PROJECT PROPOSAL: USING CNNS TO CLASSIFY IMAGE FILTERS

David Xu
Student# 1008756776
dagsion.xu@mail.utoronto.ca

Dylan Alianto
Student# 1008780035
dylan.alianto@mail.utoronto.ca

Kevin Hu
Student# 1006938888
kevi.hu@mail.utoronto.ca

Zhuolin Feng
Student# 1009095765
zhuolin.feng@mail.utoronto.ca

ABSTRACT

—-Total Pages: 9

1 BRIEF PROJECT DESCRIPTION

This project aims to train a deep neural net that is capable of identifying filters applied to images and reversing them. The primary motivations of identifying and reversing filters are media verification and especially data preprocessing for other neural nets. Neural nets working on images scraped from the internet need clean, unedited images since in a technical sense, filters can be considered a type of noise. The primary goal of this project forms a critical data processing step in many other projects.

In general, these filters are linear transformations on RGB channels and include greyscale, sepia, and black and white, but can be expanded upon to include other ones, even nonlinear filters. Filters can be global, affecting the whole image, or local, such as a vignette only darkening areas outside of the center. Aside from pure linear functions on RGB channels, they can also adjust things like contrast and blurring.

The expected input is an image, and there are two expected outputs. The first output is simply a classification of what filter has been applied. The second is the network's best model of the linear transformation applied to some original image to get the input image. Applying the inverse of this linear transformation will produce the model's best approximation at the original image before the filter was applied.

The reason machine learning is a good tool for this is twofold. Firstly, neural nets need a lot of data - far more than is remotely feasible to look through by hand. Secondly, the problem is extremely open-ended. Even after identifying the precise filter or possibly stack of filters used, each filter can produce hundreds of permutations depending on the specific implementation of that filter, the brightness, and the selected strength of the applied filter. Thirdly, traditional methods may have difficulty with non-linear transformations, especially where loss of information is involved such as blurring and vignetting. Finally, each original image is different in features, brightness, colors, and noise such as blurring. This extreme open-endedness means that no automated process with a set algorithm or formula could possibly be used, leaving an adaptable neural net that can generalize to new data as the only option.

2 INDIVIDUAL CONTRIBUTION AND RESPONSIBILITIES

This section outlines each member's contributions to maintain the progress of the project and reminds the responsibility of one's tasks. Each role was determined with a commitment to consent, collaboration, and fairness in workload distribution. The responsibilities detailed here reflect a balanced effort to maintain both individual accountability and team synergy, contributing to the project's overall success.

We work together productively to ensure consistent progress, the team meets in person every Thursday at 7 PM in the SF1013 lab room to check in with each member's tasks, progress, and further instructions. In addition, our team collaborates to maintain accountability, using Discord for efficient communication and quick virtual discussions. Furthermore, we use Google Colab to track updates and version history for shared code, manage explanatory comments, and maintain an organized workflow for code integration.

To provide a comprehensive view of our project plan, the following two tables are included:

Table 1: Updated Divided Tasks for Each Individual Group Member For Future Task

DELIVERABLE	**SUBTASK**	**ASSIGNED**	**DEADLINE**
PROJECT PRESENTATION	Problem	Dylan A.	2024-11-22
	Data	Zhuolin F.	2024-11-25
	Data Processing	Zhuolin F.	2024-11-25
	Demonstration	David Xu	2024-11-25
	Quantitative Results	David Xu	2024-11-27
	Qualitative Results	Kevin Hu	2024-11-27
	Takeaways	Dylan A.	2024-11-27
	Video Editing	Everyone	2024-11-28
FINAL REPORT	Introduction	Dylan A.	2024-11-25
	Illustration/Figure	Dylan A.	2024-11-25
	Background and Related Work	Kevin Hu	2024-11-25
	Data Processing	Zhuolin F.	2024-11-27
	Architecture	David Xu	2024-11-27
	Baseline Model	David Xu	2024-11-27
	Quantitative Results	Kevin Hu	2024-11-27
	Qualitative Results	Dylan A.	2024-11-27
	Evaluate Model on New Data	Zhuolin F.	2024-11-27
	Discussion	Zhuolin F.	2024-11-25
	Ethical Considerations	David Xu	2024-11-25
	Project Difficulty / Quality	Kevin Hu	2024-11-25
	Edits & References	Everyone	2024-11-29

