

Neural Networks Training and Testing: Hyperparameter Experimentation and Loss Curve Diagnosis in Text Classification through TensorFlow and Keras

Jomari Manto
College of Computer Studies & Engineering
José Rizal University
Mandaluyong City, Philippines
jomari.manto@my.jru.edu

Abstract—This report documents the rebuilding of a three-class customer feedback classifier in TensorFlow and Keras, following an earlier exercise performed on a no-code platform whose best model reached 62.5 per cent accuracy and a weighted F1-score of 0.617. Twenty-one runs were logged. The Keras baseline scored 0.370 on the original 257 examples and 0.496 after the corpus was expanded to 4,032 balanced reviews. Varying learning rate, hidden dimension and dropout across the three pillars of hyperparameters produced no improvement on that figure. Performance moved only when preprocessing was relaxed: raising the sequence length from twenty tokens to one hundred and fifty added no parameters yet improved weighted F1 by 0.180, more than every hyperparameter change combined. The champion model scored 0.722 on a held-out partition of 605 reviews, exceeding the benchmark by 9.6 percentage points. Sixty-six per cent of residual error falls in one class pair, indicating a labelling rather than a modelling limit.

Keywords—neural network training, hyperparameters, loss curves, overfitting, text classification, sequence truncation, Keras

I. INTRODUCTION AND PT-P1 BASELINE PERFORMANCE

Software vendors receive customer feedback in volumes that make manual triage impractical, and separating that feedback by intent allows each type to be routed to the people able to act on it. Accounting software is a useful case because the consequences of misrouting are concrete. A user unable to run payroll on a filing deadline has a different problem from one who dislikes the interface, though both may write at length and in similar language. A previous exercise built a classifier for this task on the Machine Learning for Kids platform, separating Intuit QuickBooks feedback into praise, complaints and urgent support requests.

That exercise established what tagged examples alone could achieve, but it could not explain how. The platform exposes a single training action and reports only a predicted class with a confidence percentage. Its architecture, loss function, optimiser and epoch count are not visible, and the report accompanying it inferred the position of feature extraction in the pipeline from observed behaviour rather than from inspection. There was also no held-out partition: every tagged example was used for training, and evaluation relied on twelve short messages written by the author. The distinction between a model that had learned and one that had memorised could therefore not be drawn at all.

The present exercise rebuilds the same classifier in Python using TensorFlow and Keras. The change is not merely one of implementation. Writing the pipeline exposes the internal

weights and biases that the network adapts automatically, termed model parameters, and separates them from the external configurations set by the operator before training begins, termed hyperparameters [1]. The latter divide into three pillars. Architecture comprises the embedding and hidden dimensions. Optimisation comprises the learning rate, batch size, epoch count and optimiser. Regularisation comprises the dropout rate. Each of these was fixed and invisible on the no-code platform. Each is now an experimental variable.

A. Establishing the Benchmark

The exercise brief requires the code-based model to exceed the no-code baseline on accuracy and F1-score. Establishing that baseline required computation, because the platform reported neither metric directly. Only per-item predicted classes and confidence percentages were recorded, so accuracy, precision, recall and F1 were derived afterwards from the twelve stored predictions.

Three training rounds were completed in the earlier exercise. Table I gives the derived figures across the eight scored items. The remaining four test inputs belonged to none of the three classes and are excluded, since no correct answer exists against which to score them.

TABLE I. PT-P1 PERFORMANCE, DERIVED FROM PER-ITEM PREDICTIONS

Round	Accuracy	Precision	Recall	F1
1	62.5%	0.635	0.625	0.617
2	37.5%	0.156	0.375	0.205
3	62.5%	0.391	0.625	0.481
Mean	54.2%	0.394	0.542	0.434

^a Precision, recall and F1 are weighted averages across the three classes, computed on the eight scored test items.

The table exposes something the earlier report could not state. Rounds 1 and 3 are indistinguishable on accuracy, both scoring 62.5 per cent, and the earlier report noted only that the same figure arose from a different distribution of errors. On weighted F1 they separate sharply, 0.617 against 0.481, because the third round returned Negative_Complaint for eleven of twelve inputs and scored zero on two of the three classes. Accuracy concealed a collapse that F1 makes visible. This is itself an argument for the metric the present exercise adopts.

Round 1 was therefore selected as the formal benchmark. Beating a three-round mean depressed by a round the earlier report documented as a degradation would be a weaker claim than beating the strongest result the platform achieved, and the

standard deviation across rounds, 14.4 percentage points of accuracy, makes the mean an unstable target in any case. Both figures are recorded in the experiment log for transparency.

One limitation must be stated plainly. The benchmark rests on eight scored items, so a single item is worth 12.5 percentage points of accuracy, and class support is uneven at five, two and one. The Keras runs reported below are scored on 605 held-out reviews. The two figures are therefore not like-for-like, and the code-based evaluation is by a considerable margin the harder test.

II. EXPERIMENTAL METHODOLOGY AND HYPERPARAMETER CONFIGURATIONS

A. Dataset

The corpus is drawn from a publicly released collection of Amazon product reviews covering the software category, made available for academic use by the McAuley Lab at the University of California San Diego. Reviews mentioning Intuit QuickBooks, Quicken or Intuit by name were selected and short entries excluded, so that the material retained the length and irregularity of genuine customer writing.

Two datasets were prepared. The first reproduces the 257 examples used to train the final no-code model, allowing a strict like-for-like comparison at the outset. The second contains 4,032 reviews, balanced at 1,344 per class, and excludes every review used previously. That figure is a ceiling rather than a choice. Of the reviews remaining after exclusion, 11,375 fell into `Negative_Complaint` and 3,874 into `Positive_Praise`, but only 1,344 into `Urgent_Support`, and the brief requires balance to be maintained when data is added. Duplicate reviews were removed across the corpus before sampling, to prevent identical text appearing in more than one partition.

Class labels were assigned by heuristic in the earlier exercise and were not manually verified. A five-star rating produced `Positive_Praise`, one or two stars produced `Negative_Complaint`, and a low rating combined with crisis language produced `Urgent_Support`. The third class is therefore carved from the same pool as the second and shares most of its vocabulary by construction. The earlier report predicted that these two classes would confuse in any confusion matrix. Section V returns to this point with measurements.

The corpus is long-form. Reviews average 175 words, with a median of 122, a minimum of 36 and a maximum of 3,449. This distribution becomes significant once the pipeline is described.

