Universiteit Leiden
The Netherlands

Learning from small samples

Kocaman, V.

Chapter 4

The Role of Final Batch Normalization Layer

In highly imbalanced classification problems, which encompass complex features, deep learning (DL) is much needed because of its strong detection capabilities. At the same time, DL is observed in practice to favor majority over minority classes and consequently suffer from inaccurate detection of rare events. To simulate this scenario, in this chapter, we will artificially generate skewness (99% vs. 1%) for certain plant types out of the PlantVillage dataset [157] as a basis for classification of scarce visual cues through transfer learning. By randomly and unevenly picking healthy and unhealthy samples from certain plant types to form a training set, we consider a base experiment as fine-tuning ResNet34 and VGG19 architectures and then testing the model performance on a balanced dataset of healthy and unhealthy images. Then we will investigate the role of Batch Normalization (BN) layer in modern CNN architectures to see if it would help minimize the training time and testing error for minority classes in highly imbalanced data sets.

In this chapter we shall present the following results:

1. Utilizing an additional Batch Normalization (BN) layer before the output

layer in modern CNN architectures has a considerable impact in terms of minimizing the training time and testing error for minority classes in highly imbalanced data sets.

2. When the final BN is employed, minimizing the loss function may not be the best way to assure a high F1 test score for minority classes in such problems. That is, the network might perform better even if it is not 'confident' enough while making a prediction; leading to another discussion about why softmax output is not a good uncertainty measure for DL models.

3. The performance gain after adding the final BN layer in highly imbalanced settings could still be achieved after removing this additional BN layer in inference.

4. There is a certain threshold for the *imbalance ratio* upon which the progress gained by the final BN layer reaches its peak.

5. The batch size also plays a role and affects the outcome of the final BN application.

6. The impact of the BN application is also reproducible on other datasets and when utilizing much simpler neural architectures.

7. The reported BN effect occurs only per a single majority class and multiple minority classes – i.e., no improvements are evident when there are two majority classes; and finally, (viii) Utilizing this BN layer with sigmoid activation has almost no impact when dealing with a strongly imbalanced image classification tasks.

4.1 Background

In order to better understand the novel contributions of this study, in this section, we give some background information about the Batch Normalization [74] concept.

Training deep neural networks with dozens of layers is challenging as they can be sensitive to the initial random weights and configuration of the learning algorithm. One possible reason for this difficulty is that the distribution of the inputs to

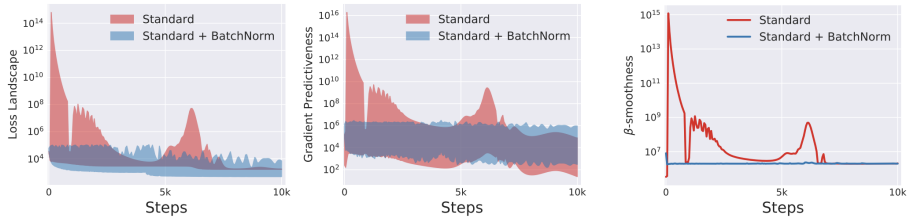
layers deep in the network may change after each mini-batch when the weights are updated. This slows down the training by requiring lower learning rates and careful parameter initialization, makes it notoriously hard to train models with saturating nonlinearities [74], and can cause the learning algorithm to forever chase a moving target. This change in the distribution of inputs to layers in the network is referred to by the technical name "internal covariate shift" (ICS).

BN is a widely adopted technique that is designed to combat ICS and to enable faster and more stable training of deep neural networks (DNNs). It is an operation added to the model before activation which normalizes the inputs and then applies learnable scale (γ) and shift (β) parameters to preserve model performance. Given m activation values $x_1 \ldots, x_m$ from a mini-batch $\mathcal{B}$ for any particular layer input $x^{(j)}$ and any dimension $j \in \{1, \ldots, d\}$, the transformation uses the mini-batch mean $\mu_{\mathcal{B}} = 1/m \sum_{i=1}^{m} x_i$ and variance $\sigma^2_{\mathcal{B}} = 1/m \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$ for normalizing the x_i according to $\hat{x}_i = (x_i - \mu_{\mathcal{B}})/\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}$ and then applies the scale and shift to obtain the transformed values $y_i = \gamma \hat{x}_i + \beta$. The constant $\epsilon > 0$ assures numerical stability of the transformation.

Typically in ML, it is common to normalize input data before passing the data to the input layer. The reason we normalize is partly to ensure that our model can generalize appropriately. This is achieved by ensuring that the scale of the values is balanced, and also the range of the values are maintained and proportional despite the scale change in the values. In a similar context, the BN operation standardizes and normalizes the input values. The input values are then transformed through scaling and shifting operations. BN was performed as a solution to speed up the training phase of deep neural networks through the introduction of internal normalization of the inputs values within the neural network layer. Normalization is typically carried out on the input data, but it would make sense that the flow of internal data within the network should remain normalized. So, we can say that BN is the internal enforcer of normalization within the input values passed between the layer of a neural network. Internal normalization limits the covariate shift that usually occurs to the activations within the layers.

BN has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks; and using BN

makes the network more stable during training. This may require the use of much larger than normal learning rates, which in turn may further speed up the learning process. Though BN has been around for a few years and has become common in deep architectures, it remains one of the DL concepts that is not fully understood, having many studies discussing why and how it works. Most notably, Santurkar et al. [158] recently demonstrated that such distributional stability of layer inputs has little to do with the success of BN and the relationship between ICS and BN is tenuous. Instead, they uncovered a more fundamental impact of BN on the training process: it makes the optimization landscape significantly smoother (Figure 4.1). This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training. Bjorck et al. [159] also makes similar statements that the success of BN can be explained without ICS. They argue that being able to use larger learning rate increases the implicit regularization of the gradient, which improves generalization.



(a) loss landscape (b) gradient predictiveness (c) "effective" β-smoothness

Figure 4.1: Analysis of the optimization landscape during training of deep linear networks with and without BatchNorm. There is a clear improvement in each of these measures of smoothness of the optimization landscape in networks with BatchNorm layers. (source: [158])

Even though BN adds an overhead to each iteration (estimated as additional 30% computation [160]), the following advantages of BN outweigh the overhead shortcoming:

- It improves gradient flow and allows training deeper models (e.g., ResNet).
- It enables using higher learning rates because it eliminates outliers activation, hence the learning process may be accelerated using those high rates.

- It reduces the dependency on initialization and then reduces overfitting due to its minor regularization effect. Similarly to dropout, it adds some noise to each hidden layer's activation.

- Since the scale of input features would not differ significantly, the gradient descent may reduce the oscillations when approaching the optimum and thus converge faster.

- BN reduces the impacts of earlier layers on the following layers in DNNs. Therefore, it takes more time to train the model to converge. However, the use of BN can reduce the impact of earlier layers by keeping the mean and variance fixed, which in some way makes the layers independent from each other. Consequently, the convergence becomes faster.

