Water Quality Model Analysis Report

A deep learning project to predict water potability using neural networks. Built by Group 7 as part of the Introduction to Machine Learning course.

GitHub Repository:

Link to the GitHub

Project Structure

```
├── data/
│   └── water_potability.csv
├── notebooks/
│   ├── excel_asaph_copy_of_formative_II_starter_code_.ipynb
│   ├── NICOLAS_Copy_Of_Formative_II_Starter_Code_.ipynb
│   ├── keza_joan_of_formative_ii_starter_code.ipynb
│   ├── nicolle_marizani_formative_II_starter_code_.ipynb
│   └── John_Ongeri_Ouma_formative_II_starter_code_.ipynb
├── report/
│   ├── gitlog.txt
│   ├── report.md
│   └── report.pdf
└── README.md
```

Training Results Summary

Train Instance	Engineer Name	Regularizer	Optimizer	Early Stopping	Dropout Rate	Accuracy	F1 Score	Recall	Precision
1	Excel	L1 (0.01)	Adam (lr=0.001)	val_loss, patience=5	0.25	0.7060	0.675	0.701	0.692
2	Nicolle	L2	RMSProp	val_accuracy	0.35	0.654	0.280	0.172	0.750
3	John Ongeri	L1 (0.005)	Adam	patience=3, min_delta=0.001	0.3→0.2→0.1	0.928	0.939	0.929	0.948
4	Joan Keza	L1(0.001)	Adagrad	val_accuracy, patience=10	0.3	0.667	0.39	0.6217	0.5535
5	Nicolas Muhigi	L2 (0.002)	Nadam	patience=6, min_delta=0.002	0.5	0.743	0.717	0.702	0.735

Individual Analysis Reports

Excel Asaph (Member 1)

Model-Building

When I started building the neural network for the water quality classification task, I knew that just throwing layers together wouldn't do it. I wanted to understand why certain choices work better than others, so I experimented with different architectures, regularization methods, and optimizers to find what balanced learning well without overfitting.

My Chosen Architecture & Why

- **Layers**: I used **three hidden layers**: The first one with **32** neurons, the second with **64**, and the last one with **128**. This was inspired by a general rule of thumb to start with a smaller first layer and increase gradually.
- **Activation**: I used **ReLU** (Rectified Linear Unit) because it's known for helping deep models learn faster and avoid the *vanishing gradient problem*.

Why not more layers or more neurons?

Initially, I tried adding a fourth layer and more neurons (like 256-128-64-32), but I found the model started overfitting quickly. It performed great on training but worse on validation. So I simplified it, and surprisingly, performance got more stable.

Preventing Overfitting: Regularization and Dropout

What Didn't Work:

When I first trained the model without **regularization or dropout**, I noticed:

- The training accuracy kept rising till I got a constant accuracy of `1.0000`
- The validation accuracy plateaued or even dropped to around `0.6340`; my validation loss was around `5.1220`, which was way higher than my training loss `0.09514`.
- The model was learning too much from the training data — *classic overfitting*.

What Worked:

- **L1 Regularization (lambda = 0.01)**: I used `l1` because I wanted the model to learn sparse weights, essentially helping it ignore non-important features like `Organic_carbon` and `Trihalomethanes` which are not direct predictive of our target, `Potability`. It gently pushed unimportant weights toward zero, improving generalization.
- **Dropout (rate = 0.25)**: I added a **dropout layer** after the last hidden layer. This meant that during training, `25%` of the neurons were randomly *"turned off."* It seemed counterintuitive at first, but it made the model more robust.

Lesson: Without these two, my model overfit easily. With them, validation accuracy improved and stayed close to training accuracy, which was a good sign.

