Assignment 4: Random Forest Analysis of Tree Size

Grant Booysen 25849646
Email: 25849646@sun.ac.za

***Abstract*—The report follows the analysis of Random Forests as an architecture for classification. The aim is to determine the effect of the different configurable parameters of the Random Forest to determine the effect that each has on the performance of the model. Specifically, the number of trees in the forest and the maximum depth of each tree examined. The analysis is done for three datasets of varying complexity to ensure that the results are not specific to a particular dataset. The results indicate that increasing the tree depth only improves the performance up until a certain point, after which the test performance stagnates for a fixed number of features considered at each split. Secondly, after fixing the maximum depth of the trees to the optimal value found in the first analysis, the number of amount of features considered at each split was varied. The results indicate that increasing the number of features considered at each split did not improve the performance of the model.**

***Index Terms*—Random Forest, Ensemble learning**

I. Introduction

The remainder of this report is organized as follows: Section 2 describes the background of the topic, Section 3 presents the methods, Section 4 explains the empirical procedure, Section 5 reports the results and Section 6 ends with the conclusion and future directions.

II. Background

The problem domain is *multi-class classification*. Multi-class classification is the process of predicting one class label from a set of available classes. Multi-class classification seeks to map inputs from a continuous space $\mathcal{X} \subseteq \mathbb{R}^d$ to categorical outputs in $\mathcal{Y} = \{1, 2, \ldots, K\}$. Given training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ with feature representations $x_i \in \mathcal{X}$ and class assignments $y_i \in \mathcal{Y}$, the task involves learning a classifier $\phi : \mathcal{X} \to \mathcal{Y}$ that generalizes to predict labels for novel inputs. Neural network approaches are powerful frameworks for approximating the mapping function and were the foundational architecture for the models that were implemented in the report.

a) Incremental Class Learning: Incremental class learning is a technique used to address class imbalance challenges in classification tasks. If a classification model is trained on the entire dataset containing class imbalances at once, the model potentially biases towards the majority classes to reduce the overall loss whilst training. The bias towards the majority classes leads to the model struggling to learn the underlying signal of the minority classes and hence does not generalize well to unseen data. The aim of incremental class learning is to expose the model to the minority classes first, which will allow the model to focus on learning the distinguishing features of the minority classes. Larger classes are added progressively, by first allowing the model to fully learn the minority classes before introducing the larger classes. The main algorithm for incremental class learning is as follows:

Algorithm 1 Incremental Class Learning

Require: Training data $\mathcal{D}$ with features X and labels Y
Ensure: A trained classifier ϕ
1: Initialize ϕ with a model capable of multi-class classification
2: **for** each class $c \in \mathcal{Y}$ **do**
3: Expose ϕ to the training samples of class c
4: Train ϕ on the current class while freezing the weights for previously learned classes
5: **end for**
6: **return** ϕ

A practical example of incremental class learning is available in the `SGDClassifier` implementation of the `scikit-learn` library. The `partial_fit` method allows the classifier to be trained incrementally on batches of data, enabling new classes to be introduced progressively during training. In the context of incremental class learning, the functionality makes it possible to expose the classifier to minority classes first, thereby strengthening its ability to capture the underlying decision boundaries for underrepresented data before the introduction of majority classes. The incremental training interface provided by `SGDClassifier` has become one of the most widely used tools for implementing online and incremental learning in practice [Pedregosa et al., 2011].

b) Related Work: A key challenge in incremental decision tree induction is that each new training observation is incorporated into the growing structure, leading to large memory requirements and slower training times. Fisher and Schlimmer [Fisher and Schlimmer, 1986] demonstrate that this is not strictly necessary. Those objects that actively influence the formation of the decision tree need to be retained. By selectively incorporating examples that alter entropy measures or decision boundaries, and discarding those that contribute no new information, the learner can substantially reduce the number of stored instances. Their analysis, supported by empirical results, shows that this reduction in object incorporation yields little to no decrease in the overall quality of the derived decision trees. In their empirical evaluation, Fisher and Schlimmer demonstrate that both the batch learner with incremental updates $\widehat{\text{ID3}}$ and its counterpart ID4 were able to achieve effective classification accuracy of approximately 90%. Remarkably, this level of

accuracy was observed after as few as 100 training instances, indicating that reliable generalization could be obtained well before the learners had processed the full dataset. The incremental version of ID4 took longer to reach the same level of accuracy and needed 750 observations to reach 90% accuracy. The cost of building ID3 was significantly larger than that of $\widehat{\text{ID3}}$ despite both having the same upper bound of computational cost of $\mathcal{O}(|I|\,|A|^2)$ where $|I|$ is the number of training instances and $|A|$ is the number of attributes.

