it is hard to choose the better one as both of them had their highlights with small differences.

Later, in section 6.2.5, oversampling is represented by Brier cs lw that performs generally well in type 2 and 3, and CE cs lw that is the best model in type 1. The representers of CIFAR-10 dataset are Brier cs lw and CE rev cal cs lw. The chosen models in modified CIFAR-10 dataset are Brier rev cal cs lw and CE rev cal cs lw. Those models from both CIFAR-10 dataset versions can cover the best models in each type or are very close to it.

6.2.4 Weighted loss comparisons with and without cost-sensitive modifications

Tables 28, 29 and 30 show the weighted loss function results for each dataset. Based on the table we can say that the results have some resemblances to oversampling and some to undersampling.

Table 28. MNIST weighted loss

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base lw	99.27 +- 0.12	109.9 +- 27.06	1	99.2 +- 0.18	331.35 +- 82.06	3	99.26 +- 0.2	329.65 +- 95.34	0
CE cs lw	99.11 +- 0.21	99.8 +- 19.53	1	99.04 +- 0.24	332.1 +- 78.92	1	99.03 +- 0.24	282.6 +- 71.58	0
CE rev cs lw	99.18 +- 0.17	**98.15 +- 18.91**	1	99.2 +- 0.18	331.35 +- 82.06	3	99.05 +- 0.23	283.25 +- 71.18	1
CE cal cs lw	99.13 +- 0.21	99.15 +- 21.2	**3**	99.06 +- 0.22	327.8 +- 77.48	2	99.06 +- 0.23	281.9 +- 74.24	**4**
CE rev cal cs lw	99.19 +- 0.16	98.95 +- 18.76	**3**	99.21 +- 0.17	333.35 +- 81.19	1	99.08 +- 0.21	284.05 +- 73.79	3
Brier base lw	99.23 +- 0.18	115.15 +- 23.19	1	99.12 +- 0.25	358.95 +- 98.8	1	99.29 +- 0.13	316.9 +- 55.65	0
Brier cs lw	98.95 +- 0.38	111.55 +- 35.15	2	98.82 +- 0.46	373.85 +- 128.69	2	98.98 +- 0.19	272.85 +- 50.33	1
Brier rev cs lw	99.1 +- 0.29	99.0 +- 26.31	2	99.12 +- 0.25	358.95 +- 98.8	1	98.99 +- 0.2	272.8 +- 56.72	3
Brier cal cs	99.12 +- 0.28	99.7 +- 25.57	2	98.95 +- 0.38	361.95 +- 121.16	3	99.13 +- 0.16	**268.35 +- 50.78**	1
Brier rev cal cs lw	99.18 +- 0.24	99.7 +- 22.45	**3**	99.14 +- 0.23	371.35 +- 95.17	0	99.12 +- 0.17	273.85 +- 56.19	0
CE base pw	99.12 +- 0.17	105.75 +- 23.55	1	99.25 +- 0.1	328.45 +- 58.68	1	99.23 +- 0.12	347.35 +- 68.94	0
CE cs pw	98.56 +- 0.51	146.75 +- 49.7	0	99.15 +- 0.16	320.35 +- 58.83	2	99.02 +- 0.21	285.65 +- 53.49	2
CE rev cs pw	99.09 +- 0.19	102.95 +- 21.88	1	99.17 +- 0.15	318.95 +- 58.09	3	99.04 +- 0.2	288.0 +- 50.02	0
CE cal cs pw	98.69 +- 0.37	133.95 +- 36.22	1	99.17 +- 0.14	**317.85 +- 57.5**	**5**	99.05 +- 0.18	285.9 +- 53.01	2
CE rev cal cs pw	99.12 +- 0.17	104.55 +- 22.43	1	99.18 +- 0.14	318.7 +- 56.01	2	99.08 +- 0.16	287.7 +- 49.21	0
Brier base pw	99.14 +- 0.18	109.75 +- 22.53	0	99.21 +- 0.14	382.6 +- 95.22	1	99.26 +- 0.11	326.95 +- 73.26	0
Brier cs pw	98.21 +- 0.79	181.9 +- 78.47	0	99.12 +- 0.2	348.55 +- 80.55	1	98.87 +- 0.23	284.9 +- 40.97	0
Brier rev cs pw	99.09 +- 0.21	109.25 +- 22.59	1	99.14 +- 0.2	347.7 +- 78.87	1	98.92 +- 0.22	282.7 +- 43.15	3
Brier cal cs pw	98.75 +- 0.41	130.35 +- 40.1	0	99.16 +- 0.18	353.15 +- 77.95	0	99.04 +- 0.17	281.6 +- 43.03	2
Brier rev cal cs pw	99.19 +- 0.14	115.05 +- 26.17	2	99.18 +- 0.16	352.45 +- 75.94	0	99.06 +- 0.17	281.8 +- 45.81	0