The development of BN as a normalization technique was a turning point in the development of DL models, and it enabled various networks to train and converge. Despite its great success, BN exhibits drawbacks that are caused by its distinct behavior of normalizing along the batch dimension. One of the major disadvantages of BN is that it requires sufficiently large batch sizes to obtain good results. This prevents the user from exploring higher-capacity models that would be limited by memory. To solve this problem, several other normalization variants are developed, such as Layer Normalization (LN) [161], Instance Normalization (IN) [162], Group Normalization (GN) [163] and Filter Response Normalization (FRN) [164].

4.2 Related Work and Prior Art

Investigating the effect of learnable parameters of BN, scale (γ) and shift (β), on the training of various typical deep neural nets, Wang et al. [165] suggest that there is no big difference in both training convergence and final test accuracy when removing the BN layer following the final convolutional layer from a convolutional neural network (CNN) for standard classification tasks. The authors claim that it is not necessary to adjust the learnable gain and bias along the training process and they show that the use of constant gain and bias is enough and these learneable parameters have little effect on the performance. They also observe that without adaptively updating learnable parameters for BN layers, it often requires less time

for training of very deep neural nets such as ResNet-101.

Frankle et al. [166] also studied the effect of learnable BN parameters, showing that BN's trainable parameters alone can account for much of a network's accuracy. They found that, when locking all other layers at their random initial weights and then training the network for fifty or so epochs, it will perform better than random. As it adjusts a network's intermediate feature representations for a given mini batch, BN itself learns how to do so in a consistent way for all mini batches. The researchers probed the impact of this learning by training only the BN parameters, γ and β, while setting all other parameters at random. Bjorck et al. [159] conducted another experiment, similar to our study. They trained a ResNet that uses one BN layer only at the very last convolutional layer of the network (removing all the other BN layers coming after each CNN layer), normalizing the output of the last residual block but without intermediate activation. This indicates that after initialization, the network tends to almost always predict the same (typically wrong) class, which is then corrected with a strong gradient update. In contrast, the network with BN does not exhibit the same behavior; rather, positive gradients are distributed throughout all classes. Their findings suggest that normalizing the final layer of a deep network may be one of the most important contributions of BN.

Zhu et al. [167] also adopted a similar idea of using the scalable version of a BN layer to normalize the channel at the final output of the network, and improving the classification performance of softmax without adding learnable BN parameters, but controlling the output distribution by another scaling parameter. However, their improvements on selected datasets were not significant (less then 0.3%).

Learning from small samples can also be regarded as out-of-distribution (OoD) detection as the minority class represents the samples out of the distribution of majority classes. While investigating the OoD image detection capability of neural networks without Learning from OoD data, [168] proposed Generalised ODIN (i.e. OoD image detection in neural networks [169]) framework that is composed of two strategies: decomposing confidence scoring and modifying the input pre-processing method. The original ODIN framework [169] also suggests that temperature scaling and adding small perturbations to the input can separate

the softmax score distributions of in- and out-of-distribution images, allowing for more effective detection. Both of these frameworks tamper the confidence scoring in the output layer but mentions BN layer within a regularization context.

Apart from these studies, to the best of our knowledge, there is no prior work investigating the effect of BN utilization over imbalanced classification tasks when set before the softmax output layer.

4.3 Implementation Details and Experimental Results

In this study, we primarily utilize the ResNet34 CNN architecture due to computational requirements and a need for an iterative process to conduct a large number of experiments. ResNet is evaluated on the ImageNet with a depth of up to 152 layers - 8 times deeper than VGG nets but still having lower complexity. An ensemble of these residual nets achieved a 3.57% error on the ImageNet test set. This result won the 1st place in the ILSVRC 2015 classification task [170].

Our training process can be regarded as fine-tuning based on ImageNet checkpoints using transfer learning. We firstly addressed the complete PlantVillage original dataset and fine-tuned a ResNet34 model for 38 classes. Using scheduled learning rates, we obtained 99.782% accuracy after 10 epochs – slightly improving the PlantVillage project's record of 99.34% using GoogleNet [171]. Having reproducing these results, relatively easily, further boosts our confidence in selecting ResNet34 for the task.

4.3.1 Dataset Details

A prominent effort towards plant diseases' detection is the Plant Village Disease Classification Challenge. As part of the PlantVillage project, 54,306 images of 14 crop species with 26 diseases (including healthy, 38 classes in total) were made publicly available [157]. The detailed distribution of the dataset and its classes can be seen at Table 4.1.

Following the release of the PlantVillage dataset, deep learning (DL) was intensively employed [171], with reported accuracy ranging from 85.53% to 99.34%. Without

Table 4.1: Plant Village dataset distribution for 54,306 images of 14 crop species with 26 diseases. Each specie has certain number of *unhealthy* classes (e.g. apple scab and black rot for Apple, bacterial spot and leaf mold for Tomato) and one *healthy* class, denoted in Class Size column as the total number of classes.

	Training Set		Validation Set		Class Size
	Healthy	Unhealthy	Healthy	Unhealthy	
Potato	121	1600	31	400	3
Peach	288	1838	72	459	2
Cherry	684	842	170	210	2
Grape	339	2912	84	727	4
Tomato	1272	13132	318	3284	10
Pepper	1181	797	295	200	2
Corn	16	2005	5		4
Orange	0	4405	0	1102	2
Blueberry	1202	0	300	0	2
Apple	1316	1220	329	306	4
Squash	0	1448	0	365	2
Soybean	4072	0	1018	0	2
Raspberry	297	0	74	0	2
Strawberry	364	886	92	222	2

any feature engineering, the best reported model correctly classifies crop and disease in 993 out of 1,000 images (out of 38 possible classes). The PlantVillage dataset is slightly imbalanced, such that accuracy is commonly used as the performance measure.

In this study, to simulate rare events in agriculture, we capitalize on this dataset to synthetically generate skewness and address our research question. With the first model, using ResNet34 architecture [170] with pre-trained weights on ImageNet and tuning the model for PlantVillage data sets, we reach 99.782% accuracy after 10 epochs, i.e., we were able to correctly classify crop and disease from 38 possible classes in 998 out of 1,000 images. This result can be attributed to the representation power of the ResNet architecture as well as to having a large number of crop disease samples from different classes. Upon firstly reproducing the accuracy rates reported in [171] on the original dataset, we considered the classification problem under the generated imbalance.

We randomly picked 1,000 healthy and 10 unhealthy samples from a plant type for the training set, 150 healthy and 7 unhealthy samples for the validation set, fine tuned using the ResNet34 architecture, and then tested the model performance on an equal number of healthy and unhealthy images (150 vs 150) in a test set. We did this for 3 different plant types: Apple, Pepper, and Tomato. Training on just 10 samples and validating on 7 samples from minority class, we managed to correctly predict more than 141 of 150 unhealthy images spanning over multiple anomalies that may not exist in the training or validation set. When experimenting with different configurations, we discovered a significant improvement of classification performance by adding a final BN layer just before the output layer. We did more experiments in the VGG19 framework [172] under the same settings, and managed to correctly predict more than 143 of 150 unhealthy images, which is higher than what we got through ResNet34: 130 out of 150.