Fisher and Schlimmer's selective incorporation idea carries directly to the neural network setting. Rather than retaining all past data, the neural network can keep a compact rehearsal buffer of informative boundary cases that change the decision boundary. The rehearsal buffer allows the model to focus on the most relevant examples, reducing memory and training cost without sacrificing generalization. The rehearsal buffer is relevant to the incremental class learning approach as it allows the model to retain knowledge of previously learned classes while learning new classes. Secondly, the implementation of the incremental class learning model in the assignment validates how the incremental learning approach is data efficient, as the model can achieve high performance with fewer training samples.

III. Methodology

The manner in which the baseline and incremental learner models were constructed, tuned and evaluated is discussed in the following section.

A. Preprocessing dataset steps

Each dataset was preprocessed to ensure that the data was in a suitable format for the models. The datasets each had unique characteristics that would highlight the strengths and weaknesses of the incremental class learning approach.

a) Iris dataset: The Iris dataset [Fisher, 1936] is a classic dataset in machine learning and statistics. It consists of a limited 150 samples from three species of iris flowers. Each sample has four features: sepal length, sepal width, petal length, and petal width. The dataset is balanced with 50 samples per class. The training data was modified to create class imbalances, where the excess points were assigned to the test set. The final class distribution for the sets is seen in Table I.

Class	Training Samples	Validation Samples	Test Samples
Iris-setosa (0)	37	8	5
Iris-versicolor (1)	21	7	22
Iris-virginica (2)	19	3	23

TABLE I: Class distribution of the modified Iris dataset

b) Fashion-MNIST dataset: The Fashion-MNIST dataset [Xiao et al., 2017] is a dataset of 70000 grayscale images of 10 different fashion items. Each image is 28x28 pixels, and the dataset is split into a training set of 60000 images and a test set of 10000 images. The dataset is balanced with 6000 images per class in the training set and 1000 images per class in the test set. The training data was modified to create class imbalances by removing samples from several classes. The training set was further separated into a validation set using 30% of the training data. The validation set was used in the training process to monitor the performance of the models and to prevent overfitting. The test set was kept separate and only used for the final evaluation of the models. The features for the Fashion-MNIST dataset are all numerical and have values between 0 and 1. The final class distribution for the training, validation, and test set is seen in Table II.

Class	Training Samples	Validation Samples	Test Samples
0	6000	1779	1000
1	5500	1640	1000
2	5000	1522	1000
3	4500	1340	1000
4	4000	1174	1000
5	3500	1064	1000
6	3000	885	1000
7	2500	763	1000
8	2000	630	1000
9	1500	453	1000

TABLE II: Class distribution of the modified Fashion-MNIST training dataset

c) Data preparation: Each training and validation dataset was separated into multiple pandas DataFrames [McKinney et al., 2020], one for each class. The DataFrames were sorted by class size in ascending order to ensure that the minority classes were exposed to the model first. The list of sorted DataFrames was then used in the training process to progressively introduce classes to the model. Each class label had to be mapped to a new label indicating the order in which the class would be introduced to the model. The mapping was done as PyTorch [Paszke et al., 2019] requires that the class labels are in the range [0, K-1] where K is the number of classes that is currently being trained. For example, in the Fashion-MNIST dataset, the class label 9 would be mapped to 0, class label 8 to 1, and so on.

d) Baseline model: The baseline model is a feedforward multilayer perceptron implemented in PyTorch, with a configurable list of hidden layers and selectable Rectified Linear Unit (ReLU) or hyperbolic tangent (Tanh) activations for the hidden layers. It produces raw logits from a final linear layer (suitable for cross-entropy loss) and enforces seeded, deterministic settings to ensure reproducible training runs.

e) Incremental class learning model: The dynamic incremental class learning model is also a feedforward multilayer perceptron implemented in PyTorch. The model starts with a single identity layer (no neurons) and one linear output layer to predict between the original two classes. The architecture of the model allows for the progressive addition of neurons to the hidden layers and to update the output layer to accommodate the new classes. Each hidden layer is limited to 512 neurons for computational reasons and as a mechanism to help combat overfitting. The dynamic addition of neurons is done by replacing the existing layer with a new layer that has the same weights as the previous layer, but with additional neurons

initialized to zero. The model adds new neurons in base-2 (powers-of-two) increments. The model keeps an internal expansion counter k and, on each expansion, adds 2^k neurons (starting with $2^0 = 1$); k is incremented after every addition. If the next expansion would exceed 512 neurons, the counter is reset to $k = 0$, a new layer is added, and the cycle restarts until the maximum number of layers is reached. The algorithm for the expansion of the model is as follows:

Algorithm 2 Dynamic Expansion of Neural Network

Require: Current model ϕ with hidden layers H, output layer O, expansion counter k

$n \leftarrow 2^k$ {Number of neurons to add}
if last(H).out_features $+ n \leq 512$ **then**
  Add n neurons to the last hidden layer in H
  $k \leftarrow k + 1$
else
  Reset $k \leftarrow 0$
  Create a new hidden layer with $2^k = 1$ neuron
  Append new hidden layer to H
  $k \leftarrow k + 1$
end if
Update output layer O to match new hidden dimension