Similarly to both resampling methods, label-wise class weights get better base results in type 1. Also, the cost-sensitive modifications impact is positively visible with prediction-wise weights. With label-wise weights, a tiny benefit can be sometimes seen. The strength of cost-sensitive modifications is more similar to undersampling as the label-wise weights generally end up with better outcomes. Modified CIFAR-10 is the only dataset in this case, where it is hard to choose a better class weights algorithm.

In type 2 different weights algorithms base results were better for oversampling and undersampling. In that sense, weighted loss is a mixture of them since cross-entropy

Table 29. CIFAR-10 weighted loss.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base lw	86.8 +- 0.9	2020.05 +- 429.14	0	86.09 +- 1.25	6109.35 +- 1083.55	2	86.77 +- 1.37	5753.35 +- 600.7	0
CE cs lw	85.75 +- 1.23	1587.05 +- 143.48	1	84.51 +- 1.93	**6036.95 +- 1085.99**	1	84.17 +- 2.37	4886.7 +- 450.65	0
CE rev cs lw	86.11 +- 1.15	1627.3 +- 178.95	2	86.09 +- 1.25	6109.35 +- 1083.55	2	84.25 +- 2.36	4907.3 +- 457.61	0
CE cal cs lw	85.17 +- 1.22	1579.1 +- 130.73	1	83.94 +- 2.1	6060.1 +- 1087.01	**3**	83.15 +- 2.15	4816.6 +- 469.99	3
CE rev cal cs lw	85.64 +- 1.16	**1574.25 +- 139.49**	2	85.8 +- 1.27	6066.25 +- 1115.56	2	83.23 +- 2.11	4830.1 +- 473.91	1
Brier base lw	86.61 +- 1.93	1905.1 +- 318.01	0	82.91 +- 3.75	6674.4 +- 1359.08	1	86.71 +- 1.25	5681.15 +- 577.48	0
Brier cs lw	85.38 +- 2.26	1599.15 +- 228.04	2	80.72 +- 4.12	6803.6 +- 1432.04	0	84.02 +- 1.68	4957.55 +- 551.27	0
Brier rev cs lw	85.77 +- 2.2	1626.55 +- 233.31	2	82.91 +- 3.75	6674.4 +- 1359.08	1	84.21 +- 1.61	4948.05 +- 543.82	0
Brier cal cs	84.84 +- 2.35	1605.1 +- 225.16	0	79.84 +- 4.63	6920.6 +- 1505.97	0	83.06 +- 1.91	4892.6 +- 559.27	1
Brier rev cal cs lw	85.37 +- 2.29	1599.6 +- 230.08	**3**	82.47 +- 4.02	6686.0 +- 1393.2	1	83.33 +- 1.83	4877.6 +- 533.87	3
CE base pw	85.33 +- 1.52	1757.75 +- 194.81	0	86.85 +- 1.09	6547.4 +- 1109.62	1	87.22 +- 1.14	5534.8 +- 615.82	0
CE cs pw	83.16 +- 1.78	1726.8 +- 157.89	1	86.12 +- 1.49	6103.15 +- 1050.19	0	84.27 +- 1.9	4721.5 +- 537.57	0
CE rev cs pw	85.08 +- 1.59	1717.05 +- 172.2	2	86.22 +- 1.4	6128.7 +- 1061.55	1	84.46 +- 1.84	4735.1 +- 540.5	0
CE cal cs pw	82.46 +- 1.67	1772.2 +- 159.54	1	85.84 +- 1.51	6064.2 +- 1039.86	**3**	83.31 +- 1.92	4671.5 +- 532.25	3
CE rev cal cs pw	84.69 +- 1.55	1677.15 +- 151.17	2	85.97 +- 1.42	6088.3 +- 1050.15	1	83.56 +- 1.78	**4668.95 +- 536.22**	3
Brier base pw	81.61 +- 3.21	1914.25 +- 285.59	1	86.76 +- 1.07	6546.85 +- 1079.81	0	86.9 +- 1.53	5625.2 +- 710.06	0
Brier cs pw	80.39 +- 2.65	1966.9 +- 257.41	0	86.13 +- 1.23	6143.95 +- 1051.21	0	84.34 +- 2.09	4874.1 +- 597.11	0
Brier rev cs pw	80.73 +- 2.83	1937.9 +- 271.76	0	86.76 +- 1.07	6546.85 +- 1079.81	1	84.39 +- 2.09	4903.45 +- 592.4	0
Brier cal cs pw	79.94 +- 2.47	2007.9 +- 242.49	0	85.88 +- 1.31	6099.3 +- 1043.68	2	83.35 +- 2.48	4802.65 +- 574.03	**4**
Brier rev cal cs pw	80.31 +- 2.66	1974.3 +- 258.57	1	86.72 +- 1.07	6417.75 +- 1068.45	1	83.43 +- 2.48	4832.75 +- 591.49	2