4.3.2 Adding a Final Batch Norm Layer Before the Output Layer

By using the imbalanced datasets for certain plant types (1,000 healthy/10 unhealthy samples in the training set, 150/7 in the validation set and 150/150 in

the test set), we performed several experiments with the VGG19 and ResNet34 architectures. The selected plant types were Apple, Pepper and Tomato - being the only datasets of sufficient size to enable the 99%-1% skewness generation. We treated unhealthy class as a minority class due to the fact that the abnormalities are rare in the real-world as well and gathering the unhealthy images is harder compared to healthy ones. So, we set the ratio of unhealthy images to 1% in train set and 5% in validation set. But we created a balanced dataset for test set to test the model performance on unseen images that are in an equal number for both classes.

In order to fine-tune our network for the PlantVillage dataset, the final classification layer of CNN architectures is replaced by Adaptive Average Pooling (AAP), BN, Dropout, Dense, ReLU, BN and Dropout followed by the Dense and BN layer again. The last layer of an image classification network is often a fully-connected layer with a hidden size being equal to the number of labels to output the predicted confidence scores that are normalized by the softmax operator to obtain predicted probabilities. In our implementation, we add another 2-input BN layer after the last dense layer (before softmax output) in addition to existing BN layers in the tail and 4 BN layers in the head of the DL architecture (e.g., ResNet34 possesses a BN layer after each convolutional layer, having altogether 38 BN layers given the additional 4 in the head). A schematic outline of this architecture is provided in Figure 4.2.

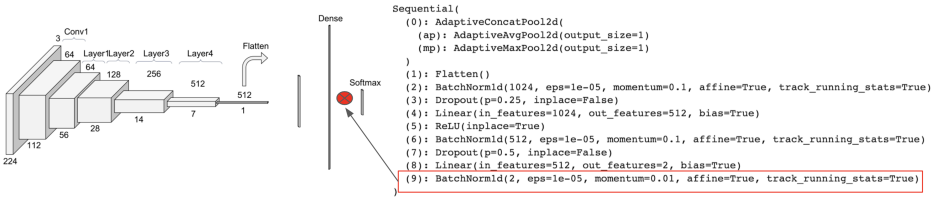


Figure 4.2: Implementation of the final BN layer in ResNet34, just before the softmax output.

At first we run experiments with VGG19 architectures for selected plant types by adding the final BN layer. When we train this model for 10 epochs and repeat this for 10 times, we observed that the F1 test score is increased from 0.2942 to 0.9562 for unhealthy Apple, from 0.7237 to 0.9575 for unhealthy Pepper and from 0.5688

to 0.9786 for unhealthy Tomato leaves. We also achieved substantial improvement in healthy samples (being the majority in the training set). For detailed metrics and charts, see Figure 4.3 and Table 4.2.

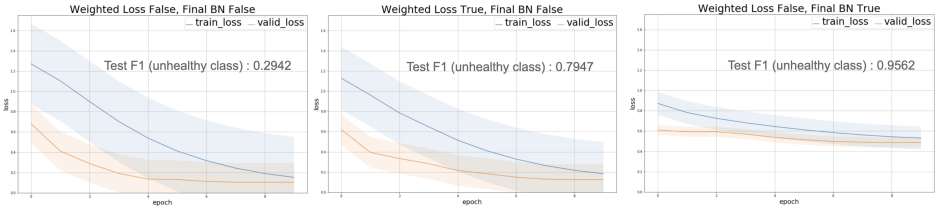


Figure 4.3: Averaged loss charts over 10 runs for the Apple dataset without final BN (left), with final BN layer (right) and with WL (center) in VGG19 architecture. Without WL, after adding the final BN layer, the test F1 score is increased from 0.2942 to 0.9562 for the minority class while the train and validation losses level off around 0.5 before the model begins to overfit. It is also evident that the standard deviation among runs (depicted as the shaded areas surrounding a given curve within each plot) is lower for the latter case, as the training becomes smoother therein.

Table 4.2: Averaged **F1 test set** performance values over 10 runs, alongside BN's total improvement, using 10 epochs with VGG19, with/without BN and with Weighted Loss (WL) without BN.

Plant	Class	without final BN	with WL (no BN)	with final BN (no WL)	BN total improvement
Apple	Unhealthy	**0.2942**	**0.7947**	**0.9562**	**0.1615**
	Healthy	0.7075	0.8596	0.9577	0.0981
Pepper	Unhealthy	**0.7237**	**0.8939**	**0.9575**	**0.0636**
	Healthy	0.8229	0.9121	0.9558	0.0437
Tomato	Unhealthy	**0.5688**	**0.8671**	**0.9786**	**0.1115**
	Healthy	0.7708	0.9121	0.9780	0.0659

In order to explain which parts of the image the network was looking at when it made a prediction, we experimented with class-discriminative activation maps (a generic localizable deep representation). Class activation maps [173], commonly called CAMs, are class-discriminative saliency maps. While saliency maps give information on the most important parts of an image for a particular class, class-discriminative saliency maps help distinguish between classes. We run experiments for *Apple* class with and wothout final BN layer using ResNet34 and visualized the activation maps. As you can see at Figure 4.4, the model without final BN layer

looks at the irrelevant parts of the picture and predicting wrong while it is looking at the right parts when the final BN layer is added.