The incremental class learning model exposes the following parameters to control the training process and help avoid under- or overfitting. The parameters are described as follows:

- **Moving average window size** — how many recent epochs that are included when smoothing the training/validation losses. Larger windows give smoother but slower-to-react curves; smaller windows react faster but can be noisy. The smoothing is used to judge trends rather than single-epoch noise.
- **Plateau threshold** — a tolerance on the absolute slope of the *smoothed* loss trends. In the implementation, the slope magnitude is treated on a 0–1 scale: values near zero mean very strict flatness, while larger values up to one would be very lenient. When both slopes are below the threshold, training is considered plateaued, and the procedure may take action.
- **Underfit threshold** — an upper bound on the *smoothed training loss*, chosen on a 0–1 scale in the implementation. Lower values closer to zero demand very low training loss before declaring the model adequately fit; higher values closer to one tolerate higher loss. For example, setting 0.5 means: if the smoothed training loss stays above 0.5 and is not meaningfully decreasing, increase complexity.
- **Overfit threshold (training)** — a requirement on how strongly the training loss must be trending downward, expressed as a normalized slope magnitude in 0–1. Smaller values near zero make the detector sensitive to even tiny decreases; larger values require a clearly negative trend before triggering. The condition, combined with the validation trend below, flags overfitting.
- **Overfit threshold (validation)** — a requirement on how strongly the validation loss must be trending upward, also on a 0–1 slope scale. Smaller values near zero react to slight increases; larger values only react to clear worsening. When training improves past its threshold while validation worsens past the threshold, the model is considered to be overfitting, and the procedure reacts.
- **Minimum number of neurons** — the least number of neurons that are added on the first expansion of the first hidden layer. A larger minimum accelerates early capacity growth; a smaller one grows the network more cautiously.
- **Learning rate** — the optimizer step size. Because the architecture can change during training by adding neurons or outputs, the optimizer is reinitialized with this rate after structural updates to maintain stable training.
- **Activation function** — the hidden-layer nonlinearity.
- **Maximum hidden layers** — the maximum number of hidden layers that can be added to the model architecture. Once the limit is reached, no further layers are added.

f) Data-efficiency protocol: To quantify data efficiency, the validation accuracy was evaluated as a function of the fraction of training data available. For each percentage, a *stratified* subset was drawn to preserve class proportions, trained with the same optimization settings, and run over a small set of random seeds from 42 to 46. The *maximum* validation accuracy was kept across those seeds at that percentage. The max-over-seeds view intentionally reflected a best-case outcome for the data available and was applied identically to the baseline and incremental learners. An acceptable validation goal threshold was chosen where the *data-efficiency point* for a model is the smallest percentage at which its best-seed accuracy first reached or exceeded the goal threshold.

B. Training procedure

The training procedure for each model was different. Both models use the Adam optimizer [Kingma and Ba, 2015] with a tuned learning rate and use cross-entropy loss. The training procedure for each model is described below.

a) Baseline model training: The model was trained for a specified number of epochs with early stopping based on the validation loss.

b) Incremental class learning model training: The model was trained for a specified number of epochs; however, unlike the baseline model, the plateau, underfit, and overfit thresholds were used to determine how the model would progress through the classes. The thresholds were thus used to determine when to stop training on the current class set and to add the next class to the set. The incremental model was trained in stages, starting with the two smallest classes and gradually introduced the remaining classes as the learning curves stabilized. At each epoch, training and validation losses/accuracies were computed, smoothed the losses with a moving-average window, and fitted a simple linear trend to the smoothed series. The slopes drive three decisions:

- **Plateau detection** — if both smoothed losses have slopes below the plateau threshold, training is considered plateaued and the next class can be added.
- **Underfitting detection** — if the smoothed training loss is above the underfit threshold and not meaningfully decreasing, the model is considered underfit and neurons are added to the last hidden layer.
- **Overfitting detection** — if the smoothed training loss is meaningfully decreasing while the smoothed validation loss is not improving, the model is considered overfit and the training for the current class set is stopped. If the model has not yet reached the minimum number of neurons, neurons are added to the last hidden layer; otherwise, the next class is added. The choice to add neurons in the seemingly overfit condition was to prevent the model from memorizing the training data and never growing in complexity as more classes are added.

The trend-based control loop prioritizes signal over per-epoch noise, letting the model grow only when needed and advance through classes when learning has saturated. Training stops early if a target validation accuracy is reached when all classes have been introduced or when the epoch budget is exhausted.

