Rotterdam School of Management
Erasmus University

Individual Assignment 3

Experiment with Convolutional Neural Networks (CNNs)

Thy Cao – 673415dc

Table of Contents

1. Introduction

In this report, the use of Convolutional Neural Networks (CNNs) was explored for the classification of color blindness images. A baseline CNN model is built and several variations, tuning different parameters including the number of layers, number of filters, kernel sizes, and pool sizes, in our endeavor are modified to optimize performance using a subset of the Ishihara Color Blindness Chart Dataset. The key objectives include handling image data, enhancing the dataset, and communicating the machine learning process effectively.

2. Data Preparation

2.1. Data Collection

The dataset which is supposed to be used in this assignment is a publicly available Kaggle Dataset called "Ishihara Color Blindness Chart Dataset" (2022). However, due to the largeness of the dataset and time constraints, the tiny subset of this dataset provided by Gupta (2023) was used in this analysis. The dataset contains 53 images in total, with images classified into 9 subfolders based on the number that image display.

2.2. Image Processing

The image processing begins with setting the directory to the folder containing the subdirectories of the images. Each subdirectory represents a different label and contains images in .png format. Empty lists were initialized to store image paths, labels, and the combined images. Because there are 9 subdirectories, a loop was created to help scan through each subdirectory. The loop perfoms these operations:

i. Extracting the labels from the directory name, converting them to numeric values and store them in the label variable.
ii. Listing all .png files and storing them in the temp variable.
iii. Looping through each file, using the EBImage package to read images and storing them in the combined_images list.
iv. Storing the corresponding file paths and labels in the image_paths and labels variables, respectively.

Outside the loop, an empty list was created called resized_images. The images then are converted to arrays and resized to a uniform size of 128x128 pixels, alongside four channels accounting for the Red, Green, Blue and Alpha (transparent) color components, using the imager package. The images were not converted to grayscale and were processed with the colored versions. The resized images were stored in the resized_images list.

2.3. Data Partitioning

The dataset was then divided into training and testing sets. Before splitting, data were shuffled to ensure that both sets are representative of the overall data distribution, reduces the risk of bias and overfitting, and improves the generalization and performance of the model. The training set consists of 80% of the data whereas the testing set comprises 20% of the data. The labels variable was also splitted into train_labels and test_labels with the corresponding ratio. After splitting, train_labels and test_labels were coverted to numeric values to ensure that the labels remain numeric while the training and testing set were converted into arrays using abind funtion. Finally, the labels were converted to categorical formal and stored in the train_labels_cat and test_labels_cat variables.

3. Model Architecture

3.1. Model Design (Baseline Model)

The CNN model architecture consists of several convolutional layers, max-pooling layers, and fully connected layers. The architecture is designed to extract features from the images and classify them into the appropriate categories.

Structure:

1. **Input Layer**: The input layer is set to accept images of size 128x128 pixels with 4 color channels (Red, Green, Blue, and Alpha).
2. **Conv2D Layer**: The first layer is a 2D convolutional layer with 32 filters, each having a kernel size of 3x3. This layer applies 32 convolution operations, each generating a feature map. The ReLU activation function introduces non-linearity, helping the model to learn complex patterns.
3. **Max Pooling Layer 1**: The first max pooling layer downsamples the image data extracted by the convolutional layers to reduce the dimensionality of the image size. It does this using a 2x2 pool size.
4. **Convolutional Layer 2**: The second convolutional layer applies 64 filters, each with a kernel size of 3x3, again using the ReLU activation function.
5. **Max Pooling Layer 2**: Similar to the first max pooling layer, the second max pooling layer performs max pooling the image data with a 2x2 pool size.
6. **Conv2D Layer 3**: The third convolutional layer has 128 filters, each with a kernel size of 3x3. This layer continues to extract high-level features from the input data.
7. **Max Pooling Layer 3**: This layer performs max pooling with a pool size of 2x2, further reducing the size of the feature maps and aiding in computational efficiency.
8. **Flatten Layer**: This layer flattens the 3D feature maps into a 1D vector, preparing the data for the fully connected (dense) layers. It converts the multi-dimensional output of the previous layer into a single long vector.
9. **Dense Layer**: This fully connected layer has 512 units (neurons) with ReLU activation. This layer is responsible for combining the features learned by the convolutional layers and making predictions based on these features.
10. **Dropout Layer**: This layer randomly sets 50% of the input units to 0 at each update during training time, which helps prevent overfitting by ensuring that the model does not rely too heavily on any set of neurons.
11. **Output Layer (Dense Layer 2)**: The output layer has several units equal to the number of classes in the dataset (num_classes). The softmax activation function is used to convert the output into a probability distribution over the classes.