B. Pipeline

The notebook supplied for the exercise implements a six-stage pipeline. Section 1 loads the corpus, builds a vocabulary, encodes each review as a fixed-length integer sequence and partitions the data. Section 2 declares the hyperparameter configuration. Section 3 constructs the network. Section 4 selects an optimiser and compiles the model. Section 5 executes training and evaluation. Section 6 plots the loss curves.

Tokenisation is minimal. Text is lowercased and split on whitespace, with no punctuation handling, stopword removal or lemmatisation. Words are ranked by corpus frequency and

the most common are assigned indices from two upward, with index zero reserved for padding and index one for tokens outside the vocabulary. Two constants govern this stage. `MAX_LEN` fixes the sequence length, truncating longer reviews from the end and padding shorter ones. `MAX_VOCAB_SIZE` caps the vocabulary, collapsing every rarer word into a single shared index.

Both constants were supplied at values of 20 and 1000 respectively. On the fifteen-row sample dataset distributed with the notebook these are inert: rows contain 11 to 15 words and the corpus holds 147 distinct tokens, so neither limit is ever reached. On the corpus used here both bind severely. Every review exceeds 20 words, including the shortest, so 100 per cent of rows are truncated and approximately 89 per cent of all tokens are discarded before training begins. The vocabulary cap admits fewer than 1,000 of roughly 34,000 distinct tokens. This observation was recorded before any experiment was run, and shaped the free-form design described in Section III.

The network stacks an embedding layer, a global average pooling layer, one dense hidden layer with rectified linear activation, a dropout layer and a three-unit output layer with softmax activation. Loss is sparse categorical cross-entropy, matching the integer labels. The pooling layer averages token embeddings across the sequence, producing a fixed-length representation independent of review length but discarding word order entirely.

The data is partitioned 70/15/15 by two successive random splits, yielding 2,822 training, 605 validation and 605 test rows on the full corpus. The split is not stratified, so partition class counts deviate slightly from the corpus balance. The vocabulary is fitted before the split rather than on the training partition alone, which allows validation and test vocabulary to enter the feature space. Both are properties of the supplied design and were retained unchanged so that runs remained comparable.

C. Reproducibility

The no-code platform was deterministic without intervention. Repeating the first round in a separate project on a different day reproduced all twelve results exactly, including confidence percentages. Keras is not. Weight initialisation, dropout masks and batch shuffling all draw from a pseudorandom generator, so determinism must be imposed rather than assumed.

Seeds were fixed at 42 for NumPy, TensorFlow and the data split. Because `tf.random.set_seed` sets a global seed that advances with use, every experiment was executed by restarting the runtime and running all cells in order, guaranteeing identical initial conditions. This was not merely precautionary. An early run in which Section 5 was executed twice without rebuilding the model in Section 3 continued training the existing weights rather than starting afresh, producing a result attributable to neither configuration. The first epoch of a correctly initialised run reports a training loss near 1.0986, the cross-entropy of uniform guessing across three classes, and this value was used thereafter as a verification check.

Two runs of identical configuration, EXP-05B and EXP-05C, produced weighted F1 scores of 0.6965 and 0.7066. That difference of approximately 0.01 establishes the run-to-run

noise band, and differences smaller than this are not treated as meaningful anywhere in this report.

D. Experimental Design

Twenty-one runs were logged. The first five follow the configurations specified in the brief. The remainder are free-form. Metrics reported are final-epoch training and validation loss and accuracy, together with weighted precision, recall and F1 computed on the validation partition. Weighted averaging is used throughout, matching the brief's requirement and accounting for the uneven class support that arises from the unstratified split.

The held-out test partition was not used by any experiment. It was scored once, after the champion had been selected, so that the final figure remains an unbiased estimate of generalisation.

Tables II and III give the configuration of every run. They are split by pillar for legibility: Table II covers preprocessing and architecture, Table III optimisation and regularisation. Reading the two together shows how little of the space was explored before EXP-05. Runs EXP-01 to EXP-04 hold the preprocessing constants fixed at the supplied values of 20 and 1,000 and vary only the pillars, which is the design the brief specifies. From EXP-05 onward the preprocessing constants become variables, and the parameter counts in Section III show why that distinction matters.

TABLE II. ARCHITECTURE AND PREPROCESSING CONFIGURATIONS

Run	Len	Vocab	Emb	Lyr	Units
01-BC	20	1k	32	1	64
01-FULL	20	1k	32	1	64
02	20	1k	32	1	16
02A	20	1k	32	1	64
03	20	1k	32	1	256
04	20	1k	32	1	128
05	150	1k	32	1	128
05A	150	8k	32	1	128
05B	250	8k	32	1	128
05C	250	8k	32	1	128
05D	250	8k	32	1	128
05E	250	8k	32	1	128
05F	250	8k	32	1	128
05G	250	8k	32	2	128-64
05H	250	8k	32	2	128-64
05I	250	8k	32	2	128-64
05J	250	8k	32	2	128-64
05K	250	8k	32	2	128-64

^b. Run identifiers abbreviate the EXP- prefix. Len is MAX_LEN, Vocab is MAX_VOCAB_SIZE, Emb is embed_dim, Lyr is the number of hidden layers and Units is hidden_dim. Run 02A is a supplementary control isolating hidden_dim; the two test evaluations reproduce 05C and 05F and are omitted.

TABLE III. OPTIMISATION AND REGULARISATION CONFIGURATIONS

Run	LR	Optimiser	Drop	Ep	Batch
01-BC	0.005	Adam	0.1	15	32
01-FULL	0.005	Adam	0.1	15	32
02	0.0001	Adam	0.0	15	32
02A	0.0001	Adam	0.0	15	32
03	0.05	Adam	0.0	15	32
04	0.005	Adam	0.4	15	32
05	0.005	Adam	0.4	15	32
05A	0.005	Adam	0.4	15	32
05B	0.005	Adam	0.4	15	32
05C	0.005	Adam	0.4	15	32
05D	0.005	Adam	0.4	4	32

05E	0.005	Adam	0.4	2	32
05F	0.005	Adam	0.4	3	32
05G	0.005	Adam	0.4	3	32
05H	0.05	SGD	0.4	15	32
05I	0.005	Adam	0.4	3	64
05J	0.005	AdamW	0.4	15	32
05K	0.05	SGD-M	0.4	15	32

^c. LR is learning_rate, Drop is dropout_rate and Ep is the epoch count. AdamW carries a weight decay of 0.01 and SGD-M a momentum of 0.9. Every run used sparse categorical cross-entropy loss and a random seed of 42.