C. Comparison of models

The models were compared using the training, cross-validation, and test results using different metrics. The final confirmation of model performance was concluded using statistical tests to have reasonable confidence in the results. In the process of cross-validation, the following metrics were recorded: Training time, accuracy, precision, recall, and F1-score for the training, validation, and test sets. The metrics are defined as follows:

- Accuracy - the proportion of all examples that the model classified correctly.
- Precision - the fraction of instances predicted as positive that are truly positive.
- Recall - the fraction of true positive instances that the model successfully detected.
- F1-score - the harmonic mean of precision and recall, giving a single measure that balances both.

a) Statistical tests of models: The normality of the test F1 and accuracy distributions was determined using Shapiro-Wilk (α=0.05) to determine the appropriate statistical tests to use. The null hypothesis of the Shapiro-Wilk test states that the data is normally distributed. If the p-value is less than the significance level, the null hypothesis is rejected, and it is concluded that the data is not normally distributed. The Shapiro-Wilk test results indicate whether the metrics for both models were normally distributed. The following statistical test was used:

- Wilcoxon signed-rank test - a nonparametric test for comparing two related samples when distributional assumptions (such as normality) do not hold. The null hypothesis is that there is no difference between the models.

IV. Empirical Procedure

The empirical procedure involved the following steps: model construction with hyperparameter tuning, training and validation, and finally testing and evaluation. Each step is described in detail in the following subsections.

A. Pipeline and hyperparameter tuning

Each dataset required different preprocessing steps for the hyperparameter tuning. The method for tuning is described in this subsection.

a) Iris dataset hyperparameter tuning: The Iris dataset was small enough to directly apply the hyperparameter tuning on the full training and validation sets. The hyperparameters that were tuned included the number of hidden layers, the number of neurons per layer, learning rate, and activation function. The following configurations were used for the baseline hyperparameter tuning over 300 epochs:

- Hidden layers: [1, 2]
- Neurons per layer: [1, 2, 4, 8]
- Learning rate: [0.001, 0.005, 0.01, 0.5, 0.1]
- Activation function: [ReLU, Tanh]

The following configurations were used for the incremental class learner hyperparameter tuning over 1000 epochs:

- Moving average window size: [1, 2, 3, 5, 10, 20]
- Plateau threshold: [0.004, 0.01, 0.05, 0.1]
- Underfit threshold: [0.1, 0.2, 0.3, 0.4, 0.5]
- Overfit threshold (training): [0.1, 0.15, 0.2]
- Overfit threshold (validation): [0.1, 0.15, 0.2]
- Minimum number of neurons: [1, 2, 4, 8]
- Learning rate: [0.001, 0.005, 0.01, 0.5, 0.1]
- Activation function: [ReLU, Tanh]
- Maximum hidden layers: [1, 2, 3]

Each configuration was run three times with different random seeds; however, the results were the same across the runs due to the small size of the dataset. The best configuration was selected based on the highest validation accuracy.

b) Fashion-MNIST dataset hyperparameter tuning: The Fashion-MNIST dataset required a smaller stratified sample of the training and validation sets to perform the hyperparameter tuning. Specifically, 10% of the training and validation set was used for the Fashion-MNIST hyperparameter tuning. The baseline tuning required a minimum of 300 epochs to converge to a good solution for the sample data. The following configurations were used for the baseline hyperparameter tuning over 300 epochs:

- Hidden layers: [1, 2, 3, 4]
- Neurons per layer: [50, 64, 100, 128, 150, 200, 256, 300, 512]
- Learning rate: [0.001, 0.005, 0.5, 0.1]
- Activation function: [ReLU, Tanh]

The incremental class learner made use of the same configurations as the Iris dataset; however, the maximum number of hidden layers was increased to [1, 2, 3, 4]. Each configuration was run three times with different random seeds, starting from 42 to 44. The best configuration was selected based on the highest validation accuracy.

B. Training and validation

Each model was trained on the full training set and validation set. The training and validation losses and accuracies were recorded at each epoch to monitor the training process. The models were cross-validated over 30 independent runs with different random seeds starting from 42 and incrementing by one per run. Over each run, the training and validation sets were ensured to be stratified to maintain the class distribution. The best model was selected based on the highest validation accuracy. The final test set was only used for the final evaluation of the models.

a) Iris model training: Each Iris dataset baseline model was trained for a maximum of 150 epochs with early stopping using a patience of 10 epochs. Each incremental class learning model was trained for a maximum of 1000 epochs.

b) Fashion-MNIST model training: Each Fashion-MNIST dataset baseline model was trained for a maximum of 350 epochs with early stopping using a patience of 25 epochs. Each incremental model was trained for a maximum of 3500 epochs.

C. Metric comparison and statistical tests

The following empirical procedure was used to determine whether the observed differences in test metrics (accuracy and F1) between the baseline and incremental models are statistically significant.