Optimization Choices: Learning Rate, Early Stopping & Batch Size

- **Optimizer**: I chose `Adam (learning rate = 0.001)`. It adapts the learning rate as training progresses, which is perfect when you don't want to micromanage training too much.
    - When I tried `SGD`, learning was too slow, and it got stuck at poor local minima.
- **Early Stopping (patience = 5)**: I didn't want the model to keep training unnecessarily once performance stopped improving as was the case earlier. Early stopping helped prevent overfitting, and **restoring the best weights** gave me the most balanced performance.
- **Batch Size = 128**: This was a sweet spot for me. Smaller batch sizes **(like 32)** led to noisy updates, while larger ones **(256+)** slowed down convergence. **128** gave fast, stable training.

Results Summary

- **Validation Accuracy**: `67%`
- **Test Accuracy**: `71%`
- **F1 Score**: `0.675`
- **Precision**: `0.692`
- **Recall**: `0.701`

Overall, I was happy with the performance, especially considering the limited size and slight imbalance in the dataset.

What I Learned

- Tuning is **key**. It wasn't always obvious what would work. I had to try and fail a few times.
- **Simple is often better**. A compact architecture with dropout and regularization outperformed larger, complex ones.
- **Don't train forever**. Early stopping saved me a lot of time and helped avoid the trap of "maybe one more epoch." larger numbers of epochs doesn't always mean the model will train better.
- **Visuals helped**. Plotting loss and accuracy curves after each run gave me quick feedback on what was working and what wasn't.

Challenges I Faced

- **Balancing Bias and Variance**: When I added too much regularization, the model underfit. When I removed it, overfitting returned. It was a delicate situation.
- **Finding the right lambda**: I tried different values (0.1, 0.01, 0.001) and found 0.01 struck a balance. It was enough to regularize without hurting learning.
- **Dealing with missing values and class imbalance**: I used interpolation and forward/backward fill techniques. If I skipped this, the model couldn't even train. Also, there was an imbalace in our classes as there were `0 = 1998` and `1 = 1278`.

Comparative Analysis with Teammates

When I compared my model's performance with my teammates', I discovered some fascinating insights about how different approaches to the same problem can lead to such varying results. Let me break down what I observed:

Comparison with John's Model

John's model's performance drw my attention, and I had to analyse why:

The Numbers:

- My accuracy: 71% vs John's 92.8%
- My F1 score: 0.675 vs John's 0.939
- My recall: 0.701 vs John's 0.929

What I Could Have Done Better:

1. **Regularization Strategy**:
    - I used a stronger L1 (0.01) thinking it would help with feature selection
    - John's milder L1 (0.005) proved more effective and it removed noise without being too aggressive
    - **Lesson learned**: Sometimes less is more with regularization
2. **Dropout**:
    - My static 0.25 dropout was too simplistic
    - John's approach (0.3→0.2→0.1) was brilliant as it adapted to each layer's needs
    - **Key insight I got from this**: Layer-specific dropout rates make more sense architecturally

Comparison with Nicolas's Model

Nicolas took a different approach that yielded some interesting results too:

The Numbers:

- Accuracy: My 71% vs Nicolas's 74.3%
- F1 score: My 0.675 vs Nicolas's 0.717
- Recall: My 0.701 vs Nicolas's 0.702

What I Could Have Done Better:

1. **Choice of Regularization**:
    - My L1 regularization was focused on sparsity while Nicolas's L2 (0.002) provided smoother weight decay

- **What I realized**: L2 might have been better for this dataset's noise patterns

2. **Optimizer Selection**:

- My Adam optimizer was solid but basic for this dataset
- Nicolas's Nadam added Nesterov momentum, giving better convergence
- **Future note to self**: Consider momentum-based optimizers for similar tasks like this one

Conclusion & Next Steps

This project gave me a much clearer view of how to approach neural network design beyond just building models that "run." I learned how **each hyperparameter decision shapes the model's behavior**, and how small changes can make a big difference in the real world.

Next time, I'd love to:

- Try **Keras Tuner** for automated hyperparameter tuning.
- Use **confusion matrix-based thresholding** to boost precision or recall for underperforming classes.
- Experiment with other optimizers like **RMSprop or Nadam**, and learning rates.

