Report

Data processing

- I processed the data in the LoadSimpson class within which I labeled each of the character classes with the digits 0,1,2....9
- I then split the provided training data into a validation and training sets, 80% of it was training set and 20% was the validation set.
- Within this class I defined two important functions: one for calculating the normalization parameters per feature (i.e. min, max mean and standard deviation) from the training set and saving them so that I can use them later to normalize the validation and test sets
  -The other to normalize any data set, taking the previously computed normalization parameters as arguments to use if the incoming set is not the training set

Multi-class perceptron implementation:

Binary class

In the Binary Perceptron class, I treated bias as a weight as well following the linear algebra notation where b = w0

This is done to make it so that the bias term undergoes the same initialization as all the weights

I then split the bias and weights apart once more before applying the predict function that gives out 1 or 0

The Multiclass Perceptron class: it implements a One vs Rest multiclass classification

- This class initiates 10 binary classifiers and trains them independently, so it implements the training loop inside via the train () function
- It uses the **argmax** functionality inside the **predict**() function to classify based on which binary classifier has the gives has the biggest linear sum of the weights from the set {0,1,2,3...9}
- The class also allows for it to be used to evaluate different stopping conditions: using a Boolean variable to control whether to apply early stopping or not based on some patience variable
- And as an extra feature for visualization, it has a function: plot_training_history() to compare training and validation accuracies

I also designed a method :**stopping_conditions()**

- It prints out the results from different stopping conditions, it is not automated to choose the optimal conditions that is done by the one running the code based on their analysis of the output

Training:

Before doing Hyperparameter tuning I did various experiments for both RGB and grayscale to find the optimal stopping conditions and:

The two strategies I compared for stopping conditions was **a fixed number of epochs** (50,100 and 150 max epochs) and the **early stopping** criteria with patience variables: 4,8,10,20,30
Please note: the variable **patience = The number of epochs to wait after the last improvement in validation accuracy before stopping training when the validation accuracy stops improving

For the grayscale images the best conditions were: Early stopping with a patience =20

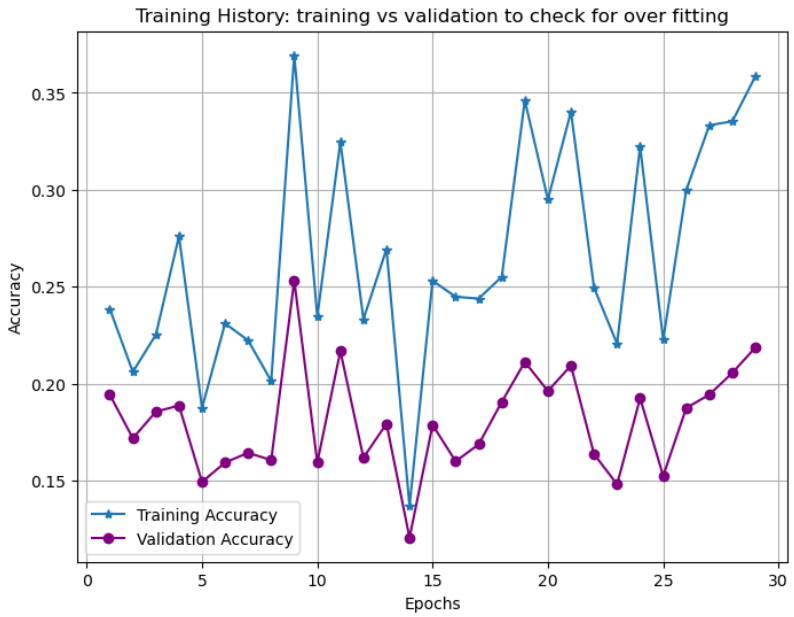
This condition was chosen because it produced the highest validation accuracy of 25.3% while having the smallest overfitting gap being around 14%

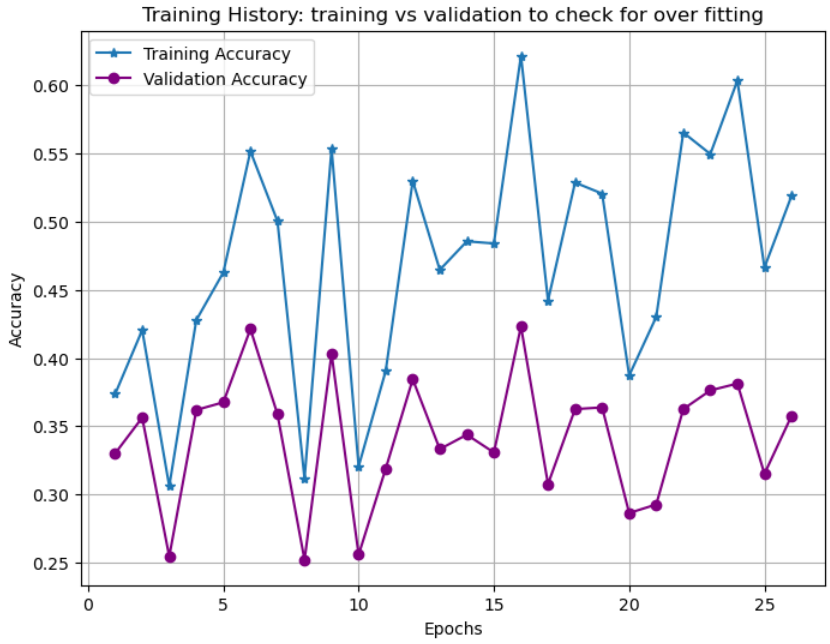
For the rgb images the best conditions were: Early stopping with a patience = 10