1) Collect paired metric values across the repeated experiments where over 30 independent runs record the baseline and incremental values.
2) Visual inspection: produce paired boxplots of the differences to check for obvious non-normality or extreme outliers.
3) Normality check: apply the Shapiro–Wilk test to the sample of paired differences for each metric using significance level $\alpha = 0.05$. Record the test statistic and p-value.
4) Test selection:
   - If the Shapiro–Wilk p-value ≥ 0.05, assume approximate normality and perform a two-sided paired Student's t-test on the differences.
   - If the Shapiro–Wilk p-value < 0.05, reject normality and perform a two-sided Wilcoxon signed-rank test on the paired differences.
5) Report results for each metric: test statistic, p-value, and decision (reject/fail to reject H_0 at $\alpha = 0.05$).

V. Research Results

The results of the models are presented in the following section. The results include the hyperparameter tuning, training, validation, and cross-validation results, and finally, the statistical tests to determine if there was a significant difference between the two models.

A. Iris dataset hyperparameter tuning results

The top configurations were found to have a single hidden layer with few neurons, a high learning rate of 0.1, and the ReLU activation function. To keep the model simple, the configuration with 4 neurons was chosen as the final model. The configurations and their corresponding mean accuracy are shown in Table III.

TABLE III: Best Baseline Model Hyperparameter Configurations for Iris

Hidden Layers	Number of Neurons (layer 1, layer 2, ...)	Learning Rate	Activation Function	Accuracy (mean)
1	(8,)	0.10	ReLU	0.977 ± 0.000
1	(4,)	0.10	ReLU	0.977 ± 0.000
2	(1,1)	0.001	ReLU	0.622 ± 0.000
1	(2,)	0.10	ReLU	0.611 ± 0.000

The optimal hyperparameter values for the incremental class learning model can be found in Table IV.

TABLE IV: Best Incremental Class Learning Model Hyperparameter Configurations for Iris

Moving Average Window	Plateau Threshold	Underfit Threshold	Overfit (Training) Threshold	Overfit (Validation) Threshold
3	0.004	0.4	0.2	0.2
3	0.004	0.3	0.1	0.1
3	0.004	0.4	0.2	0.1
3	0.004	0.2	0.2	0.2

Min Neurons	Learning Rate	Activation Function	Accuracy (mean)
1	0.5	ReLU	0.9773 ± 0.0001
8	0.5	Tanh	0.9772 ± 0.0001
8	0.5	ReLU	0.9772 ± 0.0001
8	0.5	Tanh	0.9500 ± 0.0001

A particular point to be made from looking at the results in Table IV is the high learning rate of 0.5. The high learning rate is likely due to the small size of the dataset and the simplicity of the model. The high learning rate allows the model to converge quickly to a good solution.

B. Fashion-MNIST dataset hyperparameter tuning results

The top configurations were found to have three or four hidden layers with many neurons, a low learning rate of 0.001, and the ReLU activation function. The configurations and their corresponding mean validation accuracy is shown in Table V.

TABLE V: Best Baseline Model Hyperparameter Configurations for Fashion-MNIST

Hidden Layers	Number of Neurons (layer 1, layer 2, ...)	Learning Rate	Activation Function	Accuracy (mean)
4	(512, 256, 128, 64)	0.001	ReLU	0.8860 ± 0.0054
4	(300, 200, 100, 50)	0.001	ReLU	0.8833 ± 0.0067
3	(512, 512, 512)	0.001	ReLU	0.8817 ± 0.0075
3	(256, 128, 64)	0.001	ReLU	0.8816 ± 0.0076

The optimal parameter values for the incremental class learning model indicate that adding more complexity past three hidden layers to the model was not very beneficial, with only a minimal increase in accuracy. The simplest model was thus chosen as the final model. The optimal hyperparameter values for the incremental class learning model can be found in Table VI.

TABLE VI: Best Incremental Class Learning Model Hyperparameter Configurations for Fashion-MNIST

Moving Average Window	Plateau Threshold	Underfit Threshold	Overfit (Training) Threshold	Overfit (Validation) Threshold
10	0.001	0.5	0.1	0.15
10	0.001	0.5	0.15	0.1
10	0.001	0.5	0.15	0.15
10	0.001	0.4	0.1	0.1

Min Neurons	Learning Rate	Activation Function	Accuracy (mean)
16	0.01	ReLU	0.9374 ± 0.001
16	0.01	ReLU	0.9374 ± 0.001
16	0.01	ReLU	0.9374 ± 0.001
16	0.01	ReLU	0.9277 ± 0.002

The best-performing incremental setup on Fashion-MNIST reached a mean accuracy of 0.9374 with 16 minimum neurons, a 0.01 learning rate, and ReLU activations. The top-performing models had similar configurations for the parameters, with the focus on the balance of underfitting and overfitting thresholds to ensure that the model was not too complex or too simple.