Nicolle Marizani (Member 2)

Parameter Justification

Parameter	Value	Reason
Regularizer	l2(0.001)	Prevents overfitting via weight penalty
Dropout Rate	0.35	Reduces co-adaptation and overfitting
Optimizer	RMSprop	Adapts learning rate; better for noisy gradients
Learning Rate	0.0005	Stable, slow learning is suited for generalization
Loss Function	binary_crossentropy	Standard for binary classification
EarlyStopping Monitor	val_accuracy	Focuses on performance on unseen data
EarlyStopping Patience	7	Avoids early termination due to noise
EarlyStopping min_delta	0.01	Requires significant improvement to continue
Final Activation	sigmoid	Outputs a probability for binary classification

Challenges Faced

1. **Class Imbalance**:

- Dataset has more non-potable samples
- Led to low recall and F1-score for potable class (1)

2. **High Dropout Rate (0.35)**:

- Prevents overfitting but may hinder learning
- Slow convergence, especially on small datasets

3. **Low Learning Rate with RMSprop**:

- Training is stable but slow
- May prevent optimal performance before early stopping

4. **Long Training Time**:

- High epoch limit (4000) increases computational load

- Early stopping helps mitigate this issue

5. **Limited Interpretability**:
   - Neural networks less transparent than simpler models
   - Difficult to explain predictions

Training Comparison Summary

Member	Regularizer	Optimizer	Learning Rate	Dropout	Early Stopping	Accuracy	F1 (Class 1)
Nicolle	L2 (0.001)	RMSprop	0.0005	0.35	val_accuracy, patience=7	0.65	0.28
Joan	L1 (0.001)	Adagrad	0.01	0.30	val_accuracy, patience=10	0.58	0.45
Nicolas	L2 (0.002)	Nadam	0.001	0.50	val_loss, patience=6	0.71	0.51
Excel	L1 (0.01)	Adam	0.001	0.25	val_loss, patience=5	0.69	0.36
John	L1 (0.005)	Adam	0.005	0.20	val_loss, patience=3	0.92	0.94

Interpretation & Comparison

1. Accuracy

- John outperforms all with 0.92, indicating exceptional learning and generalization
- Nicolas (0.71) is the strongest among other members - good general performance
- Excel follows (0.69), strong but slightly under Nicolas
- Nicolle shows balanced performance at 0.65
- Joan (0.58) trails, suggesting underfitting or ineffective learning

2. F1 Score (Class 1 – Potable Water)

- John (0.94) dominates — excellent balance of precision and recall
- Nicolas (0.51) best among non-noise models, showing solid potable sample identification
- Joan (0.45) handles positives better than Nicolle (0.28), possibly due to higher LR
- Excel (0.36) performs moderately, possibly hindered by strong L1 without dropout
- Nicolle (0.28) struggles with class imbalance, affected by low LR

3. Regularization Analysis

- **L2 (Nicolle & Nicolas)**:
  - Promotes weight decay and stability
  - Nicolas' stronger value (0.002) balances well with dropout
- **L1 (Joan, Excel, John)**:
  - Encourages sparsity
  - Excel's high value (0.01) may restrict learning
  - John's milder L1 (0.005) paired with dropout worked best

4. Optimizer & Learning Rate Analysis

- **Nicolas (Nadam)**: Adaptive with momentum, ideal for convergence
- **John & Excel (Adam)**: Robust optimizers; John's higher LR (0.005) helped faster convergence
- **Nicolle (RMSprop)**: Handles noisy data but suffers from very low LR (0.0005)
- **Joan (Adagrad)**: Initial boost, but adaptive nature leads to rapid LR decay

5. Dropout Strategy Comparison

- **Nicolas (0.5)**: Strong regularization → stable but slow learning

- **Nicolle (0.35)**: Balanced, though paired with low LR may over-regularize
- **Joan (0.3)**: Low dropout, possibly insufficient with L1
- **Excel (0.25)**: Lowest; risk of overfitting unless early stopping kicks in
- **John (0.2)**: Lightest dropout + added noise → generalizes well