The first five runs follow the brief. EXP-01 establishes the Keras baseline at the supplied defaults, EXP-02 induces underfitting through low capacity and a very small learning rate, EXP-03 induces overfitting through high capacity and no regularisation, and EXP-04 pairs high capacity with substantial dropout. EXP-05 onward is free-form. Those later runs fall into three groups: preprocessing changes at EXP-05 to EXP-05C, an epoch sweep at EXP-05D to EXP-05F, and single-parameter tests of depth, optimiser and batch size at EXP-05G to EXP-05K.

III. PERFORMANCE MONITORING AND LOSS CURVE ANALYSIS

Table IV gives the validation results for every run, ordered as executed. Table V gives the fit diagnosis assigned to each from its loss curve, together with the train/validation loss gap and the epoch at which validation loss reached its minimum. The two are separated because the diagnosis rests on the shape of the curve across epochs rather than on the final figures alone.

TABLE IV. VALIDATION RESULTS FOR ALL RUNS

Run	Tr loss	Val loss	Tr acc	Val acc	F1
01-BC	0.0116	2.0931	100.0	35.9	0.370
01-FULL	0.4858	2.4397	79.2	50.1	0.496
02	1.0447	1.0701	55.2	44.6	0.423
02A	1.0049	1.0467	51.7	43.1	0.345
03	0.7584	2.4193	58.6	41.8	0.334
04	0.4424	2.9322	78.5	47.8	0.469
05	0.3558	1.1353	86.6	64.6	0.649
05A	0.0006	1.9041	100.0	68.6	0.686
05B	0.0006	1.8361	100.0	69.4	0.697
05C	0.0007	1.6780	100.0	70.4	0.707
05D	0.1948	0.8824	91.8	69.6	0.695
05E	0.5600	0.7592	72.6	63.5	0.632
05F	0.3396	0.8607	85.9	68.4	0.684
05G	0.3920	0.8674	85.0	68.1	0.683
05H	1.0987	1.1001	34.3	32.1	0.156
05I	0.5527	1.3097	68.8	56.7	0.536
05J	0.0025	2.4645	100.0	68.8	0.691
05K	1.1009	1.1009	33.6	32.1	0.156

^d. Accuracies are percentages and F1 is the weighted average across the three classes, both computed on the validation partition. The PT-P1 benchmark for comparison is 62.5 per cent accuracy and 0.617 weighted F1.

TABLE V. LOSS CURVE DIAGNOSIS

Run	Gap	Best ep	Diagnosis
01-BC	2.082	3	Overfitting
01-FULL	1.954	1	Overfitting
02	0.025	15	Underfitting
02A	0.042	15	Underfitting
03	1.661	1	Overfitting, unstable
04	2.490	1	Overfitting
05	0.780	2	Overfitting
05A	1.904	2	Overfitting
05B	1.836	2	Overfitting
05C	1.677	2	Overfitting
05D	0.688	2	Good fit, mild overfit
05E	0.199	1	Good fit

05F	0.613	2	Good fit
05G	0.475	1	Good fit
05H	0.001	11	Failed to train
05I	0.757	1	Overfitting
05J	2.462	1	Overfitting
05K	0.000	6	Failed to train

^c Gap is final validation loss minus final training loss. Best ep is the epoch at which validation loss reached its minimum. Failed to train denotes a run whose loss never departed from 1.0986, the cross-entropy of uniform guessing across three classes; the brief's three categories do not cover this outcome.

A. The Prescribed Runs

EXP-01 was executed twice, first on the 257 examples used by the no-code model and then on the full corpus, with no other change. The first scored a weighted F1 of 0.370 and the second 0.496. Both diagnose as overfitting, but the character of the failure differs. On 257 rows the model reached 100 per cent training accuracy while validation accuracy never left the 33.3 per cent chance level for three balanced classes, so nothing transferable was learned at any point. On 4,032 rows validation accuracy reached 50.1 per cent, so the model learned before it memorised. Total parameters were 34,307 in both cases, giving 192 parameters per training example on the small corpus and 12 on the large one. Figure 1 shows the second of these runs.

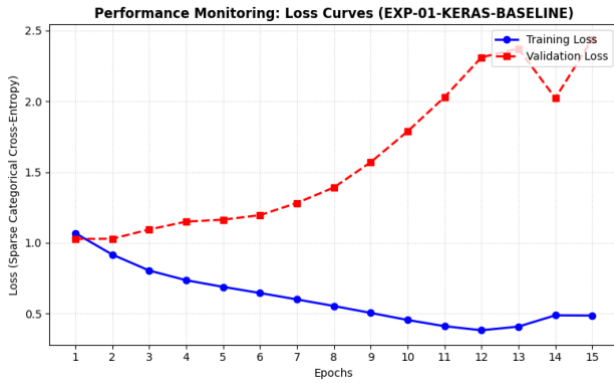


Fig. 1. EXP-01 on the full corpus. Training loss falls steadily while validation loss rises from epoch 2 onward, the classic overfitting signature.

EXP-02 induced underfitting as intended, and produced the most instructive contrast in the series. Its validation loss of 1.0701 is less than half EXP-01's 2.4397, yet its accuracy and F1 are both worse. A model that learns little remains appropriately uncertain, whereas one that memorises becomes confidently wrong, and cross-entropy penalises the second far more heavily than the first. Loss and accuracy must therefore be read together. Figure 2 shows both curves high and still descending at the final epoch, with a gap of only 0.025: the model ran out of time rather than capacity.

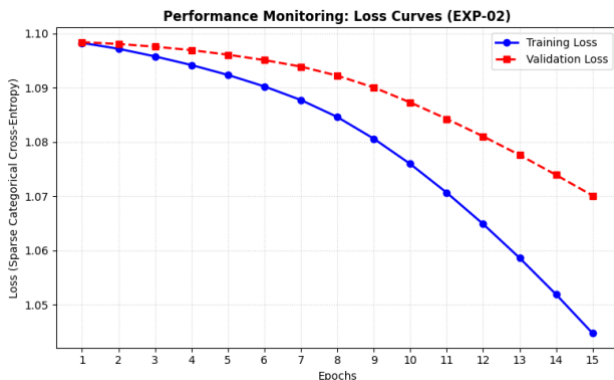


Fig. 2. EXP-02. Both losses remain high and are still falling at epoch 15, with a negligible gap between them.

A supplementary run, EXP-02A, repeated EXP-02 with hidden_dim at 64 rather than 16 and everything else identical. The smaller model performed better, 0.423 against 0.345 on weighted F1, while total parameters differed by only five per cent. This is the first indication that hidden_dim is not the binding constraint on this architecture, because the embedding table holds the overwhelming majority of the weights regardless of what the dense layer is set to.