C. Iris model training and validation results

The learning curves for the best run of the baseline and incremental class learning models are shown in Figure **??**. The difference in the two loss curves highlights the difference in learning approaches, where the baseline model's loss progressively decreases until stabilizing, whilst the incremental learner first stabilizes the loss for the first two minority classes, and when adding the next class, there is a spike in loss, which is then smoothed and stabilized. From the below learning curves, it can be seen that the incremental class learning model had sufficiently reduced the training and validation losses to a point where the model was not underfitting or overfitting. Although the final loss appears to be quite low for both training and validation, keep in mind that the dataset is small and thus has limited exposure to all the variability in the data.

The cross-validation and test results provide a better indication of the model's performance and can be seen in Table VII. The results below indicate that the baseline model outperformed the incremental class learning model on all metrics. The incremental class learning model struggled with the small dataset and was unable to learn the relationships in the data. The incremental model's underwhelming performance is likely due to the small size of the dataset. The baseline model performed better than the incremental class learning model, as it was able to consider the entire dataset at once and thus could learn the more nuanced relationships between classes better. The incremental class learning model was at a disadvantage as it had to learn the classes progressively and thus had limited exposure to the data at any given time. Furthermore, since the validation set was small, the model would not be able to compute reliable trends to determine when to add complexity or new classes. The incremental model evidently struggled to improve its performance as it consistently predicted the same incorrect outputs as seen in the lack of variance in the results.

TABLE VII: Mean ± std. dev. for Baseline and Incremental models. Percentage metrics shown as mean ± std (percent); time shown in seconds.

Metric	Baseline (Mean ± Std)	Incremental (Mean ± Std)
Accuracy (Test)	96.21% ± 2.17%	44.07% ± 0.00%
F1 (Test)	96.87% ± 1.78%	20.39% ± 0.00%
Precision (Test)	97.16% ± 1.73%	14.69% ± 0.00%
Recall (Test)	96.79% ± 1.72%	33.33% ± 0.00%
Training Time (s)	0.376 ± 0.008 s	0.271 ± 0.004 s
Accuracy (Train)	98.58% ± 2.05%	32.00% ± 0.00%
F1 (Train)	96.58% ± 2.07%	13.01% ± 0.00%
Precision (Train)	96.64% ± 2.01%	8.08% ± 0.00%
Recall (Train)	96.58% ± 2.12%	33.33% ± 0.00%
Accuracy (Validation)	98.00% ± 2.52%	32.00% ± 0.00%
F1 (Validation)	98.12% ± 2.38%	16.16% ± 0.00%
Precision (Validation)	98.35% ± 2.05%	10.67% ± 0.00%
Recall (Validation)	98.14% ± 2.37%	33.33% ± 0.00%

Note: Percentage metrics: means and standard deviations computed from raw decimals, then multiplied by 100 and shown with two decimal places. Time reported in seconds.

D. Fashion-MNIST training and validation results

Fortunately, the Fashion-MNIST dataset was large enough for the incremental class learning model to learn the relationships in the data. The results are thus more promising than that of the Iris dataset. The learning curves for the best run of the baseline and incremental class learning models are shown in Figure **??**. The baseline model's loss curves in Figure **??** show a typical training pattern where the training loss decreases and stabilizes, while the validation loss decreases and stabilizes at a higher loss. The incremental class learning model's loss curves in Figure **??** show a more complex pattern due to the dynamic nature of the model. The loss curves show spikes in loss, indicative of either a new class being added to the training and validation set or a new hidden layer being added. Due to the strict thresholds for overfitting and underfitting, the loss curves for the dynamic model seemingly lie on top of each other. The scale of the fluctuations is much smaller compared to the baseline model, where the difference in training and validation loss is not as pronounced. To better visualize the

results of the training of the dynamic Fashion-MNIST model, refer to the accuracies obtained as seen in Figure **??**.

A particular point of interest is the first 100 epochs of Figure **??**, which highlight the initial learning dynamics. The validation accuracy oscillates between extremes, achieving incredibly high and low accuracies. The oscillation is due to the following: The model initially being simple memorizes the training data, achieves high training accuracy, and low validation accuracy. The model training process recognizes that result as overfitting, thus first tries to add complexity if the minimum number of neurons was not reached. The model then adds complexity and improves the validation performance, and achieves very high validation results as the data is still small. The model then adds a new class when a sufficiently low validation loss is reached. The process then repeats itself. The training only begins to stabilize when the model is complex enough to start learning relationships instead of memorizing the data.

The results of the cross validation and test results for the Fashion-MNIST dataset were more promising than that of the Iris dataset. The results can be found in Table VIII. As is seen from the results in Table VIII, the incremental class learning model outperformed the baseline model on most metrics; however, the test metrics were very similar. The incremental learner had more stable performance as the standard deviations were lower across all metrics. A particular point of interest is the training times, where the incremental learner took significantly longer to train than the baseline model. The trade-off between the two models is between performance and training time, as the incremental class learning model took significantly longer to tune, train, and validate; however, it achieved more stable performance.