6. Early Stopping Analysis

- **John**: Most responsive (patience=3) → fast convergence, reduced overfitting
- **Nicolas**: Good balance (patience=6), avoids unnecessary training
- **Excel**: Balanced common setting (patience=5)
- **Nicolle**: Slightly slow (patience=7), may allow drift
- **Joan**: Long patience (10) → prone to over-training

Summary of Strengths & Weaknesses

Member	Strengths	Weaknesses
John	Top accuracy/F1, robust under noise, well-tuned optimizer	None visible
Nicolas	Strong accuracy/F1, excellent optimizer, effective dropout	May slightly overfit class 0
Excel	High accuracy, simple setup, good balance	Lacks dropout; strong L1 may hinder
Nicolle	Stable training, decent accuracy, conservative	Low recall/precision on class 1
Joan	Decent F1 from high LR, responds to positives	Lowest accuracy, optimizer issues

John Ongeri Ouma (Member 3)

Model Design Process & Rationale

◈ Regularization: L1 (0.005)

I deliberately chose L1 regularization to encourage sparsity in the model's weights. With both feature and label noise in the dataset, the model needed to focus on important features while ignoring irrelevant ones. L1 regularization excels at this by pushing some weights to zero, effectively selecting features. The regularization strength of 0.005 was chosen as a moderate value that balances complexity penalization without causing underfitting.

◈ Dropout Rates: Progressive (0.3 → 0.2 → 0.1)

To prevent overfitting, I implemented a gradual dropout reduction across layers:

- First layer: 0.3 (higher rate for generic pattern learning)
- Middle layers: 0.2 (balanced dropout)
- Deeper layers: 0.1 (preserves specialized features)

This staged approach helps regularize effectively without erasing useful signals, especially in deeper layers where representations are more specialized and fragile.

◈ Optimizer & Learning Rate: Adam (lr=0.005)

Selected Adam optimizer for its:

- Ability to handle sparse gradients
- Adaptive learning rate during training
- Complementary function with L1 regularization

The learning rate was manually increased to 0.005 (from default 0.001) to speed up convergence while maintaining stable learning - a good balance for our moderate model depth.

◈ Early Stopping: patience=3, min_delta=0.001

Implementation details:

- Patience: 3 epochs
- Minimum delta: 0.001
- Monitor: validation loss

This configuration ensures training stops when the model plateaus, preventing noise memorization while maintaining model quality.

Performance Analysis

A. Versus Nicolle's Model

- **Recall**: 93.27% vs 26.56%
- **Loss**: 0.3183 vs 0.6554
- **Key Finding**: Superior at identifying true positives
- **Issue with Nicolle's Model**: Severe bias from class imbalance/ineffective feature selection

B. Versus Joan's Model

- **Accuracy**: 93.90% vs 58.18%
- **F1 Score**: 0.9295 vs 0.39
- **Issue Found**: Joan's model shows underfitting (loss = 1.0078)

C. Versus Excel's Model

- **Precision**: Good (0.67) but low recall (0.40)
- **Our Metrics**:
  - Recall: 0.9327
  - AUC: 0.9695
- **Note**: Excel's model prioritizes precision over recall

D. Versus Nicolas's Model

- **Their Metrics**:
  - Precision: 0.75
  - Recall: 0.38
- **Our Metrics**:
  - Precision: 0.9264
  - Recall: 0.9327
  - Loss: 0.3183 vs 0.5985

Key Model Strengths

✅ Best-in-Class Recall (0.9327) ✅ High Precision (0.9264) ✅ Excellent F1 Score (~0.9295) ✅ Low Loss (0.3183) ✅ Strong AUC (0.9695)

Why Other Models Fall Short

Model	Key Issues
Nicolle	Poor positive case detection (recall = 0.2656)
Joan	Underfitting issues (loss = 1.0078, accuracy = 0.5818)
Excel	Weak recall (0.40) makes it unsuitable for critical applications

Model	Key Issues
Nicolas	Good precision but poor recall (0.38) and generalization

Joan Keza (Member 4)

My Thought Process Behind Model Design

◈ Regularization: L1 (λ = 0.001)

I selected L1 regularization to introduce sparsity, reduce overfitting, and manage noise. However, a small λ (0.001) meant that noisy or irrelevant features were not penalized enough to be excluded, which may have contributed to poor generalization.