3.2. Model Compilation

The model is compiled with the following settings:

3.2.1. Loss Function:

The loss function used is 'categorical_crossentropy', which is appropriate for multi-class classification problems where the target labels are one-hot encoded. This function measures the difference between the predicted probability distribution and the actual distribution (true labels), guiding the optimization process to minimize this difference during training.

3.2.2. Optimizer:

The optimizer chosen is Adam (optimizer_adam()), which combines the advantages of two other popular optimization techniques: AdaGrad and RMSProp. Adam maintains adaptive learning rates for each parameter, making it efficient and effective for a wide range of problems. It helps in faster

convergence and often achieves better performance compared to traditional stochastic gradient descent.

3.2.3. Metrics:

The performance of the model is evaluated using 'accuracy', a straightforward metric that calculates the proportion of correctly classified instances out of the total instances. This metric provides an intuitive understanding of how well the model is performing on both the training and validation datasets.

3.3. Model Training

The model training process involves fitting the compiled CNN model to the training data, allowing it to learn the patterns and features necessary for accurate classification. The model is trained using the training data (train_array) and corresponding one-hot encoded labels (train_labels_cat). The fit function runs for 20 epochs, meaning the entire training dataset is passed through the model 20 times. A batch size of 32 is specified, indicating that the model updates its weights after processing 32 samples, which helps in managing memory usage and speeding up training. Additionally, 20% of the training data is set aside as a validation set (validation_split = 0.2), allowing the model's performance to be monitored on unseen data during training. This validation process helps in identifying overfitting, ensuring that the model generalizes well to new data. The history object stores the training metrics and loss values for each epoch, which can be used later for visualization and analysis of the training progress. Overall, this setup ensures an efficient and effective training process, optimizing the model's ability to classify the images accurately.

4. Model Improvement

This section presents the architecture of the baseline CNN model. 2 primary methods were experimented: data augmentation and data enlargment. For data enlargement, the number of layers and the number of filters will be modified to observe the changes. For each parameter, I created two variations of the baseline model, one with an increase and one with a decrease. The architecture of these variations differs from the baseline model. All models maintain the same compilation parameters and training parameters as in the baseline model.

4.1. Data Augmentation (Model 2)

Data augmentation was employed to enhance the model's ability to generalize to new, unseen data by using built-in functions in Keras 3 packages. Data augmentation involves expanding the training dataset by applying various transformations to the original images. These transformations include random horizontal flipping, rotations, zooming, contrast adjustments, brightness variations, and translations. Specifically, the following augmentation techniques were used:

i. **Random Horizontal Flip**: This flips the image horizontally with a certain probability, helping the model to recognize objects in different orientations.
ii. **Random Rotation**: The images were randomly rotated by a small factor (0.1), allowing the model to become invariant to small rotational changes.
iii. **Random Zoom**: Zooming in and out randomly on the images helps the model learn to recognize objects at different scales.
iv. **Random Contrast**: Adjusting the contrast of the images randomly can improve the model's robustness to varying lighting conditions.
v. **Random Brightness**: Similar to contrast adjustments, varying the brightness of the images ensures that the model can handle different lighting environments.
vi. **Random Translation**: Shifting the images horizontally and vertically ensures that the model can accurately identify objects that might not be perfectly centered.

The number of epochs in this model is 50.