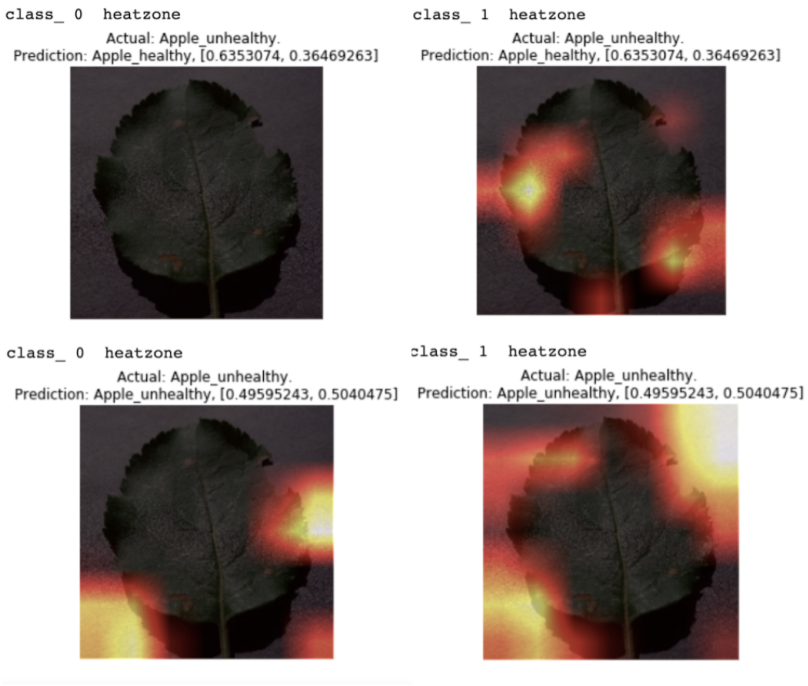


Figure 4.4: Class-discriminative activation maps of a sample prediction under two settings (with and without final BN layer). Class-0 indicates *Healthy* and Class-1 indicates *Unhealthy*. The first row belongs to the predictions under vanilla settings and the second row belongs to the predictions when final BN layer is added. With final BN, the activations for *Unhealthy* class is glowing over the wrong pixels and it is predicted as *Healthy*. The probability of being *Unhealthy* is 36.5% and we see some minor activations closer to blight areas while no activation is happening for *Healthy* class. On the other hand, with final BN, the activations for *Unhealthy* class is glowing over the right pixels and it is predicted as *Unhealthy*. The probability of being *Healthy* is 49.6% and we see some minor activations closer to *Healthy* areas.

The highlighted parts (heatmap) basically tells us which parts of an image induced the wrong classification. In this example, the model made the wrong prediction because of the highlighted part.

4.3.3 Experimentation Subject to Different Configurations

Using the following six configuration variations with two options each, we created 64 different configurations which we tested with ResNet34 (training for 10 epochs only): Adding (✓) a final BN layer just before the output layer (BN), using (✓) weighted cross-entropy loss [174] according to class imbalance (WL), using (✓) data augmentation (DA), using (✓) mixup (MX) [175], unfreezing (✓) or freezing (learnable vs pre-trained weights) the previous BN layers in ResNet34 (UF), and using (✓) weight decay (WD) [176]. Checkmarks (✓) and two-letter abbreviations are used in Table 4.3 and Table 4.4 to denote configurations. When an option is disabled across all configurations, its associated column is dropped.

Table 4.3: Best performance metrics over the Apple dataset under various configurations using ResNet34.

Class	Config Id	Test set precision	Test set recall	Test set F1-score	Epoch	BN	DA	UF	WD
Unhealthy (class = 1)	**31**	0.9856	0.9133	**0.9481**	6	✓			
	23	0.9718	0.9200	0.9452	6	✓	✓		
	20	0.9926	0.8933	0.9404	7	✓	✓	✓	✓
Healthy (class = 0)	31	0.9193	0.9867	0.9518	6	✓			
	23	0.9241	0.9733	0.9481	6	✓	✓		
	20	0.9030	0.9933	0.9460	7	✓	✓	✓	✓

As shown in Table 4.3, just adding the final BN layer was enough to get the highest F1 test score in both classes. Surprisingly, although there is already a BN layer after each convolutional layer in the base CNN architecture, adding one more BN layer just before the output layer boosts the test scores. Figure 4.5 shows that we gain a boost in Test F1 score by more than double (by 1.2 times) just by adding a final BN layer.

Notably, the 3rd best score (average score for configuration 31 in Table 4.4) is achieved just by adding a single BN layer before the output layer, even without unfreezing the previous BN layers. Moreover, it is evident that the additional BN layer is never utilized among the worst performing configurations (see Table 4.4).

The model without the final BN layer is pretty confident even if it predicts falsely. But the proposed model with the final BN layer predicts correctly even though

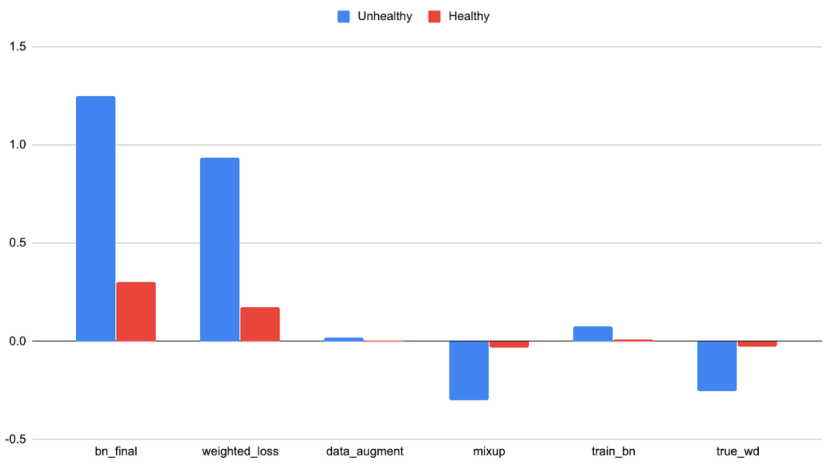


Figure 4.5: The ratio of increase in Test-F1 scores per each configuration. The highest relative gain in Test-F1 score for both classes is achieved by adding a final BN layer (only one parameter is set to True at a time). The experiments show that the enhancement is more than doubled just by adding a final BN layer (e.g., from 0.42 to 0.95 translates into a (0.95-0.42)/0.42 = 1.2 gain factor).

Table 4.4: Best (top three) and worst (bottom three) performing configurations (F1 measure, for the unhealthy/minority class) when using ResNet34 for the Apple, Pepper and Tomato datasets.

Config Id	Apple	Pepper	Tomato	Average	BN	DA	MX	UF	WD
29	0.9332	0.9700	0.9866	**0.9633**	✓			✓	
28	0.9332	0.9566	0.9966	0.9622	✓			✓	✓
31	0.9499	0.9433	0.9833	**0.9588**	✓				
49	0.6804	0.5080	0.5339	0.5741		✓	✓	✓	
48	0.5288	0.5638	0.5638	0.5521		✓	✓	✓	✓
50	0.6377	0.5027	0.4528	0.5311		✓	✓		✓

it is less confident. We end up with less confident but more accurate models in less than 10 epochs. The classification probabilities for five sample images from the unhealthy class (class = 1) with final BN layer (right column) and without final BN layer (left column) are shown in Table 4.5. As explained above, without the final BN layer, these anomalies are all falsely classified (recall that $\mathcal{P}_{\text{softmax}}(\text{class} = 0) = 1 - \mathcal{P}_{\text{softmax}}(\text{class} = 1)$).

Table 4.5: Softmax output values (representing class probabilities) for five sample images of unhealthy plants. Left column: Without final BN layer, softmax output values for unhealthy, resulting in a wrong classification in each case. Right column: With final BN layer, softmax output value for unhealthy, resulting in correct but less "confident" classifications.