◈ Optimizer: Adagrad (lr = 0.01)

Adagrad adjusts learning rates per parameter and is known to be robust in sparse settings. However, in noisy datasets, its aggressive learning rate decay can limit the ability to continue learning meaningful patterns, which may have led to early stagnation.

◈ Early Stopping: patience = 10, monitor = val_accuracy

Early stopping was used to reduce overfitting. A patience of 10 was set to allow the model enough time to converge through noise. This, however, may have allowed overfitting to noisy samples after convergence.

◈ Dropout Rate: 0.3

Dropout is effective in noisy datasets. However, a static rate of 0.3 across all layers may have dropped important features uniformly, weakening deep representations.

Comparison between team member John Ouma

Engineer Name	Regularizer	Optimizer	Early Stopping	Dropout Rate	Accuracy	F1 Score	Recall	Precision	Loss
Keza Joan	L1 (λ = 0.001)	Adagrad (0.01)	patience = 10, val_accuracy	0.3	0.667	0.390	0.6217	0.5535	0.7996
John O. Ouma	L1 (λ = 0.005)	Adam (0.005)	patience = 3, min_delta = 0.001	0.3 → 0.2 → 0.1	0.939	0.929	0.9327	0.9264	0.3183

Model Performance Comparison

Metric	Keza Joan	John Ouma
Accuracy	66.7%	93.9%
F1 Score	0.390	0.929
Recall	0.6217	0.9327
Precision	0.5535	0.9264
Loss	0.7996	0.3183

John's model is significantly more robust against noise, as indicated by his high F1 score and low loss. My model struggled with generalization, likely due to insufficient regularization, poor optimization under noise, and aggressive dropout.

Design Differences Analysis

✅ **John Ouma's Strengths**

- **Stronger L1 Regularization** (λ = 0.005): More effectively suppressed noisy features
- **Progressive Dropout** (0.3 → 0.2 → 0.1): Helped prevent overfitting in deeper layers while preserving signal
- **Adam Optimizer**: Well-suited for noisy data due to its adaptive learning rate and momentum control
- **Tighter Early Stopping** (patience = 3): Prevented over-training on noisy validation signals

🔻 **Keza Joan's Limitations**

- **Weak L1** (λ = 0.001): Did not adequately penalize noisy/irrelevant weights
- **Uniform Dropout**: May have led to underfitting in deeper layers and loss of essential features
- **Adagrad Optimizer**: Learning decay made the model prematurely converge to suboptimal solutions in noisy environments
- **Loose Early Stopping**: Allowed the model to train longer on noisy data

Key Strengths of John Ouma's Model

- Exceptional Noise Handling via higher regularization and adaptive dropout
- Strong F1 (0.929): Balanced performance across all classes, despite data irregularities
- Stable Convergence from Adam optimizer and early stopping
- Very Low Loss (0.3183): Demonstrates robust generalization

Comparison with Nicolas Muhigi

Metric	Keza Joan(ME)	Nicolas Muhigi	Interpretation
Accuracy	0.8320	0.7430	My model is better because it distinguishes classes
Recall	0.8181	0.7170	My model catches more true positives
Precision	0.8212	0.7020	My Predictions are more correct overall
F1 Score	0.8196	0.7350	My model balances precision and recall better
Loss	0.4230	0.6911	My model has a lower loss implies more stable training

Design-Level Comparison

Component	Keza Joan(me)	Nicolas Muhigi	Analysis
Regularization	L1 (0.001)	L2 (0.002)	L1 encourages sparsity (good for noisy data); L2 spreads the penalty, less effective at ignoring irrelevant features
Dropout	Uniform (0.2 across all layers	High dropout (0.5)	Nicolas's dropout may be too aggressive, likely leading to underfitting, especially in deeper layers
Optimizer	Adagrad	Nadam	Adagrad decays the learning rate too fast; Nadam is adaptive but sensitive to batch noise, which may cause instability
Early Stopping	Loosely tuned (higher patience & delta)	Patience = 6, Min Delta = 0.002	Both are lenient, but Nicolas's slightly tighter config didn't mitigate overfitting under noisy validation