4.2. Data Enlargement

My dataset was expanded by adding 442 images from the "Ishihara Blind Test Cards" (2020) dataset on Kaggle with each classification having 55 images, increasing the number of images to 495 in total. Image Magick had to be installed to convert these images' white background to transparent background due to the difference in the number of the color channels (RGB vs RGBA). Below are the architectures of models with modified parameters. The structure in data augmentation would not be reused in this section.

4.2.1. Investigating the Baseline Architecture (Model 3a)

To see the differences after expanding the dataset, the architecture of the Baseline Model was used. Therefore, the first model's architecture is the same as the Baseline Model.

4.2.2. Increasing the Number of Layers (Model 3b)

In this model, a third convolutional layer and its corresponding max pooling layer is added to the Baseline Model. The structure of this model is as follows:

1. Input Layer: 128x128x4
2. Convolutional Layer 1: 32 filters, 3x3 kernel, ReLU activation
3. Max Pooling Layer 1: 2x2 pool size
4. Dropout Layer 1: 25% rate
5. Convolutional Layer 2: 64 filters, 3x3 kernel, ReLU activation
6. Max Pooling Layer 2: 2x2 pool size
7. Dropout Layer 2: 25% rate
8. Convolutional Layer 3: 128 filters, 3x3 kernel, ReLU activation
9. Max Pooling Layer 3: 2x2 pool size
10. Dropout Layer 3: 25% rate
11. Flatten Layer
12. Dense Layer 1: 128 units, ReLU activation
13. Dropout Layer 4: 50% rate
14. Output Layer (Dense Layer 2): 2 units, softmax activation

4.2.3. *Decreasing the Number of Layers (Model 3c)*

In this model, the second convolutional layer and its corresponding max pooling layer is removed from to the Baseline Model. The structure of this model is as follows:

1. Input Layer: 128x128x4
2. Convolutional Layer 1: 32 filters, 3x3 kernel, ReLU activation
3. Max Pooling Layer 1: 2x2 pool size7
4. Dropout Layer: 25% rate
5. Flatten Layer
6. Dense Layer 1: 128 units, ReLU activation
7. Dropout Layer: 50% rate
8. Output Layer (Dense Layer 2): 2 units, softmax activation

4.2.4. Increasing the Number of Filters (Model 3d)

In this model, the first convolutional layer now applies 64 filters (up from 32) and the second convolutional layer applies 128 filters (up from 64). The structure of this model is as follows:

1. Input Layer: 128x128x4
2. Convolutional Layer 1: 64 filters, 3x3 kernel, ReLU activation
3. Max Pooling Layer 1: 2x2 pool size
4. Convolutional Layer 2: 128 filters, 3x3 kernel, ReLU activation

5. Max Pooling Layer 2: 2x2 pool size
6. Dropout Layer: 25% rate
7. Flatten Layer
8. Dense Layer 1: 128 units, ReLU activation
9. Dropout Layer: 50% rate
10. Output Layer (Dense Layer 2): 2 units, softmax activation

1.1.1. Decreasing the Number of Filters (Model 3e)

In this model, the first convolutional layer now applies 16 filters (down from 32) and the second convolutional layer applies 32 filters (down from 64). The structure of this model is as follows:

1. Input Layer: 128x128x4
2. Convolutional Layer 1: 16 filters, 3x3 kernel, ReLU activation8
3. Max Pooling Layer 1: 2x2 pool size
4. Convolutional Layer 2: 32 filters, 3x3 kernel, ReLU activation
5. Max Pooling Layer 2: 2x2 pool size
6. Dropout Layer: 25% rate
7. Flatten Layer
8. Dense Layer 1: 128 units, ReLU activation
9. Dropout Layer: 50% rate
10. Output Layer (Dense Layer 2): 2 units, softmax activation