EXP-03 did not produce the expected result. It was configured to induce overfitting through high capacity and no regularisation, and the train/validation gap of 1.661 is indeed large. The mechanism, however, was optimisation instability rather than memorisation. Training loss did not descend monotonically but oscillated, and training accuracy peaked at 67.3 per cent in epoch 7 before falling back to 58.6 per cent. A model that memorises never becomes worse on its own training data. Validation loss swung by more than a full unit between adjacent epochs. At a learning rate of 0.05 the optimiser overshoots the minimum and never settles. Despite holding 41,219 parameters, this run memorised less than EXP-01 managed with 34,307, because the extra capacity was unusable. Figure 3 shows a shape that matches none of the three categories in the exercise brief.

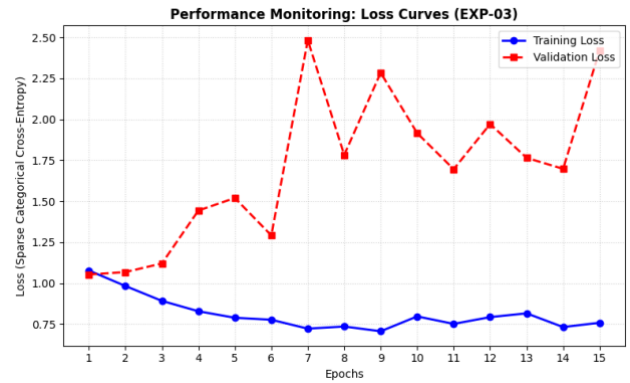


Fig. 3. EXP-03. Neither curve descends smoothly. The oscillation indicates a learning rate too high for the optimiser to converge.

EXP-04 returned the learning rate to 0.005 and raised dropout to 0.4. Stability recovered completely, which confirms that EXP-03's oscillation was caused by the learning rate and not by the capacity. Dropout, however, did not prevent overfitting. Minimum validation loss still occurred at epoch 1, final validation loss was higher than EXP-01's at 2.9322, and training accuracy was essentially unchanged at 78.5 per cent. The reason is structural. The dropout layer sits after the dense layer and therefore regularises 128 neurons, while 32,000 of the model's 36,611 parameters, eighty-seven per cent, live in the embedding table upstream of it. Most of the memorisation capacity is never touched.

At this point none of EXP-02, EXP-03 or EXP-04 had improved on the EXP-01 baseline of 0.496. Hyperparameter tuning alone had not closed the gap to the benchmark, which is precisely the condition the third discussion question anticipates.

B. Preprocessing

EXP-05 changed one variable from EXP-04: MAX_LEN was raised from 20 to 150. Weighted F1 rose from 0.469 to 0.649, an increase of 0.180 that is larger than every hyperparameter change in the series combined. Total parameters did not change at all, remaining at 36,611. Not one weight was added; the model simply received more of the text it already had. Figure 4 shows the result, and the y-axis scale is worth noting against Figure 1: validation loss now begins near 0.76 rather than 1.10.

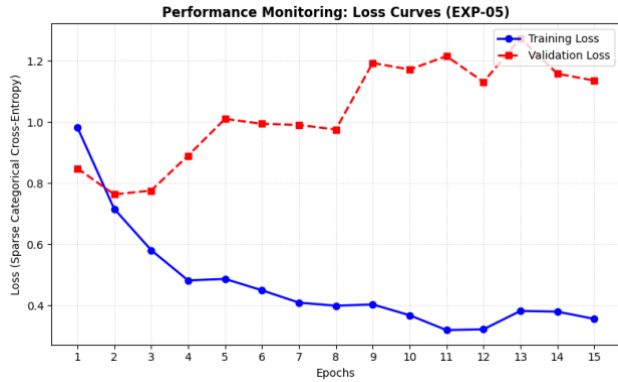


Fig. 4. EXP-05, identical to EXP-04 except that MAX_LEN is 150 rather than 20. Weighted F1 improves by 0.180 at no parameter cost.

EXP-05A raised MAX_VOCAB_SIZE from 1,000 to 8,000, improving weighted F1 to 0.686. The gain was real but modest against its cost: total parameters rose sevenfold to 260,611, with 256,000 in the embedding table alone. Comparing the two preprocessing levers directly, sequence length bought 0.180 of weighted F1 for no additional parameters while vocabulary size bought 0.038 for a sevenfold increase in model size. The cheaper change was worth roughly five times the more expensive one. EXP-05B and EXP-05C raised MAX_LEN further to 250, adding 0.010 and confirming that sequence length was by then close to exhausted against a corpus mean of 175 words.

EXP-05B and EXP-05C are identical in configuration and scored 0.697 and 0.707. That difference of approximately 0.01 arises from weight initialisation and batch shuffling alone and defines the run-to-run noise band used throughout this report.

C. The Epoch Sweep

Every run to this point reached its minimum validation loss at epoch 1 or 2 and then diverged for the remaining thirteen or fourteen epochs. EXP-05D, EXP-05E and EXP-05F therefore held the champion configuration fixed and varied only the epoch count, at four, two and three respectively. Table VI summarises the result.

TABLE VI. EFFECT OF EPOCH COUNT AT FIXED CONFIGURATION

Epochs	Val loss	F1	Gap	Diagnosis
2	0.7592	0.632	0.199	Good fit
3	0.8607	0.684	0.613	Good fit
4	0.8824	0.695	0.688	Mild overfit
15	1.6780	0.707	1.677	Overfitting

^f All four runs share MAX_LEN 250, MAX_VOCAB_SIZE 8,000, hidden_dim 128, learning rate 0.005 and dropout 0.4. Epoch count is the only variable.

One variable produces all three fit categories. At two epochs the training loss of 0.5600 sits below the validation loss of 0.7592, the curves have not diverged and the gap of 0.199

is the smallest recorded, but weighted F1 falls to 0.632. At fifteen epochs F1 is highest at 0.707, but training accuracy reaches 100 per cent, training loss falls to 0.0007 and validation loss more than doubles from its minimum. Three epochs is the best compromise, holding 97 per cent of the champion's F1 at roughly half its validation loss and a third of its gap. Figures 5 and 6 place the two extremes side by side.



Fig. 5. EXP-05F at three epochs. Both losses descend together and the gap remains small.