gives better base results with label-wise weights and Brier score results are better with prediction-wise weights. After adding the cost-sensitive output modifications, label-wise weights algorithm base results rarely see a slight improvement. Prediction-wise predictions, on the other hand, improve always. After the modifications, prediction-wise predictions can be seen as somewhat better weights algorithm choice, although the performance is quite equal for both of them.

Table 30. Modified CIFAR-10 weighted loss

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
CE base lw	64.39 +- 1.58	4892.5 +- 599.21	0	61.6 +- 2.61	16134.65 +- 2712.15	1	65.88 +- 1.19	14750.9 +- 899.15	0
CE cs lw	61.95 +- 1.87	3869.65 +- 189.5	1	57.03 +- 4.04	16508.0 +- 2875.09	0	57.38 +- 3.15	12105.65 +- 1002.53	0
CE rev cs lw	62.53 +- 1.79	3879.25 +- 194.16	1	61.6 +- 2.61	16134.65 +- 2712.15	1	57.73 +- 3.06	12080.05 +- 993.77	0
CE cal cs lw	61.56 +- 1.77	3874.3 +- 186.61	1	55.77 +- 4.23	16703.7 +- 2897.27	0	55.15 +- 3.23	12000.65 +- 1008.2	3
CE rev cal cs lw	62.11 +- 1.67	3869.4 +- 188.7	3	61.02 +- 2.47	16116.0 +- 2754.86	**6**	55.6 +- 3.08	**11976.75 +- 1006.67**	1
Brier base lw	63.79 +- 2.25	5138.95 +- 577.04	0	55.48 +- 4.79	17457.35 +- 2770.6	3	63.99 +- 2.32	15381.15 +- 1269.62	0
Brier cs lw	62.06 +- 2.47	4055.75 +- 239.27	1	53.62 +- 5.19	17578.15 +- 2798.37	0	58.62 +- 3.08	13327.7 +- 1445.91	0
Brier rev cs lw	62.59 +- 2.45	4167.0 +- 266.04	0	55.48 +- 4.79	17457.35 +- 2770.6	3	58.97 +- 2.9	13301.9 +- 1422.1	0
Brier cal cs	60.42 +- 2.48	3980.25 +- 244.67	**5**	50.72 +- 6.21	18167.2 +- 2918.07	0	52.64 +- 3.89	12808.85 +- 1569.31	1
Brier rev cal cs lw	60.83 +- 2.51	3969.55 +- 236.57	0	53.48 +- 5.59	17613.6 +- 2833.99	2	53.18 +- 3.74	12744.7 +- 1542.78	3
CE base pw	62.93 +- 1.48	3956.45 +- 240.93	1	65.82 +- 1.39	16745.3 +- 2273.18	0	65.5 +- 1.85	14604.55 +- 1006.32	0
CE cs pw	61.08 +- 1.58	3900.25 +- 155.53	0	63.36 +- 1.94	15797.25 +- 2485.46	0	57.16 +- 3.56	12242.2 +- 1348.13	0
CE rev cs pw	62.47 +- 1.5	3893.15 +- 203.31	4	63.76 +- 1.84	15798.05 +- 2470.62	1	57.76 +- 3.31	12229.15 +- 1361.79	0
CE cal cs pw	60.79 +- 1.61	3922.6 +- 160.65	0	62.48 +- 1.93	15779.7 +- 2499.01	2	54.65 +- 3.54	12140.45 +- 1406.55	1
CE rev cal cs pw	61.94 +- 1.39	**3867.95 +- 176.96**	2	62.88 +- 1.82	**15775.5 +- 2501.08**	1	55.41 +- 3.27	12073.9 +- 1414.26	**6**
