Maryam Abrahams 18 September 2025

ABRMAR043 AI Assignment 2 - Supervised Learning

Perceptron Image Classification

Data Processing:

The Simpsons-MNIST dataset that we worked with consisted of 8000 training images and 2000 test images, each in black-and-white as well as in color, spread across 10 different characters from the Simposons show, each picture being 28*28 pixels. To get the data ready for our perceptron I had to load the images using PIL to read in the JPEG files and convert them into a numpy array for easier use as well as split the training dataset into a training and validation set, then normalize the data.

Design Implementation:

1. **load()**: After loading the data using PIL, I converted all the images to numpy arrays and created a mapping from character folder names to numbered labels 0-9.
2. **splits()**: I divided the training dataset into 80% training (6400 samples) images and 20% validation (1600 samples) using stratified sampling. This is because there is already a test set provided so that we can later evaluate the best settings for our hyperparameters.
3. **Data flattening**: All the images were flattened to 1D vectors - 784 features for grayscale (28×28) and 2352 features for RGB (28×28×3) so that it can be processed by my standard perceptron.
4. **normalize()**: I implemented three normalization strategies: none, min-max scaling to [0,1], and standardization, to later test their impact on perceptron performance and prepare my data for the perceptron models.
5. **Reproducibility**: Using a fixed random seed ensures reproducibility across runs.

Multi-Class Perceptron Implementation:

I implemented a Multiclass perceptron from scratch with two main classes (BinaryPerceptron - part of the multiclass perceptron - and MulticlassPerceptron) following the one-vs-rest approach, and taking guidance from the tutorials.

Design Implementation:

1. **BinaryPerceptron**: Implements the basic perceptron algorithm. It starts with initializing weights and the bias and uses a prediction method returning 0 or 1. It uses

the perceptron learning rule for weight and intercept updates. By tracking the number of errors per epoch I can monitor the convergence efficiency.

2. **MulticlassPerceptron**: Uses 10 binary perceptrons in a one-vs-rest setup. The predict method for each input sample calculates a confidence score from all 10 binary perceptrons and selects the class with the highest associated score.
3. **Object-Oriented Design**: Seperating the binary and multi-class classes makes the code modular and allows for better readability as wee as independent character classification.

Training:

I implemented enhanced training loops, **EnhancedBinaryPerceptron**, to investigate different stopping criteria and their effectiveness for perceptron convergence.

Design Implementation:

1. **Data Shuffling**: To prevent biased learning patterns of our perceptron from just the ordering of the data I randomly shuffled the training examples at the start of each epoch.
2. **Multiple Stopping Criteria**: Implemented three approaches: fixed number of epochs, error threshold, and early stopping based on validation accuracy with patience. I did this so that I could compare strategies to find the most efficient stop condition.
3. **Validation Tracking**: the *val_accuracies* list is where I track the validation performance over time so that I can identify overfitting.
4. **Convergence Analysis**: Used the Bart Simpson binary classification task to compare the effectiveness of the different stopping criteria. At first I had a loop which compares the effectiveness for each character but that took too long to run so I removed it since the results were consistent nonetheless.

Results Analysis & Discussion

Strategy	Epochs	Final Error	Final Accuracy
Fixed Epochs	500	901	0.883750
Error Threshold	1000	907	0.895625
Early Stopping	11	991	0.898750

```
=== Testing: Fixed Epochs (1000) ===
Epoch 100, Training error: 914
Epoch 200, Training error: 906
Epoch 300, Training error: 918
Epoch 400, Training error: 873
Epoch 500, Training error: 901
Reached maximum epochs (500)

=== Testing: Error Threshold (0 errors) ===
Epoch 100, Training error: 911
Epoch 200, Training error: 936
Epoch 300, Training error: 902
Epoch 400, Training error: 925
Epoch 500, Training error: 897
Epoch 600, Training error: 892
Epoch 700, Training error: 921
Epoch 800, Training error: 915
Epoch 900, Training error: 890
Epoch 1000, Training error: 907
Reached maximum epochs (1000)

=== Testing: Early Stopping (patience=10) ===
Early stopping at epoch 11, best accuracy value: 0.8988
```

Early stopping achieved the highest validation accuracy (89.88%) in just 11 epochs, showing that it is the most effective strategy for preventing overfitting. The error threshold approach failed to converge even after 1000 epochs, whereas the fixed epochs continued training unnecessarily similarly only stopping after a maximum of 500 epochs. All results showed fairly similar final accuracy so the quick speed of early stopping really differentiates from the other methods and stands out as the best.

Hyperparameter tuning:

Due to the massive runtime from resulting combinations, I restricted the number of combinations of hyperparameters that I tested on the validation set. Ideally I would've tested learning rates: 0.01, 0.05, 0.1, 0.5; initializations: constant, zero, uniform, gaussian; and nomralizations: none, min-max, z-score. That would have lead to 48 combinations on each of the 1600 validation images.