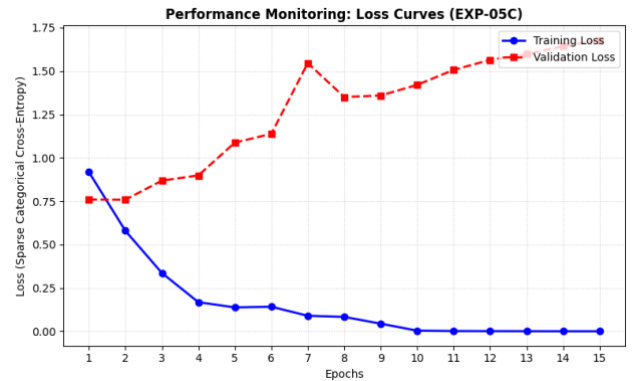


Fig. 6. EXP-05C at fifteen epochs, the champion on weighted F1. Training loss approaches zero while validation loss more than doubles from its epoch-2 minimum.

D. Single-Parameter Tests

Four further runs each varied one parameter that had not previously moved. EXP-05G added a second hidden layer of 64 neurons, giving an architecture of 128-64. Weighted F1 moved from 0.684 to 0.683, a difference of 0.001 and roughly an eighth of the noise band. Depth changed the model by less than re-running the same configuration twice. The reason is twofold: the second layer adds 8,256 parameters to a model of 268,675, and by the time data reaches the dense stack it is a single 32-number average of the whole review with word order already discarded by the pooling layer, leaving little hierarchical structure for a further layer to find.

EXP-05I raised batch size from 32 to 64 and weighted F1 fell from 0.683 to 0.536. This is a training-volume effect rather than a property of batch size itself. At batch 32 an epoch contains 89 weight updates and at batch 64 it contains 45, so three epochs delivered 135 updates against 267. Since the best runs deliberately stop at two or three epochs, halving the update count under-trains the model. A strictly fair comparison would give batch 64 six epochs to match the

update count, and this is recorded as a confound rather than a finding.

EXP-05H and EXP-05K tested the optimiser. Stochastic gradient descent was given deliberately favourable settings relative to Adam, a learning rate of 0.05 and fifteen epochs, and did not train at all: training loss held at 1.0987 across every epoch and never departed from 1.0986, the cross-entropy of uniform guessing. Validation accuracy was frozen at 32.1 per cent throughout, and a weighted F1 of 0.156 with precision of 0.103 indicates a single class predicted for all 605 validation rows. EXP-05K added momentum of 0.9 to test whether the failure was caused by the absence of acceleration. It made no difference whatsoever, producing identical figures. Adam at a learning rate ten times lower trains the same architecture to 70 per cent accuracy in three epochs, so the failure is structural rather than a matter of step size.

EXP-05J tested AdamW, whose decoupled weight decay acts on every parameter including the embedding table and therefore addresses the limitation identified in EXP-04. At a decay of 0.01 it did not solve the problem. Training accuracy still reached 99.96 per cent and validation loss still climbed to 2.4645. Across dropout at 0.4 and weight decay at 0.01, no regularisation mechanism tested prevented memorisation; limiting the epoch count was the only effective control.

E. Held-Out Test Evaluation

The champion was selected by comparing validation scores across twenty-one runs, so the validation figure is no longer an unbiased estimate of generalisation. The test partition of 605 reviews had not been used by any experiment and was scored once, after selection. Table VII reports the outcome, together with the same evaluation applied to EXP-05F for comparison.

TABLE VII. HELD-OUT TEST RESULTS

Model	Val F1	Test acc	Test F1
PT-P1 benchmark	n/a	62.5	0.617
EXP-05C, 15 ep	0.697	72.1	0.722
EXP-05F, 3 ep	0.657	68.9	0.683

[‡] Test accuracy is a percentage on 605 held-out reviews. The PT-P1 figures are computed on eight author-written test inputs and are not like-for-like.

The champion scores 72.1 per cent accuracy and 0.722 weighted F1 on the test partition, exceeding the benchmark by 9.6 percentage points and 0.105 respectively. The test figures are higher than the validation figures, which is the reassuring direction and indicates that selection across twenty-one runs did not materially bias the validation estimate.

The second row of Table VII overturns an expectation. EXP-05F is the better-fitted model by every diagnostic measure, with less than half the validation loss and a train/validation gap a third the size, yet it loses on the test set by 3.2 percentage points of accuracy and 0.039 of weighted F1, both well outside the noise band. The per-class report locates the loss precisely: recall on Urgent_Support falls from 0.68 to 0.43 while its precision rises from 0.75 to 0.84. At three epochs the model has become cautious, labelling a review Urgent_Support only when highly confident and missing 110 of 212 genuine cases. Three epochs is not sufficient to learn the harder of the two overlapping classes, so the additional twelve epochs in the champion are not purely memorisation. A favourable loss curve does not by itself identify the better classifier.

The test confusion matrix locates the residual error unambiguously. Of 185 true Negative_Complaint reviews, 40 were predicted Urgent_Support, and of 212 true Urgent_Support reviews, 61 were predicted Negative_Complaint. Those 101 items are 66 per cent of the model's 165 test errors. Per-class F1 was 0.84 for Positive_Praise, 0.71 for Urgent_Support and 0.60 for Negative_Complaint, and Positive_Praise barely confuses with either. This is the outcome the earlier report predicted from the labelling scheme rather than from any property of the network, and Section V returns to its implications.

IV. DISCUSSION

A. Code-Based Optimisation versus No-Code Heuristics

The two exercises differ in what they hide, and the difference is not simply that one is more capable than the other. The no-code platform presented a single training action. Its architecture, loss function, optimiser and epoch count were fixed and invisible, and the earlier report was reduced to inferring where feature extraction occurred from the behaviour of the finished model. Keras exposes all of it. The consequence is not that better results become available but that responsibility transfers to the operator, and that transfer has a cost as well as a benefit.

The clearest benefit is that some settings are not refinements at all but preconditions. EXP-05H and EXP-05K show this starkly. Stochastic gradient descent, given a learning rate ten times higher than Adam's and five times the epochs, never departed from a training loss of 1.0986 and predicted a single class for all 605 validation rows. Adding momentum of 0.9 changed nothing. Adam at a learning rate ten times lower trains the same architecture to 70 per cent accuracy in three epochs. Had the no-code platform used a non-adaptive optimiser internally, there would have been no way to observe the fact and no mechanism to correct it. Explicit control turned an invisible constraint into a one-line change.

A second benefit is that diagnosis became possible at all. The earlier exercise had no held-out partition, so there was only ever one curve and no way in principle to distinguish a model that had learned from one that had memorised. It could describe the collapse of its second round to 37.5 per cent accuracy but could not name the mechanism. With two curves the distinction is immediate, and Section III assigns a diagnosis to every run.