Brier base pw	59.84 +- 1.97	4015.95 +- 197.48	1	64.74 +- 1.49	17246.85 +- 2669.72	0	64.27 +- 2.05	15095.5 +- 908.62	0
Brier cs pw	59.84 +- 1.97	4015.95 +- 197.48	1	63.12 +- 1.9	16450.2 +- 2660.08	0	58.89 +- 3.31	13129.8 +- 1073.96	0
Brier rev cs pw	59.84 +- 1.97	4015.95 +- 197.48	1	63.35 +- 1.84	16480.05 +- 2663.12	0	59.32 +- 3.14	13154.45 +- 1026.43	0
Brier cal cs pw	59.84 +- 1.97	4015.95 +- 197.48	1	61.16 +- 2.4	16349.65 +- 2676.82	4	52.84 +- 4.72	12663.45 +- 1187.39	2
Brier rev cal cs pw	59.84 +- 1.97	4015.95 +- 197.48	1	61.46 +- 2.3	16346.6 +- 2669.97	1	53.59 +- 4.43	12580.3 +- 1171.98	3

In the third matrix type, weighted loss works as resampling methods. The cost-sensitive modification improves the results for both weights algorithms. If we compare the weighting algorithms, then label-wise weights give better results in MNIST, prediction-

wise weights in CIFAR-10 and in modified CIFAR-10 they were quite equal. Generally, previous holds for results before and after cost-sensitive modifications, where modified CIFAR-10 cross-entropy is a small exception.

Similarly to resampling methods, calibration gave a small advantage over the uncalibrated models in some cases, but not always. Also, it is not possible to choose, which cost-sensitive version to use, as they seem to be quite equal choices. If we compare the loss functions, then cross-entropy had either a small edge over Brier score or they were equal. Type 3 MNIST dataset is the only place where Brier score can be chosen as a better loss function.

In the next section weighted loss functions are represented by CE cal cs lw and CE cal cs pw from MNIST dataset. The representers of CIFAR-10 and modified CIFAR-10 datasets are CE rev cal cs lw and CE rev cal cs pw. Those models cover the best results in different types or are at least close to the best one.

6.2.5 Comparing the best results from every approach

In this section cost-sensitive baseline, undersampling, oversampling and weighted loss models are compared to each other. First, we use tables 31, 32, 33 to compare the chosen best models from each technique in each used dataset. After, we use table 34 to compare the best average costs received with all the different techniques.