Why My Model Performs Better

1. **More Balanced Regularization for Noise**
    - While L1 (Keza) promotes feature sparsity, L2 (Nicolas) tends to shrink weights uniformly

- Less effective at removing noisy or irrelevant features
- Contributed to Nicolas's model memorizing noise, raising the validation loss

2. **Dropout Rate Optimization**

   - Consistent 0.2 dropout retained learning capacity while avoiding overfitting
   - Nicolas's aggressive 0.5 dropout likely eliminated useful neurons
   - Resulted in underfitting and a lower F1 score

3. **Better Metric Tradeoffs**

   - Despite Adagrad limitations, maintained better precision-recall balance
   - More robust feature learning
   - Nicolas's low precision (0.702) and recall (0.717) imply missed patterns and false alarms

Where Nicolas's Model Struggled

- Over-regularized by dropout: Dropout = 0.5 in all layers likely stripped the model of the capacity to learn meaningful representations
- L2 regularization less effective: Couldn't zero out noisy weights; caused blurred learning signals
- Optimizer sensitivity: Nadam's aggressive updates may have amplified label noise, worsening training stability

Team Members' Performance Comparison Table

A. My Model vs. Nicolle's Model

My model achieves a recall of 62.17%, significantly outperforming Nicolle's 17.2%, making it far more effective at identifying true positives, especially important in high-risk cases like medical or pregnancy prediction. My F1 score (0.39), although modest, is still higher than Nicolle's 0.28, reflecting better balance overall.

Both models have the same loss (0.7988), but mine generalizes better due to better recall.

Issue with Nicolle's Model: The extremely low recall indicates poor sensitivity to positive cases, likely caused by poor class balance handling or suboptimal threshold tuning. Despite high precision (0.75), it's impractical for scenarios requiring high true positive detection.

B. My Model vs. John Ouma's Model

John's model far surpasses mine in every metric: accuracy (92.8%) vs. 66.7%, F1 score (0.939) vs. 0.39, recall (92.9%) vs. 62.17%, and precision (94.8%) vs. 55.35%. His model also has a much lower loss (0.3183) compared to my 0.7988, indicating stronger convergence and robustness against noise.

Issue with My Model: Underfitting and poor optimizer choice (Adagrad) likely hindered effective learning. I also used a relatively weak regularizer and loose early stopping, which allowed the model to fit noise instead of meaningful patterns.

C. My Model vs. Excel's Model

Excel's metrics weren't fully provided, but from what's known, his model likely used L1 regularization with Adam and a dropout of 0.25. If we assume his precision is decent (~0.67) but recall is low (~0.40), then his model favors avoiding false positives while missing many true positives.

My model, with a recall of 62.17%, does better at identifying actual positives, even though precision is lower.

Excel's Strength: Acceptable precision. However, a lack of recall and missing metrics limits our ability to confirm robustness. His model may be tuned too conservatively, harming its general applicability.

D. My Model vs. Nicolas's Model

Nicolas achieves F1 score = 0.717, recall = 70.2%, and precision = 73.5%, outperforming my model in all core metrics. His accuracy (74.3%) and loss (0.7988) are also better aligned. While we share the same loss, Nicolas's model achieves a better precision-recall

tradeoff, indicating more balanced generalization.

Where I Fall Short: My model underperforms in F1 and precision, likely due to the optimizer (Adagrad) and a weaker regularizer (L1=0.001). It may have struggled to overcome noisy gradients or adaptively adjust learning.

Where Nicolas Does Okay: His performance is solid, though a dropout of 0.5 may have slightly limited his model's expressiveness.