Against these, the costs were substantial and are worth stating plainly. The first is that the search is difficult to steer without prior experience. Twenty-one runs were required to locate a constraint that turned out to lie in preprocessing rather than in any of the three pillars, and for much of that sequence there was no reliable way to judge in advance whether a given change was worth attempting. An educated guess that fails still costs a full run, and the absence of any means of verifying the instinct beforehand is itself a cost of the code-based approach. The no-code platform offered fewer choices and therefore fewer opportunities to spend effort in the wrong direction.

The second cost is that the parameters are not independent, and the log shows repeatedly that correcting one constraint disturbs others. Raising MAX_LEN from 20 to 150 recovered the truncated text but gave the model more signal to memorise, so the overfitting became more pronounced rather than less. Raising MAX_VOCAB_SIZE to 8,000 admitted the missing

vocabulary but increased total parameters sevenfold, moving 98 per cent of the model's weights into the embedding table where the dropout layer cannot reach them. Raising `batch_size` from 32 to 64 halved the number of weight updates per epoch, so a setting chosen to smooth the gradient silently invalidated the epoch count that had been tuned around it. Most pointedly, reducing the epoch count from fifteen to three halved the validation loss and produced a favourable fit diagnosis, but recall on `Urgent_Support` fell from 0.68 to 0.43 on the test partition. The loss curve was corrected and the classifier was made worse.

The third cost is that the direction of improvement is not monotonic. EXP-02A outperformed EXP-02 with a quarter of the hidden units. EXP-03, holding 41,219 parameters, memorised less than EXP-01 managed with 34,307, because its learning rate prevented the additional capacity from being used at all. More capacity, more data and more training are each helpful in some regions of the space and harmful in others, and there is no reliable way to predict which region a configuration occupies without running it. This is precisely why a structured experiment log, rather than intuition, is the instrument that makes the search tractable.

Determinism illustrates the transfer of responsibility in miniature. The no-code platform reproduced twelve results exactly across separate projects on different days without any action by the operator. Keras required seeds to be fixed for three separate generators, and even then an early run was silently invalidated by executing the training cell twice without rebuilding the model, producing thirty epochs of training recorded as fifteen. The property that came free in the first exercise had to be engineered and verified in the second.

The trade-off is therefore not one of capability but of where the difficulty is placed. The no-code tool constrains the search space to the training data alone, which is limiting when the constraint lies elsewhere and efficient when it does not. The code-based pipeline removes that constraint and replaces it with a larger space, interacting parameters and no guidance about where to look. In this exercise the constraint did lie inside the pipeline, at `MAX_LEN`, and only the code-based approach could have found it. That does not make the search cheap.

B. Diagnosing Model Fit via Loss Curves and Regularisation

The intended target throughout was straightforward: both curves descending together and settling close to one another. Recognising departures from that pattern proved simple. Obtaining it did not, because the corrections available act on different parts of the model and interact with one another.

Underfitting appeared as two high, nearly parallel curves still descending at the final epoch. EXP-02 ended with a training loss of 1.0447 against a validation loss of 1.0701, a gap of 0.025, both close to 1.0986, the cross-entropy of uniform guessing across three classes. The decisive detail is not the height of the curves but their direction. Minimum validation loss occurred at epoch 15, the last one, so the model was still improving when training stopped and had run out of time rather than capacity.

Overfitting appeared as divergence following an early minimum. EXP-01 on the full corpus reached its minimum validation loss at epoch 1 and rose from 1.0275 to 2.4397 over the remaining fourteen epochs, while training loss fell to

0.4858. The gap of 1.954 is the diagnostic quantity, but the epoch of the minimum is what makes the diagnosis actionable, and in almost every run in this log that minimum fell at epoch 1 or 2. Thirteen or fourteen of the fifteen epochs were therefore actively harming generalisation.

A third signature appeared that the three categories in the exercise brief do not cover. EXP-03 was configured to induce overfitting and produced a large gap of 1.661, but its training loss did not descend monotonically. It oscillated, and training accuracy peaked at 67.3 per cent in epoch 7 before falling back to 58.6 per cent. A model that memorises never becomes worse on data it has already seen. Validation loss swung by more than a full unit between adjacent epochs. This is optimisation instability arising from a learning rate of 0.05, not memorisation, and the distinction matters because the remedy differs: EXP-04 restored stability by lowering the learning rate alone, without altering capacity or regularisation.

Against these signatures, both regularisation mechanisms tested proved ineffective, for a reason that is structural rather than a matter of degree. Dropout did not resolve overfitting. EXP-04 raised the rate to 0.4, silencing nearly half the hidden neurons on every batch, yet minimum validation loss still occurred at epoch 1, final validation loss was worse than EXP-01's at 2.9322, and training accuracy was essentially unchanged at 78.5 per cent against 79.2. The dropout layer sits after the dense layer and therefore regularises 128 neurons, while 32,000 of the model's 36,611 parameters, some 87 per cent, reside in the embedding table upstream of it. The overwhelming majority of the memorisation capacity is never touched. This became more pronounced once vocabulary was expanded. At 260,611 parameters the embedding accounts for 98 per cent of the model, and EXP-05A reached 100 per cent training accuracy at the same dropout rate. EXP-05J tested AdamW, whose decoupled weight decay does act on the embedding table, and at a decay of 0.01 it likewise failed, reaching 99.96 per cent training accuracy.

Adjusting `hidden_dim` did not resolve the issue either, and in one case the smaller model was the better one. The parameter was tested at 16, 64, 128 and 256. EXP-02 and EXP-02A form a controlled pair at identical settings differing only in `hidden_dim`, and the 16-neuron model outscored the 64-neuron one, 0.423 against 0.345 on weighted F1, while total parameters differed by five per cent. EXP-03 at 256 units held 41,219 parameters yet memorised less than EXP-01 managed with 34,307. Adding a second hidden layer in EXP-05G moved weighted F1 by 0.001, roughly an eighth of the run-to-run noise band.

What did resolve overfitting was limiting the epoch count. Holding the champion configuration fixed and varying only the number of epochs produced all three fit categories, as Table VI shows: a gap of 0.199 at two epochs, 0.613 at three, 0.688 at four and 1.677 at fifteen. Neither pillar-based mechanism could prevent a model from memorising 2,822 examples. Stopping before it had the opportunity was the only effective control.

The wider pattern across the log is that the standard remedy for each diagnosis behaved inconsistently. Increasing capacity is the prescribed remedy for underfitting, yet EXP-02A with 64 hidden units scored below EXP-02 with 16. Reducing capacity or increasing regularisation is the prescribed remedy for overfitting, yet dropout at 0.4 and weight decay at 0.01 both left training accuracy above 99 per

cent. The one intervention that worked, limiting the epoch count, is not a pillar parameter at all, and it carried its own penalty in the form of reduced recall on the hardest class. Diagnosing the fit from the curves was therefore considerably easier than acting on the diagnosis.