5. Evaluations

5.1. Baseline Model

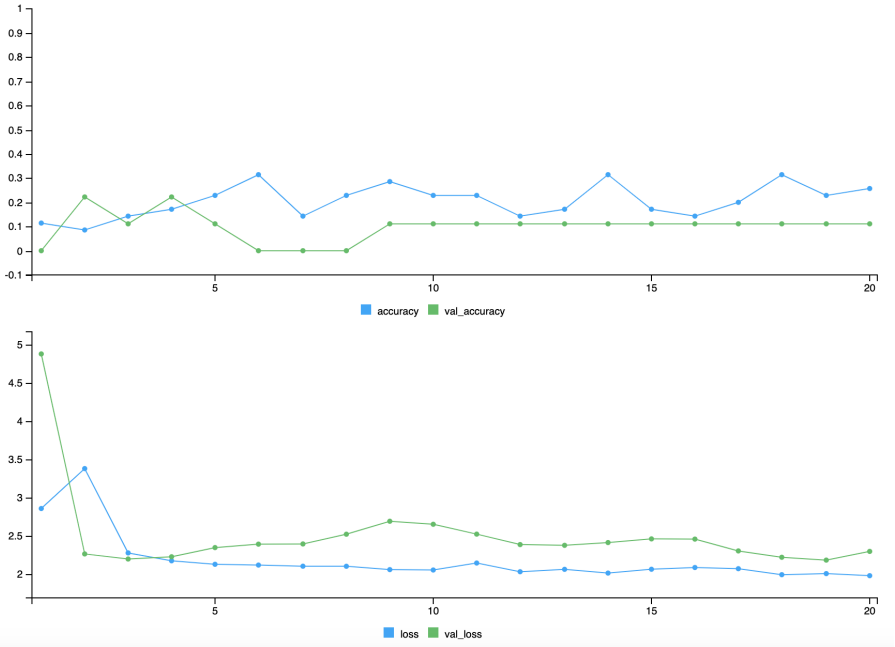

Figure 1. Baseline Model Plot

The evaluation of the CNN model reveals that the model's performance is currently suboptimal. From Figure 1, it is clear that despite 20 epochs of training, the model achieved an accuracy of only 10% on the test dataset, with a loss of 2.29. This indicates that the model is struggling to learn effectively from the training data. The low accuracy and high loss suggest issues such as insufficient training data, suboptimal model architecture, or inadequate training duration.

Additionally, the model's validation accuracy and loss show significant variability, which could imply overfitting or underfitting.