Design Implementation:

1. **Reduced Search Space**: Tested 2 learning rates [0.01, 0.5], 3 initialization strategies [zero, uniform, gaussian], and 3 normalization methods [none, z-score, minmax] to reduce the number of combinations to just 18.
2. **Fair Comparison**: I used identical hyperparamters for both the RGB and grayscale data to enable direct performance comparison and fiar results analysis.
3. **Computational Efficiency**: I limited the training to 20 epochs per combination to further reduce the runtime of this tuning task.

Results and Analysis:

	Best Grayscale	Best RGB
Learning Rate	0.5	0.5
Initialization	Gaussian	Gaussian
Normalization	none	none
Validation accuracy	0.2175 ≈ 22%	0.4094 ≈ 41%

```
BEST LEARNING RATE:
Gray: 0.5, RGB: 0.5

BEST INITIALIZATION:
Gray: gaussian, RGB: gaussian

BEST NORMALIZATION:
Gray: none, RGB: none

ACCURACY COMPARISON:
Grayscale: 0.2175, RGB: 0.4094
Difference: 0.1919 (RGB better)
```

For both the Grayscale and RGB implementations, a learning rate (α) of 0.5, the use fo the gaussian initialization strategy, and no normalization proved to be the best, producing the highest accuracy levels. In saying this however, an accuracy of 22% for grayscale and 41%

for RGB is not that high, likely due to how similar all the characters look stylistically, with color only distinctly differentiating a few (like marge with her blue hair).

Evaluation:

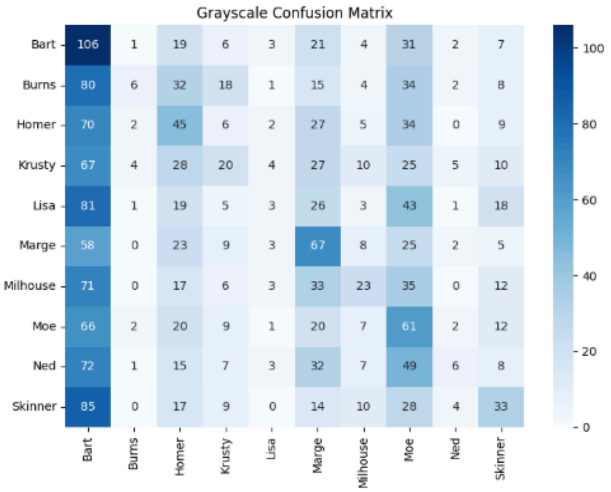
After selecting optimal hyperparameters, I trained the final models and evaluated them on both the validation and test sets using evaluation metrics such as accuracy, precision, recall and the F1-score, then constructed the corresponding (by character) confusion matrix.

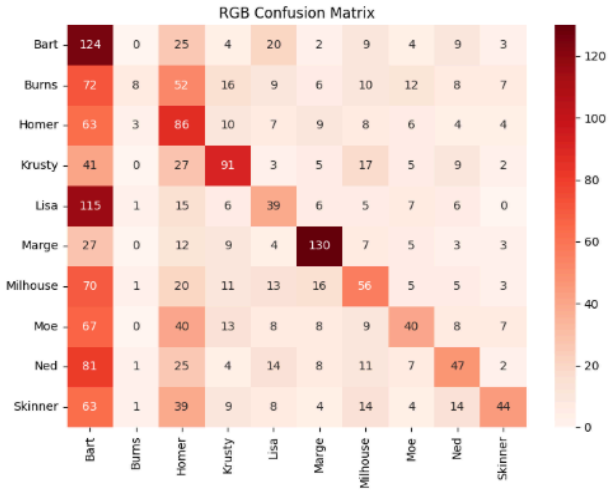
Design Implementation:

1. **Holistic analysis**: By looking at all four evaluation metrics: accuracy, precision, recall and the F1-score, I can get a much more rounded view pf my model's efficiency and correctness.
2. **Test Set Isolation**: By only evaluating the test set, after having already previously completed both training and validation, I avoided any data leakage.
3. **Confusion Matrix Analysis**: By separating the confusion matrices into grayscale and RGB I can see which characters are distinctly differentiated by their color profiles and better understand specific character challenges.

Graphs and Analysis:

Metric	Grayscale	RGB	Difference
Accuracy	0.1850 ≈ 19%	0.3325 ≈ 33%	0.1475 ≈ 15%
Precision	0.2235 ≈ 22%	0.4272 ≈ 43%	0.2037 ≈ 20%
Recall	0.1850 ≈ 19%	0.3325 ≈ 33%	0.1475 ≈ 15%
F1-score	0.1563 ≈ 16%	0.3264 ≈ 33%	0.1701 ≈ 17%





Overall accuracy, recall and the F1-score is quite low for both models with precision being slightly higher but still low for both as well. This shows that our model produces more false positive than false negative results. Across all metrics the RGB model outperforms the grayscale model by roughly double. All performance results however for both models are below 50% however, proving to be mostly inaccurate. This is likely due to the poor resolution of the images combined with how all characters are yellow and stylistically similar.