C. Data Volume and Diversity versus Architectural Capacity

Hyperparameter tuning alone did not outperform the baseline. EXP-02, EXP-03 and EXP-04 varied all three pillars and none improved on the EXP-01 figure of 0.496 weighted F1. The dataset was then expanded and the preprocessing constraints relaxed, and the exercise ended with three separate measurements of what the word data can mean. The binding constraint moved between them.

Volume was the first, and it worked. EXP-01 was executed twice with identical code and identical hyperparameters, differing only in the corpus: 257 examples against 4,032. Weighted F1 rose from 0.370 to 0.496 and accuracy from 35.9 to 50.1 per cent. Total parameters were unchanged at 34,307, because MAX_VOCAB_SIZE capped the embedding table on both corpora, so the ratio of parameters to training examples fell from 192 to 12. The character of the failure changed as well as its magnitude. On 257 rows validation accuracy never departed from the 33.3 per cent chance level for three balanced classes, so nothing transferable was learned at any point; on 4,032 rows the model learned before it memorised. This is the clearest illustration in the log of why capacity must be balanced against volume. The architecture was identical in both runs. Only the quantity of evidence available to fit 34,307 weights differed, and at 179 training rows there was insufficient evidence to fit them to anything but the training set itself.

Sequence length was the second, and it was worth considerably more. EXP-05 raised MAX_LEN from 20 to 150 with every other setting matching EXP-04, and weighted F1 rose from 0.469 to 0.649. That single change is larger than every hyperparameter adjustment in the series combined, and it added no parameters whatsoever: the model remained at 36,611 weights before and after. The reason is that the supplied constant was fitted to a different corpus. On the fifteen-row sample distributed with the notebook, rows contain 11 to 15 words and MAX_LEN of 20 never activates. On a corpus averaging 175 words per review it truncates every row without exception and discards approximately 89 per cent of all tokens. Four runs of pillar tuning were therefore applied to a model reading roughly a ninth of its own input. Diversity in this instance was not a matter of collecting more examples but of not discarding the ones already held.

Vocabulary was the third, and it demonstrates the cost side of the same principle. EXP-05A raised MAX_VOCAB_SIZE from 1,000 to 8,000, improving weighted F1 by 0.038 while increasing total parameters sevenfold to 260,611. Set against sequence length, which bought 0.180 for nothing, the cheaper intervention was worth roughly five times the more expensive one. The additional capacity was not merely poor value: it was actively harmful to the fit, because 256,000 of those parameters sit in the embedding table upstream of the dropout layer, and EXP-05A reached 100 per cent training accuracy by epoch 10. Adding capacity to represent more of the data made the model harder to regularise without making it proportionally better.

None of these three levers, however, explains where the exercise stopped. The champion reached 72.1 per cent accuracy on the held-out partition and no configuration tested moved materially beyond it. The test confusion matrix identifies the reason. Of 185 true Negative_Complaint reviews, 40 were predicted Urgent_Support, and of 212 true Urgent_Support reviews, 61 were predicted Negative_Complaint. Those 101 items are 66 per cent of the model's 165 test errors. Per-class F1 was 0.84 for Positive_Praise, 0.71 for Urgent_Support and 0.60 for Negative_Complaint, and Positive_Praise barely confuses with either of the other two. The error is not distributed across the problem; it is concentrated almost entirely in one class pair.

That pair is an artefact of how the labels were produced. Class assignment in the earlier exercise was heuristic and unverified: five stars produced Positive_Praise, one or two stars produced Negative_Complaint, and a low rating combined with crisis-language keywords produced Urgent_Support. The third class is therefore carved from the same low-rated pool as the second, and separated from it only by the presence of particular words. Two reviews describing the same failure receive different labels according to whether a keyword happens to appear. The dataset is balanced in count, at 1,344 examples per class, but the classes are not independently separable, and a model asked to reproduce that boundary is being asked to reproduce an arbitrary rule rather than to learn a distinction present in the text.

This is what representative diversity means in practice, and it is a different property from either volume or balance. A model cannot exceed the consistency of its labels. If the two classes were perfectly separable the same predictions would yield an accuracy near 89 per cent, so the remaining error is not a modelling deficiency to be tuned away. It explains why twenty-one runs across four optimisers, four values of hidden_dim, two network depths, four epoch counts and two preprocessing constants all plateaued in the same region. The earlier report predicted this outcome from the labelling scheme alone, before any Keras model existed, and the confusion matrix confirms it. Raising performance further would require relabelling rather than retuning, and that is a data collection task rather than a neural network one.

V. PIPELINE MAPPING

The training loop described in the lecture material comprises five stages: the forward pass, loss calculation, the backward pass, the weight update and epoch iteration [1]. In the notebook these stages are not written out. All five execute inside the single call to model.fit in Section 5, which is the operational loop. What the earlier sections do is configure them. Table VIII gives the mapping.

TABLE VIII. MAPPING OF TRAINING STAGES TO NOTEBOOK SECTIONS

Stage	Configured in	Executes in
Forward pass	Section 3, layer stack	Section 5, model.fit
Loss calculation	Section 4, loss argument	Section 5, model.fit
Backward pass	automatic differentiation	Section 5, model.fit
Weight update	Section 4, optimizer	Section 5, model.fit
Epoch iteration	Section 2, epochs and batch	Section 5, model.fit

The forward pass is defined by the sequence of model.add calls in Section 3, which establish the order in which a batch traverses the network. The loss function is supplied in Section

4 as the loss argument to `model.compile`, set to sparse categorical cross-entropy to match integer class labels. The backward pass is not configured anywhere: TensorFlow constructs the computational graph as the forward pass executes and differentiates it automatically, so no gradient expression appears in the notebook. The weight update is governed by the optimiser object built in Section 4, which the conditional statement selects according to `CONFIG`. Epoch iteration is controlled by the `epochs` and `batch_size` values declared in Section 2 and passed to `model.fit`.

One consequence of this arrangement deserves note. Because the four stages are encapsulated within a single call, the model retains its state between invocations. Executing Section 5 a second time without rebuilding the model in Section 3 continues training the existing weights rather than beginning afresh, which occurred during this exercise and produced thirty epochs of training recorded as fifteen. The layer stack in Section 3 begins with a call to `Sequential`, and that call is the only point at which weights are reinitialised.