Without final BN layer	With final BN layer
0.1082	0.5108
0.1464	0.6369
0.1999	0.6082
0.2725	0.6866
0.3338	0.7032

4.3.3.1 Removing the additional BN layer during inference

Since adding the final BN layer adds a small overhead (four new parameters) to the network at each iteration, we experimented if the final BN layer could be dropped once the training is finished so that we can avoid the cost. Dropping this final BN layer means that training the network from end to end, and then chopping off the final BN layer from the network before saving the weights. We tested this hypothesis for Apple, Pepper and Tomato images from PV dataset under three conditions with 1% imbalance ratio: *Without final BN, with final BN and then removing the final BN during testing*. We observed that removing the final BN layer in inference would still give us a considerable boost on minority class without losing any performance gain on the majority class. The results in Table 4.6 show that the performance gain is very close to the configuration in which we used the final BN layer both in training and inference time. As a consequence, we confirm an hypothesis that the final BN layer can indeed be removed in inference without compromising the performance gain.

Table 4.6: Training with the final BN layer, and then dropping this layer while evaluating on the test set proved to be still useful in terms of improving the classification score on minority classes, albeit not as much as with the final BN layer kept (imbalance ratio 0.01, epoch 10, batch size 64).

	Apple		Pepper		Tomoto	
	healthy	unhealthy	healthy	unhealthy	healthy	unhealthy
with no final BN	0.71	0.22	0.74	0.45	0.74	0.46
with final BN	0.92	0.91	0.94	0.94	0.98	0.98
train with final BN remove while testing	0.74	**0.83**	0.75	**0.82**	0.78	**0.85**

4.3.3.2 Impact level regarding the imbalance ratio and the batch size

In the previous sections, we empirically showed that the final BN layer, when placed before the softmax output layer, has a considerable impact in highly imbalanced image classification problems but they fail to explain the impact of using the final BN layer as a function of level of imbalance in the training set. In order to find if there is a certain ratio in which the impact is maximized, we tested this hypothesis (H-1) over various levels of imbalance ratios and conditions explained below. In sum, we ended up with 430 model runs, each with 10 epochs (basically tested with 5, 10, 15, .. 100 unhealthy vs 1000 healthy samples). During the experiments, we observed that the impact of final BN on highly imbalanced settings is the most obvious when the ratio of minority class to the majority is less than 10%; above that almost no impact. As expected, the impact of the final BN layer is more obvious on minority class than it is on majority class, albeit the level of impact with respect to the imbalance ratio is almost same, and levels off around 10%. It is mainly because of the fact that the backbone architecture (ResNet34) is already good enough to converge faster on such data set (Plant Village) and the model does well on both classes after 10% imbalance (having more than 100 unhealthy with respect to 1000 healthy samples can already be handled regardless of final BN trick). During these experiments, we also tested if unfreezing the previous layers in the backbone CNN architecture (ResNet34) would also matter. We observed that unfreezing the pretrained layers helps without even final BN layer, but unfreezing adds more computation as the gradient loss will be calculated for each one of them. After adding the final BN layer and freezing the previous pretrained layers, we observed similar metrics and learning pattern as we did with unfreezing but not

with final BN layer. This is another advantage of using the final BN that allow us to freeze the previous pretrained layers. The results are displayed as charts in Figure 4.6 and Figure 4.7.

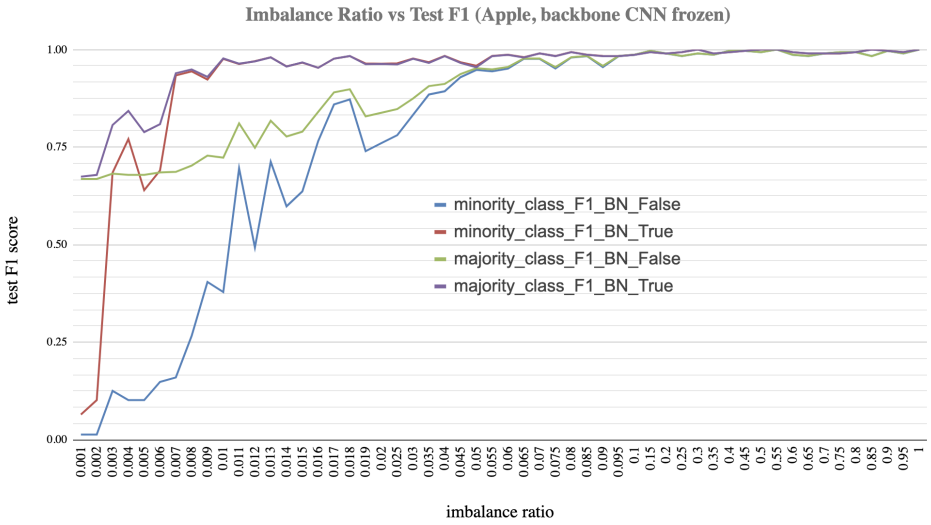


Figure 4.6: The impact of final BN layer on the F1 test score of each class for Apple plant. The impact is the most obvious when the ratio of minority class to the majority is less than 0.1.

We also experimented if the batch size would also be an important parameter for the minority class test accuracy when the final BN is added and found out that the highest score is achieved when the batch size is around 64, whereas the accuracy drops afterwards with larger batches. The results are exhibited in Figure 4.7.

4.3.3.3 Experimentation on the MNIST Dataset: The impact of final BN layer in basic CNNs and FC networks

In order to reproduce equivalent results on a well known benchmark dataset when utilizing different DL architectures, we set up two simple DL architectures: a CNN network with five Conv2D layers and a one-layer (128-node) FC feed forward NN. Then we sampled several pairs of digits (2 vs 8, 3 vs 8, 3 vs 5 and 5 vs 8) from MNIST dataset that are mostly confused in a digit recognition task due to similar patterns in the pixels. During the experiments, the ratio of minority

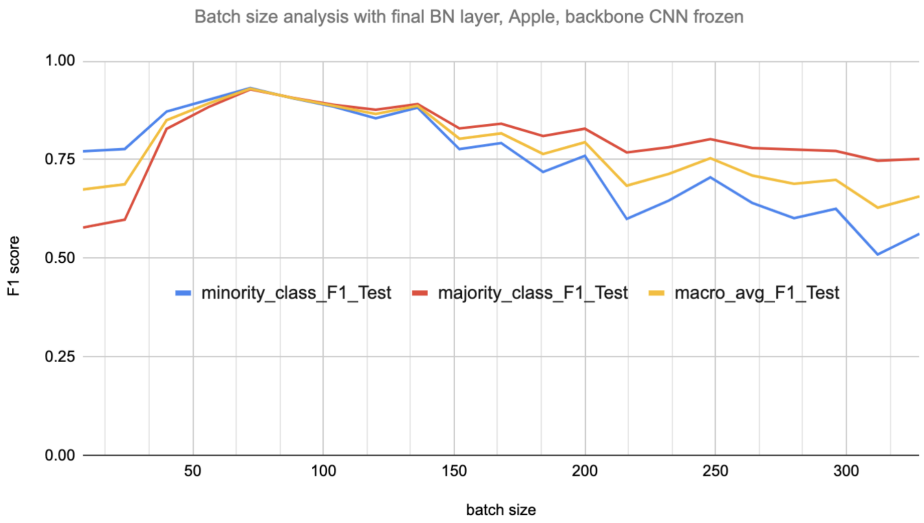


Figure 4.7: The impact of batch size on the F1 test score when the final BN layer added. The highest score is gained when the batch size is 64 and then the accuracy starts declining.