TABLE VIII: Mean $\pm$ std. dev. for Baseline and Incremental models. Percentage metrics shown as mean $\pm$ std (percent); time shown in seconds.

Metric	Baseline (Mean $\pm$ Std)	Incremental (Mean $\pm$ Std)
Accuracy (Test)	83.59% $\pm$ 7.04%	83.77% $\pm$ 5.74%
F1 (Test)	82.96% $\pm$ 8.18%	83.73% $\pm$ 6.33%
Precision (Test)	83.87% $\pm$ 7.70%	83.54% $\pm$ 7.45%
Recall (Test)	83.55% $\pm$ 6.93%	84.35% $\pm$ 4.43%
Training Time (s)	9.65 $\pm$ 1.88 s	115.77 $\pm$ 12.70 s
Accuracy (Train)	88.88% $\pm$ 9.79%	93.94% $\pm$ 8.26%
F1 (Train)	88.53% $\pm$ 10.66%	94.29% $\pm$ 7.93%
Precision (Train)	89.72% $\pm$ 9.43%	93.84% $\pm$ 9.27%
Recall (Train)	89.00% $\pm$ 9.57%	95.25% $\pm$ 5.36%
Accuracy (Validation)	86.89% $\pm$ 8.81%	90.21% $\pm$ 7.36%
F1 (Validation)	86.31% $\pm$ 9.79%	90.29% $\pm$ 7.42%
Precision (Validation)	87.24% $\pm$ 9.12%	89.98% $\pm$ 8.71%
Recall (Validation)	86.76% $\pm$ 8.66%	91.07% $\pm$ 5.09%

Note: Percentage metrics: means and standard deviations computed from raw decimals, then multiplied by 100 and shown with two decimal places. Time reported in seconds.

The test metrics were very similar, thus requires further analysis to determine if the models were statistically different. The box plots in Figure **??** show the distributions of the test accuracy and F1 metrics across the 30 cross-validation runs. The box plot was used for the primary metrics of interest, accuracy and F1, as they provide a good visual representation of the distribution of the metrics. Both boxplot figures show the same outlier performance in the incremental model. The baseline model had a higher median performance across both metrics and higher maximum values. The baseline performance is skewed to the left, where the distribution of the values is gathered at the higher scores. The results suggest that the baseline model is generally, a stronger performer than the incremental model; however, the baseline model had more variability in its performance. The statistical tests will confirm whether there is any statistical difference between the two models.

a) Per-class breakdown (Fashion-MNIST test): The per-class breakdown of the test results will better expose how the model performed on the minority classes on a particular run. The class-level F1s in Table IX show that the dynamic model's main gain appears on Class 3, rather than exclusively on the minority classes in the training distribution. In other words, the incremental learner unfortunately did not simply "boost the small classes".

TABLE IX: Per-class F1 on Fashion-MNIST test. Delta is (Dynamic $-$ Baseline). The largest improvement is on Class 3.

Class	F1 (Baseline)	F1 (Dynamic)	Δ
0	0.93	0.94	+0.01
1	0.95	0.95	+0.00
2	0.92	0.94	+0.02
3	**0.62**	**0.65**	**+0.03**
4	0.94	0.95	+0.01
5	0.77	0.77	+0.00
6	0.87	0.86	$-$0.01
7	0.77	0.78	+0.01
8	0.97	0.97	+0.00
9	0.81	0.80	$-$0.01

b) Data-efficiency findings (Fashion-MNIST): Using random seeds 42 to 46 and a target validation accuracy of 92%, the incremental learner reached the threshold with a smaller fraction of the training data than the baseline network. In other words, as the data budget increased, the incremental curve crossed 0.92 earlier, whereas the baseline required a higher percentage to meet the same criterion. The behaviour matched the staged, trend-based training. By focusing capacity on the early classes and expanding only when losses plateau or change, the incremental model learns useful patterns sooner. The data efficiency graph can be seen in Figure **??**. Over the four random seeds, both models dipped in performance at 50% of the training data. The dip is likely due to the particular samples chosen for that percentage, as the samples were stratified, but not guaranteed to be representative of the full dataset. The incremental model reached the 92% target at 40% of the training data, whereas the baseline model required 60% of the training data to reach the same target in the best run. The incremental model was thus more data efficient than the baseline model.

c) Statistical Significance Testing: Shapiro-Wilk tests were used to assess the normality of the paired differences

between the two models. The results for the tests were:

- Shapiro-Wilk p-value for accuracy difference equals 0.0195; Shapiro-Wilk p-value for F1 difference equals 0.0204.
- Both p-values are less than 0.05, so the null hypothesis of normality is rejected for both differences.

Hypotheses (for each metric): H_0: no difference between models; H_1: difference between models.