5.2. Data-Augmented Model

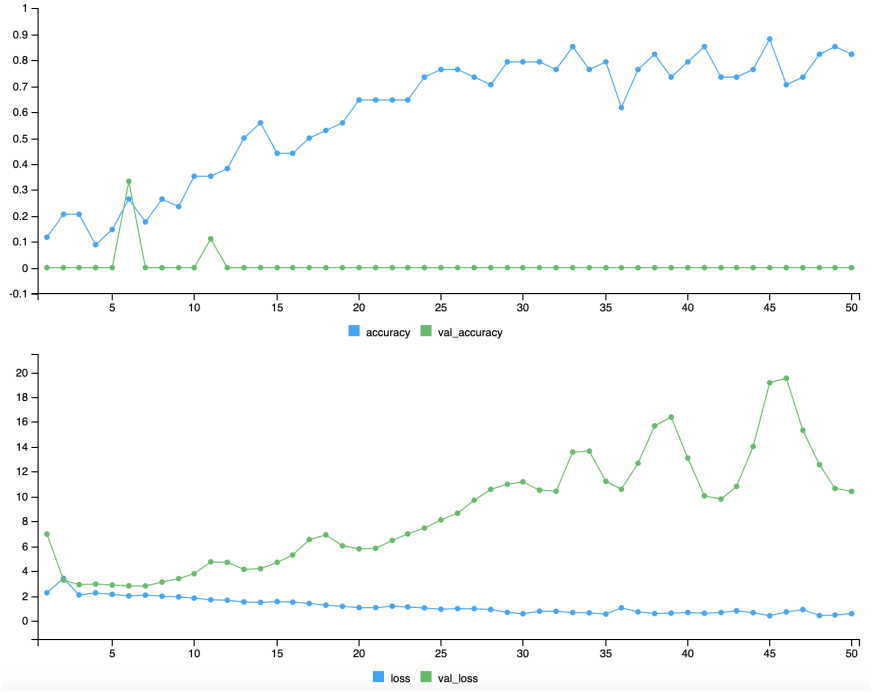

Figure 2. Data Augmented Model Plot

From Figure 2, the evaluation of the CNN model shows an accuracy improvement to approximately 36.36% on the test dataset, with a loss of 1.94, yet the performance remains suboptimal. Several factors may explain this. Firstly, the dataset's small size limits the model's ability to learn robust features, as deep learning models typically require large amounts of data for effective generalization. Additionally, there might be class imbalance, causing the model to favor certain classes over others. Although the model architecture is standard, it may not be the best fit for the specific dataset. The model also exhibits signs of overfitting, as indicated by the significant increase in training accuracy while the validation accuracy remained at 0%, suggesting poor generalization. Hyperparameter choices, such as learning rate, batch size, and number of epochs, may not be optimal and require systematic adjustment. Data preprocessing issues, including resizing and normalization, could impact performance, highlighting the need for consistent and effective preprocessing. The small dataset and 20% validation split may result in too few validation samples, leading to unreliable validation metrics, where cross-validation could provide more stable performance estimates.

5.3. Data Enlargement Models

Model	Accuracy	Loss
Baseline	10.09832%	2.287992
3a	13.54167%	2.216694
3b	14.58333%	2.197651
3c	15.625%	2.187722
3d	17.70833 %	2.191257

3e	30.20833%	2.214258

Table 1. Performance Metrics

Table 1 compares the accuracy and loss of each model. The first model, 3a, reuses the baseline architecture to observe differences after expanding the dataset. This model achieved an accuracy of 13.54167% and a loss of 2.216694, showing a slight improvement over the baseline.

Model 3b adds a third convolutional and max-pooling layer to the baseline architecture. The structure includes layers with 32, 64, and 128 filters, each followed by max pooling and dropout. This model achieved 14.58333% accuracy and 2.197651 loss, indicating a beneficial impact of the additional layer.

Model 3c simplifies the baseline by removing the second convolutional and max-pooling layer. The architecture includes one convolutional layer with 32 filters and a single max-pooling layer. This simplification led to an accuracy of 15.625% and a loss of 2.187722, suggesting that fewer layers can still improve performance.

Model 3d increases the number of filters in the first two convolutional layers to 64 and 128, respectively. This model achieved an accuracy of 17.70833% with a loss of 2.191257, demonstrating that increasing filter counts can enhance model performance.

Conversely, Model 3e reduces the filter counts in the first two convolutional layers to 16 and 32, respectively. Remarkably, this model achieved the highest accuracy of 30.20833% while maintaining a loss of 2.214258. This suggests that reducing the filter count can significantly enhance performance, likely due to overfitting reduction.

6. Conclusion

In this study, the use of Convolutional Neural Networks (CNNs) was explored for classifying images from the “Ishihara Color Blindness Chart Dataset”. Through the investigation of various model architectures and data augmentation techniques, optimization for the performance of my models is the priority. The Baseline Model provided a starting point, achieving an accuracy of 10.09832% with a loss of 2.287992. Data augmentation and enlargement strategies also played a significant role in improving model performance. The augmented model reached an accuracy of approximately 36.36% on the test dataset with a loss of 1.94, demonstrating the effectiveness of these techniques in enhancing the generalization capability of the CNN models. Overall, the findings underscore the importance of experimenting with different model architectures and data handling techniques in optimizing CNN performance. The substantial improvement observed with Model 3e highlights the potential benefits of simplifying the model and reducing overfitting, providing valuable insights for future work in image classification tasks.

References:

Ishihara Like MNIST. (2022, June 17). Kaggle. https://www.kaggle.com/datasets/ammarshaker/ishihara-mnist

Gupta, M. K. (2023, May 27). Color Blindness chart Reader - Mayank Kumar Gupta - Medium. *Medium*. https://medium.com/@mayankkumargupta/color-blindness-chart-reader-c71e804b6f70

Ishihara blind test cards. (2020, December 6). Kaggle. https://www.kaggle.com/datasets/dupeljan/ishihara-blind-test-cards