class to majority is kept as 0.1 and experiments with the simple CNN architecture indicates that, after adding the final BN layer, we gain $\sim$ 20% boost in minority class and $\sim$ 10% in majority class. Lower standard deviations across the runs with final BN layer also indicates the regularization effect of using the final BN layer. As a result, we rejected a hypothesis that the performance gain through the final BN layer can be reproduced with another dataset or with much simpler neural architectures. Another important finding is that the final BN layer boosts the minority class' F1 scores only when there is a single majority class. When two majority classes are set, no improvements are evident regardless of the usage of the final BN layer (see 4th and 5th settings in Table 4.7). Therefore, we partially confirm hypothesis (H-5) by showing that the performance gain through the final BN layer can also be achieved in multi-classification settings but the number of majority and minority classes may affect the role of the final BN layer. In the experiments with a one-layer FC network, we enriched the scope and also tested whether using different loss functions and output activation functions would have an impact on model performance using final BN layer with or without another BN

layer after the hidden layer. We observed that adding the final BN layer, softmax output layer and categorical crossentropy (CCE) as a loss function have the highest test F1 scores for both classes. It is also important to note that we observe no improvement even after adding the final BN layer when we use sigmoid activation in the output layer (Table 4.7). Accordingly, we reject hypothesis (H-6).

Table 4.7: Using the same ResNet-34 architecture and skewness (1% vs 99%) and setting up five different configurations across various confusing classes from MNIST dataset, it is clear that adding the final BN layer boosts the minority class F1 scores by 10% to 30% only when there is a single majority class. When we have two majority classes, there is no improvement observed regardless of the final BN layer is used or not. (✓) indicates minority classes.

	Setting-1		Setting-2		Setting-3		Setting-4			Setting-5		
	3 (✓)	8	2 (✓)	8	3 (✓)	5	3 (✓)	5	8	3 (✓)	5 (✓)	8
without final BN	0.19	0.69	0.25	0.70	0.24	0.70	0.06	0.77	0.78	0.28	0.32	0.56
with final BN	0.55	0.76	0.50	0.74	0.54	0.76	0.00	0.77	0.78	0.58	0.59	0.69

Table 4.8: Using one-layer (128 node) NN and the 0.01 skewness ratio, with nine different settings for 3 (minority) and 8 (majority) classes from MNIST dataset (100-epoch). Adding the final BN layer, softmax output layer and CCE as a loss function has the highest test F1 scores for both classes. (CCE - categorical cross entropy, BCE - binary cross entropy, first BN - a BN layer after the hidden layer).

output activation	loss function	first BN layer	final BN layer	class-3 (minority)	class-8 (majority)
sigmoid	BCE			0.17	0.67
softmax	BCE			0.00	0.67
softmax	BCE	✓		0.60	0.78
softmax	CCE			0.67	0.80
sigmoid	BCE		✓	0.05	0.67
softmax	BCE		✓	0.85	0.88
softmax	BCE	✓	✓	0.83	0.87
softmax	CCE		✓	**0.88**	**0.90**
softmax	CCE	✓	✓	0.78	0.85

4.4 Discussion

The metrics at Table 4.2 clearly indicate that after adding the final BN layer, the F1 test score is increased from 0.2942 to 0.9562 for unhealthy Apple, from 0.7237 to 0.9575 for unhealthy Pepper and from 0.5688 to 0.9786 for unhealthy Tomato leaves on average (10 runs). When compared to a network with WL but no final

BN layer, the average improvement was 0.1615, 0.0636 and 0.1115 respectively.

To see what would have happened had we let the models further train, we conducted a 100-epoch training for both models and compared the results. We observed that the test accuracy with the model having a final BN layer quickly rises to 94% within only 6 epochs, but then decreases steadily to 78% within a total of 100 epochs.

By adding a final BN layer just before the output, we normalize the output of the final dense layer, and feed into the softmax layer so that we shrink the gap between class activations (see Figure 4.8). As its name suggests, softmax will always favor the large values when the layer outputs spread over large ranges (exponential normalization). By applying BN to dense layer outputs, the gap between activations is reduced (normalized) and then softmax is applied on normalized outputs, which are centered around the mean. Therefore, we end up with centered probabilities (around 0.5), but favoring the minority class by a small margin.

This shows that DNNs have the tendency of becoming 'over-confident' in their predictions during training, and this can reduce their ability to generalize and thus perform as well on unseen data. Actually, softmax activation leading to an over-confident outputs is already a well-studied phenomenon and observed especially when a neural network is trained with the cross-entropy loss as softmax classifiers tend to output a highly confident prediction [177]. [178] sheds another light on this topic and argues that the deep stack of non-linear layers in between the input and the output unit of a neural network are a way for the model to encode a non-local generalization prior over the input space and it makes output unit to assign nonsignificant probabilities to regions of the input space that contain no training examples in their vicinity.

In addition, large datasets can often comprise incorrectly labeled data, meaning inherently the DNN should be a bit skeptical of the 'correct answer' to avoid being overconfident on bad answers. This was the main motivation of Müller et al. [179] for proposing the *label smoothing*, a loss function modification that has been shown to be effective for training DNNs. Label smoothing encourages the activations of the penultimate layer to be close to the template of the correct class and equally