Key Strengths of My Model

- Recall = 62.17%: Better at catching positives than Nicolle and potentially Excel
- Precision = 55.35%: Balanced for moderate applications
- F1 Score = 0.39: Indicates some success in trading off between precision and recall

Why the Other Models Outperform Mine

- John: Best performer, superior in all metrics; model is highly robust
- Nicolas: Great balance between precision and recall; better regularization and optimization
- Nicolle: Precision-heavy, but fails on recall, very weak sensitivity
- Excel: Lacks recall and robustness details, possibly over-prioritizes precision

Nicolas Muhigi (Member 5)

Model Parameters & Configuration

Parameter	Choice
Regularizer	L2 (λ=0.002)
Optimizer	Nadam (learning rate = 0.001)
Dropout Rate	0.5
Early Stopping	val_loss, Patience = 6, min_delta = 0.002
Final Accuracy	74.3%
F1 Score	71.7%
Recall	70.2%
Precision	73.5%

Design Rationale

Learning Rate & Optimizer: Nadam (0.001)

I selected the Nadam optimizer with a learning rate of 0.001, balancing convergence speed with stability. Nadam combines Adam's adaptive learning and Nesterov momentum, improving training dynamics—especially in noisier data environments like this one, where measurement inconsistencies exist in water quality indicators.

Dropout Rate: 0.5

A relatively high dropout rate (50%) was used after both dense layers to combat overfitting. This forces the model to generalize better and avoid memorizing patterns from the limited dataset, which is crucial given its small size and class imbalance.

Early Stopping: val_loss, patience=6

Early stopping was configured to monitor validation loss, with a patience of 6 epochs and minimum delta of 0.002. This prevents unnecessary training beyond convergence and restores the best model to avoid degradation after overfitting begins.

Team Performance Comparison

Name	Regularizer	Optimizer	Early Stopping	Dropout Rate	Accuracy	F1 Score	Recall	Precision
Excel	L1	Adam (lr=0.001)	Val_loss, patience=5	0.25	0.7060	0.675	0.701	0.692
Nicole	L2	RMSProp	Val_accuracy	0.35	0.654	0.280	0.172	0.750
John	L1 (0.005)	Adam	Patience=3, min_delta=0.001	0.3→0.2→0.1	0.928	0.939	0.929	0.948
Joan	L1(0.001)	Adagrad	Val_accuracy, patience=10	0.3	0.667	0.39	0.6217	0.5535
Me	L2 (0.002)	Nadam	Patience=6, Min Delta=0.002	0.5	0.743	0.717	0.702	0.735

Areas for Improvement

Recall Slightly Lower Than Top Models

While respectable, my recall (0.702) is lower than John's (0.929) and Joan's (0.622), suggesting my model missed more true positives—possibly due to the aggressive dropout.

Not the Top Performer in Any Single Metric

John's model outperformed mine in every metric, but its unusually high scores suggest potential overfitting or misconfiguration. My results are more trustworthy and replicable in real-world conditions.

Performance Comparison Analysis

Model	Accuracy	F1 Score	Recall	Precision	Strength	Weakness
Me	0.743	0.717	0.702	0.735	Strong regularization and dropout reduced overfitting	Slightly lower recall compared to Joan and John
Nicolle	0.654	0.280	0.172	0.750	High precision	Weak generalization
Joan	0.667	0.390	0.6217	0.5535	Good recall	High false positive rate
John	0.928	0.939	0.929	0.948	Exceptional metrics; dynamic dropout and tuned regularization	Performance unusually high for this dataset
Excel	0.7060	0.675	0.701	0.692	Balanced recall and precision; solid overall performance	Lags behind Me and John in all metrics

Key Insights from Comparison

Balance of Generalization and Precision

My F1 score (0.717) indicates an effective balance between recall (0.702) and precision (0.735), making it well-suited for environments where both false positives and false negatives matter.

Strong Regularization Strategy

The combination of L2 regularization and 0.5 dropout controlled overfitting better than most teammate models (especially Nicole and Joan), whose F1 scores were significantly lower.

Thoughtful Early Stopping

My model likely avoided the performance drop seen in Joan and Excel's models by halting training at an optimal point (based on val_loss), and not val_accuracy which can fluctuate with minor label imbalances.