Table 31. MNIST chosen best models.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
Base CE cal cs	99.17 +- 0.14	101.52 +- 19.29	2	99.19 +- 0.13	329.35 +- 59.67	1	99.04 +- 0.16	289.73 +- 43.56	3
Under Brier rev cs lw	99.25 +- 0.1	89.35 +- 13.86	1	98.97 +- 0.25	374.3 +- 83.49	0	99.07 +- 0.19	259.45 +- 39.0	3
Under Brier cs pw	98.51 +- 0.54	152.6 +- 52.1	0	99.24 +- 0.16	285.55 +- 60.09	5	98.83 +- 0.31	295.5 +- 67.45	3
Over Brier cs lw	99.06 +- 0.28	100.2 +- 28.12	4	99.29 +- 0.22	**229.55 +- 54.73**	**11**	98.97 +- 0.23	**264.25 +- 50.95**	3
Over CE cs pw	99.31 +- 0.18	**76.15 +- 20.74**	**11**	99.24 +- 0.11	308.1 +- 54.24	0	99.09 +- 0.18	269.1 +- 49.39	2
Weighted CE cal cs lw	99.13 +- 0.21	99.15 +- 21.2	1	99.06 +- 0.22	327.8 +- 77.48	1	99.06 +- 0.23	281.9 +- 74.24	**4**
Weighted CE cal cs pw	98.69 +- 0.37	133.95 +- 36.22	1	99.17 +- 0.14	317.85 +- 57.5	2	99.05 +- 0.18	285.9 +- 53.01	2

When we try to find the best technique for multiclass cost-sensitive learning, we can notice that oversampling models gave almost always the best average costs. The only exception to this is type 2 undersampling model in modified CIFAR-10 that managed to outperform oversampling in those circumstances. If we also look at the number of times best columns, we can observe that oversampling generally makes the best predictions using type 1 and 2 matrices. However, in type 3, all the techniques seem to be quite equal as all the models make the best predictions almost the same number of times.

Undersampling seems to be a good choice when one does not want to increase the size of the dataset as it mostly seems to have second-best results. However, there are situations, where the cost-sensitive base and weighted loss models work better than undersampling. Hence, the choices can vary a lot based on the cost matrix.

Table 32. CIFAR-10 chosen best models.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
Base CE cal cs	86.37 +- 1.22	1528.49 +- 141.84	3	86.59 +- 1.26	5822.34 +- 1098.24	3	83.97 +- 1.54	4601.63 +- 512.97	**6**
Under CE cal cs lw	86.55 +- 1.17	1471.7 +- 96.97	0	79.03 +- 3.61	7845.55 +- 1233.58	0	83.06 +- 1.79	4823.1 +- 624.25	2
Under CE cal cs pw	84.23 +- 1.47	1603.7 +- 141.16	1	87.03 +- 0.99	5504.35 +- 1113.92	5	81.47 +- 3.14	5180.85 +- 645.19	2
Over Brier cs lw	87.49 +- 1.17	1467.7 +- 191.04	4	87.21 +- 1.27	**4864.45 +- 824.25**	**8**	84.83 +- 1.63	4801.9 +- 654.52	2
Over CE rev cal cs pw	88.19 +- 1.61	**1314.95 +- 173.5**	**11**	86.87 +- 0.73	5776.85 +- 1153.67	3	84.62 +- 1.61	**4436.75 +- 542.29**	5
Weighted CE rev cal cs lw	85.64 +- 1.16	1574.25 +- 139.49	0	85.8 +- 1.27	6066.25 +- 1115.56	0	83.23 +- 2.11	4830.1 +- 473.91	1
Weighted CE rev cal cs pw	84.69 +- 1.55	1677.15 +- 151.17	1	85.97 +- 1.42	6088.3 +- 1050.15	1	83.56 +- 1.78	4668.95 +- 536.22	2

Table 33. Modified CIFAR-10 chosen best models.