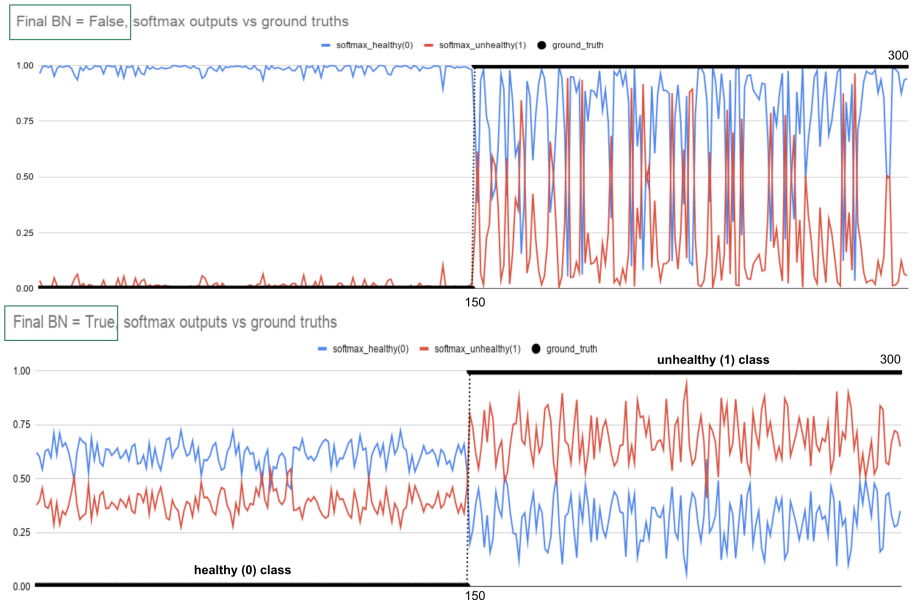


Figure 4.8: The x-axis represents the ground truth for all 150 healthy (0) and 150 unhealthy (1) images in the test set while red and blue lines represent final softmax output values between 0 and 1 for each image. **Top chart (without final BN):** When ground truth (black) is class = 0 (healthy), the softmax output for class = 0 is around 1.0 (blue, predicting correctly). But when ground truth (black) is class = 1 (unhealthy), the softmax output for class = 1 (red points) changes between 0.0 and 1.0 (mostly below 0.5, NOT predicting correctly). **Bottom chart (with final BN):** When ground truth (black) is class = 0 (healthy), the softmax output is between 0.5 and 0.75 (blue, predicting correctly). When ground truth (black) is class = 1 (unhealthy), the softmax output (red points) changes between 0.5 and 1.0 (mostly above 0.5, predicting correctly).

distant to the templates of the incorrect classes [179]. Despite its relevancy to our research, label smoothing did not do well in our study as previously mentioned by Kornblith et al. [180] who demonstrated that label smoothing impairs the accuracy of transfer learning, which similarly depends on the presence of non-class-relevant information in the final layers of the network.

Another observation is that a CNN with final BN is more calibrated when compared to an equivalent network lacking that layer. The calibration principle [181] states that *the probability associated with the predicted class label should reflect its ground truth correctness likelihood* and a model is calibrated if its predicted

outcome probabilities reflect their accuracy. Put it simply, neural networks output "confidence" scores along with predictions in classification. Ideally, these confidence scores should match the true correctness likelihood. For example, if we assign 80% confidence to 100 predictions, then we'd expect that 80% of the predictions are actually correct. If this is the case, we say that the network is calibrated. However, this is not always being observed even in state-of-the-art neural architectures.

We tested the calibration of both networks (with and without final BN) by drawing the reliability diagrams as suggested by Guo et al. [181] and found that when the final BN layer is added, a network has much lower ECE (Expected Calibration Error), i.e., ideal confidence relative to its own accuracy. In effect, a final BN layered network is not 'over-confident' and as a result generalizes and performs better on real world data. This is also confirmed by the fact that the network with a final BN layer has a lower Brier score [182] *(squared error between the predicted probability vector and the one-hot encoded true response)* than without a final BN layer. Our calibration experiments (Figure 4.9.) show that that a network has much lower ECE, i.e. has a more ideal confidence relative to its own accuracy. In effect, a final BN-layered network is not 'over-confident' and as a result generalizes and performs better on unseen datasets.

Figure 4.8 shows that softmax outputs level off around 0.5, as also depicted in Figure 4.3. Since the network loss is optimized by using softmax output value and the ground truth, focusing too much on loss minimization may not be viable when the final BN layer is used in similar settings.

Additionally, we empirically demonstrated that the final BN layer could still be eliminated in inference without compromising the attained performance gain. This finding supports the assertion that adding the BN layer makes the optimization landscape significantly smoother, which in turn renders the gradients' behavior more predictive and stable – as suggested by [158]. We argue that the learned parameters, which were affected by the addition of the final BN layer under imbalanced settings, are likely sufficiently robust to further generalization on the unseen samples, even without normalization prior to the softmax layer.

The observation of locating a sweet spot (10%) for the imbalance ratio, at which

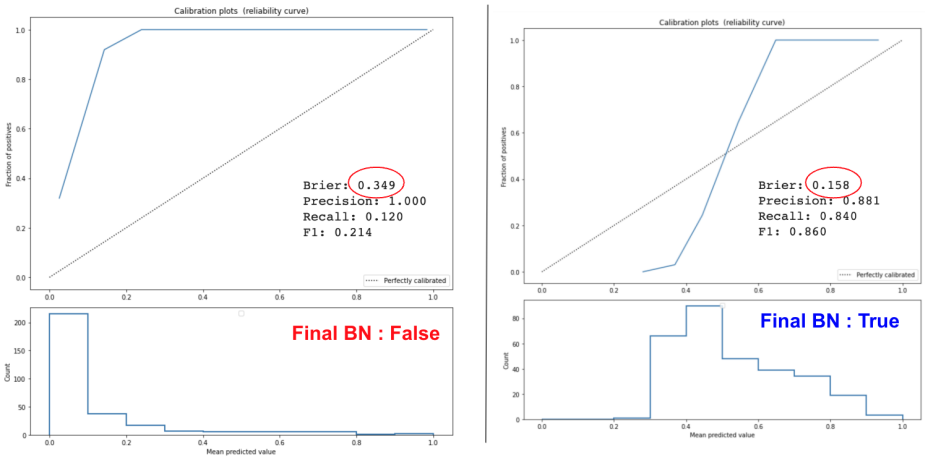


Figure 4.9: The Brier score measures the mean squared difference between the predicted probability assigned to the possible outcomes for item and the actual outcome. Therefore, the lower the Brier score is for a set of predictions, the better the predictions are calibrated. In this chart we see that a network without a final BN layer has a Brier score 0.349 while the one with the final BN has 0.158, indicating a network much better calibrated.

we can utilize the final BN layer, might be explained by the fact that the backbone ResNet architecture is already strong enough to easily generalize on the PV dataset and the model does not need any other regularization once the number of samples from the minority class exceeds a certain threshold. As the threshold found in our experiments is highly related to the DL architecture and the utilized dataset, it is clear that it may not apply to other datasets, but can be found in a similar way.

As discussed before, the BN layer calculates mean and variance to normalize the previous outputs across the batch, whereas the accuracy of this statistical estimation increases as the batch size grows. However, its role seems to change under the imbalanced settings – we found out that a batch size of 64 reaches the highest score, whereas by utilizing larger batches the score consistently drops. As a possible explanation for this observation, we think that the larger the batches, the higher the number of majority samples in a batch and the lesser the chances that the minority samples are fairly represented, resulting in a deteriorated performance.

During our experiments, we expected to see similar behavior with sigmoid activation

replacing softmax in the output layer, but, evidently, the final BN layer works best with softmax activations. Although softmax output may not serve as a good uncertainty measure for DNNs compared to sigmoid layer, it can still do well on detecting the under-represented samples when used with the final BN layer.