A. Automated Feature Extraction

Feature extraction occurs in the first two layers of Section 3, before any classification takes place, and requires no action by the operator. Section 1 has already reduced each review to a fixed-length sequence of integers, but those integers carry no meaning. They are frequency ranks: the most common token in the corpus receives index two, the next receives three, and so on, with index zero reserved for padding and index one for tokens outside the vocabulary. The index is a label, not a measurement, and the arithmetic distance between two indices signifies nothing.

The Embedding layer converts each of these indices into a vector of `embed_dim` real numbers, producing a matrix of shape 250 by 32 for a single review under the champion configuration. These vectors are initialised randomly and are model parameters in the sense defined in Section I: they are adapted by backpropagation during training exactly as the dense weights are. Nothing specifies in advance what any dimension represents. Words appearing in similar contexts across the training examples are drawn toward similar regions of the space because that arrangement reduces the loss, and the representation is therefore learned from the labelled data rather than designed. This layer holds 256,000 of the champion model's 260,611 parameters, so the great majority of what the network learns is this representation rather than the classification rule applied to it.

The `GlobalAveragePooling1D` layer reduces that matrix to a single vector of 32 numbers by taking the mean of each dimension across the sequence. Because the Embedding layer is constructed with `mask_zero` set to true, padding positions are excluded from the mean, so a review of 40 real tokens is averaged over 40 values rather than 250. This layer holds no parameters at all and learns nothing. Its function is to produce a fixed-length document representation from a variable-length one, which the dense layers require.

The operation discards word order entirely, and the consequence is worth stating because it bounds what the architecture can express. Two reviews containing the same words in any arrangement produce identical representations. Sarcasm, negation and qualification are all carried by word order, and the earlier exercise identified sarcastic phrasing as the single hardest case for the no-code classifier. A mean-

pooled architecture cannot in principle recover that information. It also explains the null result from EXP-05G: adding a second dense layer offers no benefit when the input to the dense stack is already a single averaged vector with no internal structure remaining for further layers to decompose.

Only after these two layers does classification begin. The first Dense layer applies a learned weighting to the 32 pooled features and passes the result through a rectified linear activation, the Dropout layer randomly suppresses a fraction of those units during training only, and the final Dense layer of three units with softmax activation converts the result into three probabilities summing to one. That final vector is the same quantity the no-code platform reported as a confidence percentage. In the earlier exercise its origin could only be inferred from behaviour; here it is the output of a named layer, three lines from the end of Section 3.

VI. CONCLUSION

Reflecting on the exercise as a whole, the outcome worth carrying forward is not the final score but where the performance turned out to be hiding. The exercise is organised around the three pillars of hyperparameters, and a reasonable expectation on beginning it is that careful work within those pillars is what separates a weak model from a strong one. That expectation was not borne out here.

Every pillar was varied thoroughly. Four values of hidden dimension, two network depths, three learning rates, four optimisers and three dropout rates were tested, and no configuration among them improved on the untuned baseline. The improvement came instead from two constants sitting outside the hyperparameter configuration entirely, in the preprocessing stage, one of which had been left at a value fitted to a fifteen-row sample dataset. The consequence was that a network reading roughly a ninth of each review was being carefully tuned. That this was the situation could not be inferred from the loss curves, which looked like ordinary overfitting throughout, nor from the metrics, which improved plausibly with each sensible change. It became visible only from comparing the corpus statistics against the supplied constants, which is an inspection of the data rather than of the model.

The experience of conducting the search was more laborious than anticipated, and the difficulty was not the running of experiments but the choosing of them. Each run costs the same whether the hypothesis behind it is good or poor, and at this level of experience there was no dependable way to tell the difference beforehand. The parameters compound this, because they are not independent. Recovering the truncated text made the overfitting worse. Admitting the missing vocabulary moved almost all the model's weights beyond the reach of its own regularisation layer. Enlarging the batch halved the number of weight updates and quietly invalidated an epoch count that had been chosen around the previous value. Reducing the epochs produced a textbook loss curve and a worse classifier. The direction of improvement was not consistent either: a smaller hidden layer beat a larger one, and a model with more parameters memorised less than a model with fewer. Twenty-one runs of this is the reason a structured log exists, and the reason it was the only reliable instrument available.

One artefact-level improvement follows directly. The notebook preserves the six numbered sections of the supplied

design but carries almost no explanatory annotation, so the reasoning behind a given configuration lives in the log rather than beside the code. Adding markdown documentation to each section, while retaining that sectional structure, would make the pipeline substantially easier to follow and to hand on.

A supplementary experiment placed the labour in perspective. A separate notebook performed an automated random search over the same pipeline, sampling forty configurations from a space of approximately 39.8 million combinations spanning preprocessing, all three pillars, vocabulary fitting policy and split stratification. It completed in under six minutes. Its best configuration scored 0.735 weighted F1 on the test partition against the manual champion's 0.722, though it scored lower on validation, 0.691 against 0.707, so the two are better described as comparable than as ranked. More interesting than the scores was what the search confirmed without being told: it selected the same preprocessing values arrived at manually, ranked hidden dimension and network depth among the least influential of the fourteen parameters searched, and concentrated 66 per cent of its own test errors in the same class pair.

Two conclusions follow, and they pull in opposite directions. Forty configurations were evaluated in the time a handful of manual runs would have taken, and reaching that number by hand at the present level of experience would have taken considerably longer than the twenty-one runs already completed. That is a genuine argument for the tooling. Against it, the search performed well only because the space it was given already contained the two preprocessing constants as variables, and they were included solely because the manual work had established beforehand that this was where the performance lay. A search confined to the three pillars, which is the conventional framing and the one this exercise opens with, would have explored precisely the flat region documented in EXP-02 to EXP-04 and returned a configuration near 0.50 as optimal, in six minutes, with no indication that anything was wrong.

The lesson taken from the exercise is therefore that automation reduces the labour of the search without reducing the judgement required to aim it. The operator still decides which parameters are candidates, what ranges are plausible, which metric matters and how large a difference must be before it means anything rather than reflecting run-to-run variation. Those choices set the direction, and an automated procedure follows a poorly chosen direction as efficiently as a well chosen one. The instinct to aim it correctly is not something the tooling provides. On the evidence of this exercise it is built through exactly the slow, logged and occasionally misdirected experimentation that the exercise demanded, and further practice of that kind is what would most improve the result next time.

ACKNOWLEDGMENT

The author thanks Dr. Rodolfo C. Raga Jr. for the design of this exercise and for the baseline notebook on which the pipeline is built. The review data used for training was made available for academic use by the McAuley Lab, University of California San Diego.

REFERENCES

- [1] R. C. Raga Jr., "Lecture 4: Neural network training and testing," *ITC 611*, José Rizal University, Mandaluyong City, Philippines, 2026.