Model	Type 1 accuracy	Type 1 cost	Type 1 Times best	Type 2 accuracy	type 2 cost	Type 2 Times best	Type 3 accuracy	type 3 cost	Type 3 Times best
Base CE cal cs	63.04 +- 1.67	3785.63 +- 169.28	0	62.76 +- 2.06	15885.0 +- 2601.4	1	55.02 +- 3.06	12114.77 +- 1197.45	0
Under CE rev cal cs lw	63.82 +- 1.1	3732.9 +- 103.46	4	57.68 +- 2.6	17424.2 +- 2627.42	1	54.35 +- 3.0	12230.3 +- 1125.67	3
Under CE cal cs pw	61.95 +- 1.48	3809.3 +- 148.01	3	63.56 +- 2.16	**15398.0 +- 2588.39**	**5**	52.15 +- 5.28	12753.65 +- 831.29	0
Over Brier rev cal cs lw	63.65 +- 1.56	**3704.65 +- 174.71**	**9**	59.45 +- 1.93	15734.2 +- 2499.83	4	55.35 +- 3.18	12315.6 +- 1211.89	4
Over CE rev cal cs pw	61.63 +- 1.57	3862.25 +- 155.71	1	63.7 +- 1.62	15757.45 +- 2651.51	3	56.4 +- 3.16	**11802.55 +- 957.82**	3
Weighted CE rev cal cs lw	62.11 +- 1.67	3869.4 +- 188.7	3	61.02 +- 2.47	16116.0 +- 2754.86	4	55.6 +- 3.08	11976.75 +- 1006.67	4
Weighted CE rev cal cs pw	61.94 +- 1.39	3867.95 +- 176.96	0	62.88 +- 1.82	15775.5 +- 2501.08	2	55.41 +- 3.27	12073.9 +- 1414.26	**6**

Table 34 displays the smallest average costs received with all the techniques. From the table, we can observe that every single used technique or its combination with cost-sensitive output modifications improved the total cost. This table also confirms that oversampling indubitably performed the best among used techniques.

Table 34. Smallest binary average total costs from every technique.

Technique	**Type 1 cost**	**Type2 cost**	**Type 3 result**	**Dataset**
Ordinary baseline	134.09	350.22	337.12	MNIST
Cost-sensitive baseline	100.78	329.35	288.35	MNIST
Undersampled	89.35	285.55	259.45	MNIST
Oversampled	**76.15**	**226.45**	**250.75**	MNIST
Weighted loss	98.15	317.85	268.35	MNIST
Ordinary baseline	2248.6	6304.26	5512.25	CIFAR-10
Cost-sensitive baseline	1528.49	5822.34	4601.63	CIFAR-10
Undersampled	1471.7	5504.35	4823.1	CIFAR-10
Oversampled	**1307.05**	**4864.45**	**4436.75**	CIFAR-10
Weighted loss	1574.25	6036.95	4668.95	CIFAR-10
Ordinary baseline	6139.47	17154.95	15059.19	mod CIFAR-10
Cost-sensitive baseline	3785.63	15885.0	12114.77	mod CIFAR-10
Undersampled	3720.45	**15398.0**	12230.3	mod CIFAR-10
Oversampled	**3704.65**	15734.2	**11802.55**	mod CIFAR-10
Weighted loss	3867.95	15775.5	11976.75	mod CIFAR-10

7 Conclusion

In this thesis, an experimental study was performed in minimizing the total misclassification cost instead of other metrics while using neural networks in balanced dataset settings. The experiments involved the usage of 3 datasets with different degree of difficulty. The used datasets were MNIST and CIFAR-10. On top of CIFAR-10, an additional self-made dataset with worse accuracies was constructed by covering some pixels and adding noise. Each dataset had a multiclass and a binary version of it.

Additionally, the experiments involved the usage of three different cost matrix generation algorithms resulting in different cost matrix types. Each type contained ten different matrices for multiclass datasets and two matrices for binary datasets. Such experiments settings cover a wide variety of possible circumstances that others may run into.

In this thesis, the used techniques were cost-sensitive neural network output modifications, resampling the datasets and usage of weighted loss functions. Cost-sensitive neural network modifications included the calculation of expected costs for each prediction based on a cost matrix and model output probabilities. In the end, the predictions were modified so that the class with the smallest expected cost was predicted. Additionally, the output probabilities were calibrated using Temperature scaling.