Moreover, large datasets can often comprise incorrectly labeled data, meaning inherently the DNN should be a bit skeptical of the 'correct answer' to avoid being overconfident on bad answers. This was the main motivation of Müller et al. [179] for proposing the *label smoothing*, a loss function modification that has been shown to be effective for training DNNs. Label smoothing encourages the activations of the penultimate layer to be close to the template of the correct class and equally distant to the templates of the incorrect classes [179]. Despite its relevancy to our research, label smoothing did not do well in our study as previously mentioned by Kornblith et al. [180] who demonstrated that label smoothing impairs the accuracy of transfer learning, which similarly depends on the presence of non-class-relevant information in the final layers of the network.

4.4.1 Additional Experiments

In addition to the aforementioned experiments, we ran more tests to further support our findings: Turning off the bias at the final dense layer when the final BN layer is added, using SeLU [183] instead of BN in the last layer, freezing and unfreezing the previous BN layers, label smoothing, training a simple CNN from scratch rather than utilizing SOTA architectures.

Also, additional corroboration was achieved over the ISIC Skin Cancer dataset [184] as well as the Wall Crack dataset [185] exhibiting similar patterns of behavior and thus supporting our reported findings. These observations and the code to reproduce the results mentioned in this paper are fully reported in the supplementary material.

4.5 Conclusions

This chapter demonstrated that regardless of freezing or unfreezing the previous layers in the pre-trained modern CNN architectures, putting an additional BN layer just before the softmax output layer has a considerable impact in terms of minimizing the training time and test error for minority classes in a highly

imbalanced dataset (especially in a setting where a high recall is desired for the minority class). Our experiments show that using ResNet34 we can achieve an F1 test score of 0.94 in 10 epochs for the minority class while we could only achieve 0.42 without using the final BN layer. Using VGG19, the same approach increases the F1 test score from 0.25 to 0.96 in 10 epochs. Using VGG19, the same approach increases the F1 test score from 0.25 to 0.96 in 10 epochs. As evident from Table 4.2, upon adding the final BN layer the F1 test score is increased from 0.2942 to 0.9562 for the unhealthy Apple minority class, from 0.7237 to 0.9575 for the unhealthy Pepper and from 0.5688 to 0.9786 for the unhealthy Tomato when WL is not used (all are averaged values over 10 runs). When compared to a network with WL but without a final BN layer, the averaged improvements were 0.1615, 0.0636 and 0.1115 respectively. We showed that the highest gain in test F1 score for both classes (majority vs. minority) is achieved just by adding a final BN layer, resulting in a more than three-fold performance boost on some configurations.

We also argueed that trying to minimize validation and train losses may not be an optimal way of getting a high F1 test score for minority classes. We presented that having a higher train and validation loss but high validation accuracy would lead to higher F1 test scores for minority classes in less time. That is, the model might perform better even if it is not confident enough while making a prediction. So, having a confident model (lower loss, higher accuracy) may favor the majority class in the short run and ends up with lower F1 test scores for minority class due to the conjecture that it learns to minimize error in majority class regardless of class balance. Since BN (like dropout) adds stochasticity to the network, and the network learns to be robust to this stochasticity during the entire training, it was expected that the final BN layer would be detrimental for the network to cope with the stochasticity right before the output [167]. However, we observed the contrary in our experiments.

This chapter also showed that lower values in the softmax output may not necessarily indicate 'lower confidence level', leading to another discussion why softmax output may not serve as a good uncertainty measure for DNNs. In classification, predictive probabilities obtained at the end of the pipeline (the softmax output) are often wrongly interpreted as model confidence. As we have clearly shown, a model can be uncertain in its predictions even when having a high softmax output [186].

4.5. Conclusions

In the experiments mentioned in this chapter, we noticed that the performance gain after adding the final BN layer in highly imbalanced settings could still be achieved after removing this additional BN layer during inference; in turn enabling us to get a performance boost with no additional cost in production. Then we explored the dynamics of using the final BN layer as a function of the imbalance ratio within the training set, and found out that the impact of final BN on highly imbalanced settings is the most apparent when the ratio of minority class to the majority is less than 10%; there is hardly any impact above that threshold.

We also ran similar experiments with simpler architectures, namely a basic CNN and a single-layered FC network, when applied to the MNIST dataset under various imbalance settings. The simple CNN experiments exhibited a gain of $\sim 20\%$ boost per the minority class and $\sim 10\%$ per the majority class after adding the final BN layer. In the FC network experiments, we observed improvements by 10% to 30% only when a single majority class was defined; no improvements were evident for two majority classes, regardless of the usage of the final BN layer. While experimenting with different activation and cost functions, we found out that using the final BN layer with sigmoid activation had almost no impact on the task at hand.

Overall, our study has shed light on the unexpected impact of the final BN layer in simpler neural networks while dealing with imbalanced datasets. This discovery presents exciting opportunities for further investigation in the future, as it requires a more comprehensive analysis to fully understand its implications. Moving forward, we suggest that future researchers explore the generalization of our findings to a wide range of neural models, utilizing a combination of softmax activation or an appropriate loss function for use in imbalanced image classification problems.

Additionally, we recommend examining the potential implications of our findings in real-world scenarios, such as improving the accuracy of image classification in medical diagnosis or object recognition in autonomous vehicles. We believe that exploring these avenues will not only advance our understanding of neural networks, but also have practical implications for a variety of fields.

To sum, the overall contributions of this chapter have been the following compo-

nents:

- Putting an additional BN layer just before the output layer has a considerable impact in terms of minimizing the training time and test error for minority classes in highly imbalanced image classification tasks. Our experiments show that the initial F1 test score increases from the 0.29-0.56 range to the 0.95-0.98 range for the minority class when we added a final BN layer just before the softmax output layer.

- Trying to minimize validation and train losses may not be an optimal way for getting a high F1 test score for minority classes in binary anomaly detection problems. We illustrate that having a higher training and validation loss but high validation accuracy leads to higher F1 test scores for minority classes in less time and using the final BN layer has a calibration effect. That is, the network might perform better even if it is not 'confident' enough while making a prediction. We will also argue that lower values in softmax output may not necessarily indicate 'lower confidence level', leading to another discussion about why softmax output is not a good uncertainty measure for deep learning (DL) models.

- The performance gain after adding the final BN layer in highly imbalanced settings could still be achieved after removing this additional BN layer during inference; in turn enabling us to get a performance boost with no additional cost in production.

- There is a certain threshold for the ratio of the imbalance for this specific PV dataset, upon which the progress is the most obvious after adding the final BN layer.

- The batch size also plays a role and significantly affects the outcome.

- We replicated the similar imbalanced scenarios in MNIST dataset, reproduced the same BN impact, and furthermore demonstrated that the final BN layer has a considerable impact not just in modern CNN architectures but also in simple CNNs and even in one-layered feed-forward fully connected (FC) networks.

- We illustrate that the performance gain occurs only when there is a single majority class and multiple minority classes; and no improvement observed regardless of the final BN layer when there are two majority classes.
- We argue that using the final BN layer with sigmoid activation has almost no impact when dealing with a strongly imbalanced image classification tasks.