- For this there were other options that produced a higher accuracy for the validation set e.g. in one of my test runs, using a fixed number of epochs=150 gave an accuracy of 44.1% but a wide overfitting gap of 40.51%
- while the chosen option gave only 41.7% and an Overfitting Gap: 24.09%
- it was chosen because it provided a good balance between the validation accuracy and overfitting

In each case the maximum number of epochs =100

Below are the plots from the optimal stopping conditions (the one on top is for the Grayscale and the one below is for the RGB)





Hyperparameter tuning: (I had 27 total models/combinations)

These were my hyperparameter sets: learning_rates = [0.2,0.5, 1.0]

init_strategies = ['uniform', 'gaussian', 'zeros']

normalizations = ['normalization', 'standardization', 'none']

- I implemented the grid search using 3 for loops and for each combination e I stored the model to print the one that gave the highest accuracy later

The final optimal hyperparameters for each model were:

These were chosen from one of my test runs, because the pictures are shuffled each time training takes place it's possible that the best model as defined by the best parameters found, changes each time

The best GRAYSCALE Model (with a 26.3% accuracy)
learning rate: 1.0 Initialization strategy: gaussian Normalization type: none

The best RGB Model (with a 44.9 % accuracy)
learning_rate:1.0 Initialization strategy: gaussian Normalization type: normalization (0-1 normalization)

results and insights**:** In my earlier experiments i got a best model for the GRAYSCALE that was using standardization which was quite unusual as I expected grayscale images to do better under 0-1 normalization because it works best for data that has hard limits

Evaluation:

Grayscale: In this model Lisa was the character that misclassified as 196 out of 200 (98%) Lisas we misclassified, and the model had a 26.9% recall which was way too low leading to most of the characters other than Principal Skinner and Ned Flanders being misclassified

This may be due to Ned Flanders and Principal Skinner being distinguishable via their physical statures like Ned's moustache and glasses and the Principal having a more angular face than most characters.

RGB Model: In this model interestingly, Ned was the least misclassified character but was also the most predicted character leading to most other characters being wrongly predicted as Ned Flanders the model was biased, and it guessed Ned whenever it was unsure.

This leads to Ned having a way higher recall of 0.705 while having a low precision of 0.244 due to the model predicting Ned too many times while its actually other characters

Another thing to note is that the one character that was predicted the least to be Ned was Marge who had the highest precision and recall, and she has that classic blue hair that makes it easy to identify her. And two others to mention are Krusty the Clown and Milhouse Van Houten who each have color defined features and had very good precision and recall values.

RGB vs grayscale analysis: Overall

The very first thing I took note of was while testing for the optimal stopping condition the RGB model would give higher accuracy, and this was later confirmed in the last results

- This served as an indicator that the perceptron was more efficient when working with color i.e. my output "(RGB outperforms GRAYSCALE by: 22.900% accuracy)"

- With a recall of 41.6%: of all the characters, 41,6% of them were correctly identified by the RGB model while the Grayscale has a recall of 18.8%
  The grayscale performed poorly in all metrics compared to the rgb model

- Both models are rather careful seeing as they both have a higher precision than recall, but this means they both missed quite a lot of positive cases that were not apparent

RGB vs grayscale analysis: Per class

Marge Simpson and Krusty the clown are two biggest examples of how color leads to better and improved performance, the two characters had significant improvements in their recall, F1 and precision metrics

- Marge had the biggest improvement:
  going from: Precision of 0.325, Recall: 0.065, F1: 0.108
  -To: Precision: 0.760, Recall: 0.695, F1: 0.726
- Another character of interest is Milhouse Van Houten who has blue hair his recall also saw a good improvement from grayscale to rgb
- Principal Skinner has a smaller recall in the rgb model this was most likely due to my rgb model being biased towards predicting most classes as Ned Flanders also Principal is a character that is more unique in terms of physical shape than color, so these results are consistent with that factor.

- On average there was an increase in performance for most characters in the rgb model I can say based on the training I did, color provides a very good advantage even more so for characters that can be easily distinguished by color such as Marge's blue hair or Krusty as a clown having many colors

Acknowledgements: Used DeepSeek AI to debug my **split()** and **norm_params()** functions and the code for printing out the final two confusion matrices.