Resampling involved the usage of undersampling and oversampling. A Common thing between resampling methods and weighted loss is that they all need class weights. In this thesis, two different class weights algorithms were proposed. One of them considers the prediction-wise costs and another one considers the label-wise costs.

Resampling and weighted loss techniques were also made cost-sensitive with and without calibration using two different cost-sensitivity versions. The first one makes the network cost-sensitive based on the model output probabilities and the cost matrix. The second one considers the class weights and adjusts the output probabilities accordingly before changing the output to be cost-sensitive.

The training process is kept as similar as possible for all the tests. In this thesis, the used network architecture was ResNet18 with Adam optimizer. Early stopping was used for determining the end time of training. Each model was trained using two loss functions: cross-entropy and Brier score. In the end, the average and standard deviation of accuracies and total costs were reported over multiple runs. In a binary case, three models from each technique were trained per a cost matrix and in a multiclass case, two models were trained per cost matrix.

The results show that all the used techniques affected the total cost positively. The best results for both, binary and multiclass case, were received using oversampling and cost-sensitive modifications combinations. In the binary case, the next best techniques were baseline model with cost-sensitive modification and weighted loss functions. In the multiclass case, on the other hand, undersampling performed quite well.

However, the results also showed that no technique performed the best every time. The models were trained multiple times with different cost matrices and among different

comparisons, it happened rarely that one model did the best predictions more than 50% of the time. The reason for this can be either the randomness in neural network training procedures or the models are prefered over different cost matrices circumstances. Hence, only some general guidelines can be given.

The loss functions are hard to compare. The generalization can be biased but Brier score seemed to be a little better choice when the dataset was really easy or extremely complicated. Otherwise, cross-entropy can be prefered. Also, Brier score seemed to perform better in binary datasets than it did in multiclass cases compared to cross-entropy. Hence, in the multiclass case, Brier score may not be worth a try. This generalisation is done based on only 3 datasets without statistical tests which makes it unreliable.

If we look into cost-sensitive modification in more detail, then they seem to greatly improve the basic neural network results. However, after using resampling techniques or weighted loss they may and may not be useful. In the multiclass case, the class weights algorithms had a great influence on the effect of cost-sensitive modifications. Generally, cost-sensitive modifications did not improve the predictions for prediction-wise class weights when the cost matrices were prediction-wise related but worked well for label-wise related matrices and vice versa.

One thing to note about cost-sensitive modifications in the multiclass case is an observation that quite often it is useful to make the resampling and weighted loss models cost-sensitive based the cost-matrix that was already used for resampling. It came out because there were two different cost-sensitive modification versions where one considered the fact that class weights should already make the output of neural network cost-sensitive and another one did not. Nevertheless, the results showed that the costs were generally similar to each other and both of them had own highlights. At the same time, there was a clear difference in accuracy.

Another thing to note is that calibrated models did not have a clear advantage over uncalibrated models. They generally had close results, but sometimes one performed better and sometimes the other one. Yet, may be useful to try calibration, as it often improved the results a tiny bit.

In the end, in the multiclass cases, the most important factor for resampling techniques and weighted loss functions seemed to be the class weights algorithm. In this thesis, two different algorithms were used, but it is not enough to formalize general rules on how to choose the best class weights for minimizing total cost. Hence, this is a possible future work direction that has the potential to improve the total cost.

Additionally, in the future more different techniques can be compared to already existing ones. There are many possible opportunities for that. For example, a cost matrix can be directly passed to a loss function and the weight of the loss could be found based on the predicted and actual class combination. This technique would be similar to loss function with class weights, but in theory, it should be able to improve the quality of predictions as it has more information about the costs than the weighted loss function.

References

[BGW03] Ulf Brefeld, Peter Geibel, and Fritz Wysotzki. Support vector machines with example dependent costs. In Nada Lavrač, Dragan Gamberger, Hendrik Blockeel, and Ljupčo Todorovski, editors, *Machine Learning: ECML 2003*, pages 23–34, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[BJ15] Ajay Kumar Boyat and Brijendra Kumar Joshi. A review paper: Noise models in digital image processing. *CoRR*, abs/1505.03489, 2015.