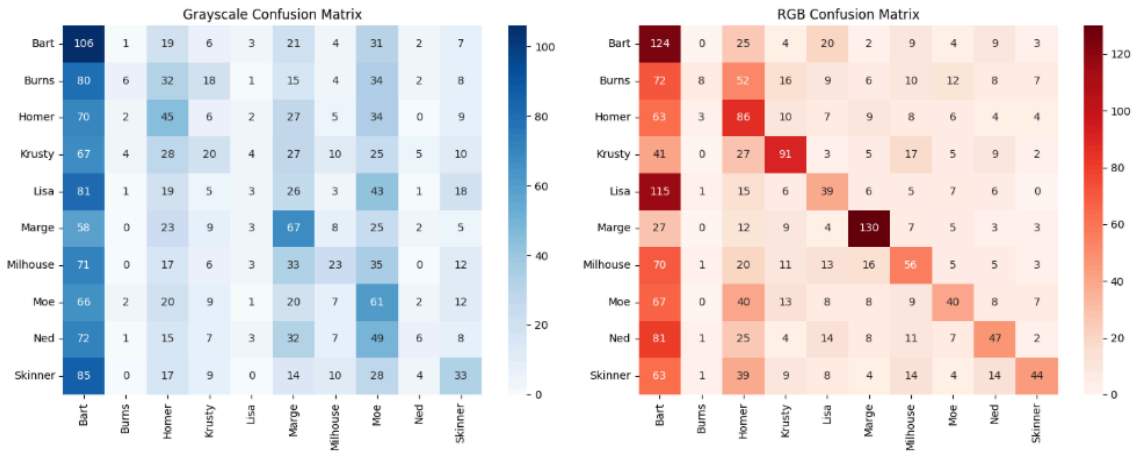
Within the confusion matrices, the leading diagonal shows the correct predictions per class while the off-diagonal rows per class show false negatives and the off-diagonal columns per class show false positives. From the confusion matrices we can see that other characters were often misclassfied as Bart, especially with Lisa in the colored analysis, likely because they're a similar height and he wears a red shirt while she wears a red dress. Marge, Krusty and Homer benefited the most from the RGB model due to their unique color profiles.

RGB vs Grayscale Analysis:

As previously discussed it can be see that the RGB model outperforms the grayscale model however it's better to identify by how much, and more importantly which characters benefit most and least from both our color and gray analyses.

Design Implementation:

1. **Per-Character Analysis**: Calculated F1-scores for each character for both RGB and Grayscale to identify which characters benefit most from color information.
2. **Statistical Significance**: By analyzing the magnitude of performance differences I can find out if RGB gives a meaningful advantage or not, even if there's no p-value method implementation.
3. **Visual Comparison**: Used side-by-side confusion matrices previously to highlight classification pattern differences between RGB and Grayscale.



```
Per-character analysis:
----------------------------------
Bart      : Gray=0.222, RGB=0.269,
Diff=+0.047
Burns     : Gray=0.055, RGB=0.074,
Diff=+0.019
Homer     : Gray=0.207, RGB=0.318,
Diff=+0.111
Krusty    : Gray=0.136, RGB=0.488,
Diff=+0.352
Lisa      : Gray=0.027, RGB=0.240,
Diff=+0.213
Marge     : Gray=0.278, RGB=0.660,
Diff=+0.382
Milhouse  : Gray=0.164, RGB=0.324,
Diff=+0.160
Moe       : Gray=0.216, RGB=0.271,
Diff=+0.055
Ned       : Gray=0.054, RGB=0.300,
Diff=+0.247
Skinner   : Gray=0.205, RGB=0.320,
Diff=+0.115
```

```
Characters that benefit most:
----------------------------------
1. Marge: +0.382 improvement (Gray:
0.278 → RGB: 0.660)
1. Krusty: +0.352 improvement (Gray:
0.136 → RGB: 0.488)
1. Ned: +0.247 improvement (Gray:
0.054 → RGB: 0.300)

Characters that benefit least:
----------------------------------
1. Moe: +0.055 change (Gray: 0.216 →
RGB: 0.271)
1. Bart: +0.047 change (Gray: 0.222
→ RGB: 0.269)
1. Burns: +0.019 change (Gray: 0.055
→ RGB: 0.074)
```

```
Most Challenging characters:
----------------------------------
Most challenging in Grayscale:
  Lisa: 0.027
  Ned: 0.054
  Burns: 0.055
Most challenging in RGB:
  Burns: 0.074
  Lisa: 0.240
  Bart: 0.269
```

Results Analysis:

Characters Benefiting Most from Color:
Marge (+0.38), Krusty (+0.35), Ned Flanders (+0.27)
This result is likely due to Marge's distinctive bright blue hair, Krusty's colorful palette and bright red nose, and Ned's brown mustache and green shirt. These important differentiating qualities aren't noticeable in the grayscale version.

Most Challenging Characters Overall:
Lisa (gray: 0.027, RGB: 0.269), Burns(gray: 0.055, RGB: 0.074)
Lisa's correct detection improves significantly from grayscale to RGB likely because of her bright red dress whereas Burns is definitely the most challenging character, barely benefitting from RGB due to lack of any defining or unique features. Moe similarly struggles to benefit from RGB but his unique hair and posture compensates for this.