Because normality was rejected, Wilcoxon signed-rank tests were used:

TABLE X: Normality and Wilcoxon test results

Test (metric)	Statistic	p-value
Shapiro-Wilk (Accuracy diff)	–	0.0195 (reject normality)
Shapiro-Wilk (F1 diff)	–	0.0204 (reject normality)
Wilcoxon (Accuracy)	215.0000	0.7188 (fail to reject H_0)
Wilcoxon (F1)	228.0000	0.9262 (fail to reject H_0)

Both paired differences deviate from normality, so nonparametric Wilcoxon tests were used, and in both cases fail to reject the null hypothesis, as no significant difference was detected between the models for accuracy or F1.

d) Discussion: The statistical tests indicate that there was no statistically significant difference between the two models for both accuracy and F1 metrics. Therefore, in the case of the Fashion-MNIST dataset, the incremental class learning model does not provide a statistically significant improvement over the baseline model in pure evaluation metrics. However, despite the lack of statistical significance, the incremental class learning model showed major improvement on the Fashion-MNIST dataset as compared to the Iris dataset. The poor performance on the Iris dataset was due to the small size of the dataset, where inter-class relationships could not be learned effectively. The architecture of the model favoured more aggressive learning, where the model would quickly add complexity to learn the relationships in the data. The aggressive learning was not suitable for the small Iris dataset, where the model would quickly overfit to the small amount of data. The incremental class learning model was more suited to the larger Fashion-MNIST dataset, where the model could learn the relationships in the data better. The incremental class learning model was more stable, with lower standard deviations across all metrics. The improvement in stability was surprising as the incremental class learning model had more hyperparameters to tune, which would typically lead to more variability in the results.

The incremental class learning model was also more data efficient, reaching a target validation accuracy of 92% with only 40% of the training data, whereas the baseline model required 60% of the training data to reach the same target for the Fashion-MNIST dataset. The incremental model was not able to show significant improvement on the minority classes, where the largest improvement was on Class 3, which was not a minority class. Despite the lack of significant improvement on the minority classes, the ability of the model to be more data efficient is beneficial in scenarios where data is imbalanced, limited, or expensive to obtain. In such scenarios, the imbalanced data problem is often coupled with limited data, where the incremental class learning model could be beneficial.

Overall, the incremental class learning model shows promise as a learning approach if data efficiency and stability are priorities. The challenge with the dynamic model was the hyperparameter tuning, where the model had many more parameters to tune than the baseline. Secondly, the dynamic model required significantly more time and resources to train and validate, as the model took over 12 times longer to train than the baseline model. However, due to the flexibility of the approach, the model could be adapted to different datasets and learning scenarios.

VI. Conclusion

The intention of the report was to determine the benefit of making use of incremental learning in neural networks to combat imbalanced datasets. Two datasets were used to evaluate the performance of the incremental class learning model against a baseline model. The Iris dataset was too small for the incremental class learning model to be effective, where the baseline model outperformed it on all metrics. The Fashion-MNIST dataset was more promising, where the incremental class learning model achieved higher scores on all metrics. Despite these metrics being higher, the models did not have a statistically significant difference in the accuracy and F1 scores. Despite the lack of statistical significance, the incremental class learning model was more data efficient, reaching a target validation accuracy of 92% with only 40% of the training data, whereas the baseline model required 60% of the training data to reach the same target for the Fashion-MNIST dataset. The incremental class learning model was also more stable, with lower standard deviations across all metrics. The trade-off between the two models is between performance and training time. The incremental class learning model took significantly longer to tune, train, and validate; however, it was able to achieve acceptable performance on fewer data observations than the baseline model.

Further work could involve testing the models on larger and more complex datasets to better evaluate the performance of the incremental class learning model. The incremental class learning model for the assignment was limited due to computational constraints, where the model could not be made too complex. With more computational resources, the model could be made deeper and wider to better learn the relationships in the data. The associated code for all the calculations [1] can be found in the footnote and the references on the last page.

[1]The code for this report is available on GitHub: https://github.com/grantxxcoder/25849646rw441Assignment3.

APPENDIX

LIST OF ACRONYMS

ReLU	Rectified Linear Unit
Tanh	Hyperbolic Tangent
H_0	Null Hypothesis
H_1	Alternative Hypothesis

REFERENCES

[Fisher and Schlimmer, 1986] Fisher, D. H. and Schlimmer, J. C. (1986). A case study of incremental concept induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI)*, page incremental learning section. AAAI Press.

[Fisher, 1936] Fisher, R. A. (1936). Iris [dataset]. UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/iris, accessed 2025-09-26.

[Kingma and Ba, 2015] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

[McKinney et al., 2020] McKinney, W., Reback, J. F., Tratner, J., Augspurger, T., Cloud, P., Hawkins, S., gfyoung, Sinhrks, Klein, A., Petersen, T., She, C., Ayd, W., Garcia, M., Schendel, J., Hayden, A., Saxton, D., Dong, K., Overmeire, W., and Wang, P. (2020). pandas-dev/pandas: Pandas.

[Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.