[BMM17] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *CoRR*, abs/1710.05381, 2017.

[Bri50] Glenn W. Brier. Verification of forecasts expressed in terms of probability. 1950.

[CBHK02] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, Jun 2002.

[cC20] Stanford CS class CS231n. *Convolutional Neural Networks for Visual Recognition*, (accessed May 15, 2020).

[CLY15] Yu-An Chung, Hsuan-Tien Lin, and Shao-Wen Yang. Cost-aware pre-training for multiclass cost-sensitive deep learning. *CoRR*, abs/1511.09337, 2015.

[EE05] Rogério Espíndola and Nelson Ebecken. On extending f-measure and g-mean metrics to multi-class problems. *Sixth international conference on data mining, text mining and their business applications*, 35:25–34, 01 2005.

[Elk01] Charles Elkan. The foundations of cost-sensitive learning. In *In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 973–978, 2001.

[GPSW17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. *CoRR*, abs/1706.04599, 2017.

[HSS12] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Lecture 6.5 - rmsprop, coursera. 2012.

[HV03] David J. Hand and Veronica Vinciotti. Choosing k for two-class nearest neighbour classifiers with unbalanced classes. *Pattern Recognition Letters*, 24(9):1555 – 1562, 2003.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[KH92] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 950–957. Morgan-Kaufmann, 1992.

[KK98] Matjaz Kukar and Igor Kononenko. Cost-sensitive learning with neural networks. In *ECAI*, 1998.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[LC10] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[LFMTH12a] Victoria López, Alberto Fernández, Jose Moreno-Torres, and Francisco Herrera. Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. open problems on intrinsic data characteristics. *Expert Systems with Applications*, 39:6585–6608, 06 2012.

[LFMTH12b] Victoria López, Alberto Fernández, Jose Moreno-Torres, and Francisco Herrera. Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. open problems on intrinsic data characteristics. *Expert Systems with Applications*, 39:6585–6608, 06 2012.

[NH10] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[Pla00] John Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Adv. Large Margin Classif.*, 10, 06 2000.

[Pre97] Lutz Prechelt. Early stopping - but when? In *Neural Networks: Tricks of the Trade, volume 1524 of LNCS, chapter 2*, pages 55–69. Springer-Verlag, 1997.

[RK04] Bhavani Raskutti and Adam Kowalczyk. Extreme re-balancing for svms: A case study. *SIGKDD Explorations*, 6:60–69, 06 2004.

[RMW16] Vidwath Raj, Sven Magg, and Stefan Wermter. Towards effective classification of imbalanced data with convolutional neural networks. In Friedhelm Schwenker, Hazem M. Abbas, Neamat El Gayar, and Edmondo Trentin, editors, *Artificial Neural Networks in Pattern Recognition*, pages 150–162, Cham, 2016. Springer International Publishing.

[SMDH13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[Tin98] Kai Ming Ting. Inducing cost-sensitive trees via instance weighting. In Jan M. Żytkow and Mohamed Quafafou, editors, *Principles of Data Mining and Knowledge Discovery*, pages 139–147, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[VCC99] Konstantinos Veropoulos, C. Campbell, and N. Cristianini. Controlling the sensitivity of support vector machines. *Proceedings of International Joint Conference Artificial Intelligence*, 06 1999.

[YNWN08] Ly Yang, MH Nie, Z W Wu, and YY Nie. The operations on matrix for assignment problem. 2008.

[ZL06] Zhi-Hua Zhou and Xu-Ying Liu. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *Knowledge and Data Engineering, IEEE Transactions on*, 18:63– 77, 02 2006.

[ZWZL18] Jian Zheng, Yifan Wang, Xiaonan Zhang, and Xiaohua Li. Classification of severely occluded image sequences via convolutional recurrent neural networks. 09 2018.

Appendix

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Andreas Baum**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Cost-sensitive classification with deep neural networks,

   supervised by Meelis Kull.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Andreas Baum
15/05/2020