Table 2: Completion of Tasks per Group Member

Member	Completed Task	Next Step
David Xu	• **Baseline Model:** Developed functions to apply grayscale, sepia, and inverted filters to images, saving the transformed versions to designated directories for organized data processing. Preprocessed images through resizing and flattening, creating a uniform input format for the SVM model. • **Primary Model:** Monitored training for Dylan's modification of the primary CNN model, providing support by debugging minor issues encountered during training. • **Report Writing:** Authored sections detailing team contributions and responsibilities, as well as the Baseline Model section for the report, outlining implementation, processes, and results.	Prepare a small testing dataset to facilitate rapid debugging of the primary and baseline models without the need to download all datasets.
Dylan A.	• **Baseline Model:** Developed an SVM (Support Vector Machine) classifier for filter detection to achieve effective classification for three filter types. Loaded image data into training and test sets, training the SVM model on these sets. Evaluated model performance using metrics such as test accuracy, confusion matrix, and classification report. • **Primary Model:** Supported modifications to Kevin's implementation of the primary CNN model, providing essential code for performance evaluation functions. • **Report Writing:** Authored the introduction and the quantitative and qualitative analysis sections for the primary model, adding and editing other sections as necessary.	Adjust the baseline model's complexity if necessary to prevent overfitting due to its current high performance.
Kevin Hu	• **Data Processing:** Implemented functions for downloading datasets into Google Colab. Wrote code for initial data preprocessing, including unzipping and organizing data into directories, followed by renaming files for indexability. Developed functions to homogenize image sizes by discarding images smaller than 150 by 150 pixels while cropping and resizing larger images. • **Primary Model:** Created a DataLoader function to efficiently pull data from directories, splitting it into training, validation, and test sets while shuffling the data. Additionally, implemented other helper functions for the primary model. • **Report Writing:** Drafted the section on the primary model and made revisions to other sections of the report.	Continue refining data preprocessing techniques and tuning the primary model for better performance.
Zhuolin Feng	• **Data Processing:** Finalized data processing by generating filtered images from original datasets.	N/A (Course dropped).

3 Data Processing

Our goal for this project is to develop a model capable of categorizing images based on various applied filters. To accomplish this, we created a dataset of photos with a size of 150x150 pixels and split it into training, validation, and test sets. Cropping was minimized to only obtain square images. We used the Numpy array representation to crop each image to a centered square. To downsize the image, we used the native Pillow resizing function. Next, to prepare the dataset for training the model to recognize filter-specific features, we applied three different filters to each image to create variety.

Our dataset was curated with ethical considerations and model limitations in mind. Therefore, we selected two types of images: photos of Indian food and environmental landscapes. Both types of data were obtained from publicly available datasets on Kaggle. By using visually diverse content, we aimed to help the model learn to identify patterns that would enable it to distinguish between filters across varied subjects. In particular, the food images provide variety in color but tend to have important features centered in the middle, while landscape images are less diverse in color, presenting the model with more challenges in identifying color-changing transformations while having information distributed across the entire image. Each image was processed uniformly to prepare it for modeling.

3.1 Data Collection and Organization

The images were sourced from Kaggle's Indian food and environmental photo datasets, which offered a diversity of textures, colors, and features. To streamline processing, we downloaded and organized the images into separate folders for each category within Google Colab. By using Google Colab for dataset integration, we ensured smooth access to storage and processing tools, enabling organized data preparation for model training. The number of Indian food images we processed was roughly 4000, and the number of landscape images was about 7000.

3.2 Image Resizing

To standardize the dataset, each image was resized to 150x150 pixels. This step was essential to maintain uniform input dimensions across the dataset, allowing for consistent data processing and optimizing compatibility with our model. After resizing, we examined a sample of images to confirm that they had been resized correctly and matched the target resolution. This verification ensured that the dataset was ready for the next phase of filtering.

3.3 Applying Filters

We applied three filters—Sepia, Inverted, and Grayscale—using RGB functions for accuracy and uniformity. This created the variations that the model would learn to recognize. Each filter introduced a unique transformation:

- **Sepia Filter**: Adds a warm, vintage effect by adjusting RGB values to mimic the look of old film.
- **Inverted Filter**: Reverses the RGB values, producing a distinct "negative" effect for each image.
- **Grayscale Filter**: Converts the images to grayscale, removing color information and leaving only brightness values.

Using image processing code, we systematically applied these filters to every image, creating three filtered versions for each original one. From the roughly 4000 original Indian food images and 7000 landscape images, we obtained 12000 filtered Indian food images and 21000 filtered landscape images. They are stored in six different directories labelled by filter and category. After processing, we examined several filtered images to verify that each filter had been applied correctly. This visual validation confirmed that our code functioned as expected and produced accurately transformed images for each filter type.

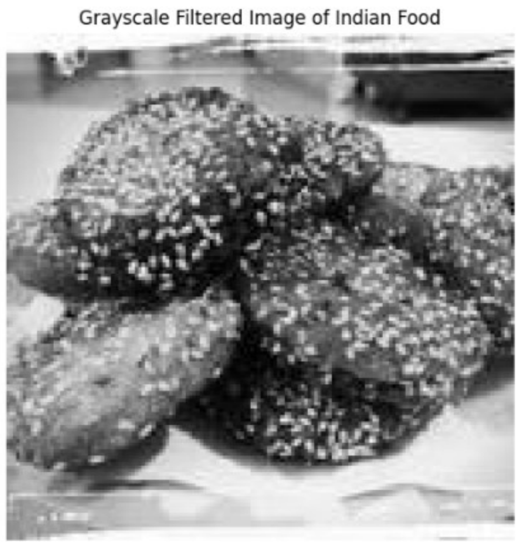


Figure 1: Example of an image applied with a grayscale filter and resized to 150x150.

Below is the code that operated on the RGB channels of each image to add a filter.

```
# Grayscale filter
def apply_grayscale(image):
    grayscale = np.dot(image[...,:3], [0.299, 0.587, 0.114])
    return np.stack((grayscale, grayscale, grayscale), axis=-1).astype(np.uint8)

# Sepia filter
def apply_sepia(image):
    sepia = np.copy(image)
    tr = 0.393 * image[:,:,0] + 0.769 * image[:,:,1] + 0.189 * image[:,:,2]
    tg = 0.349 * image[:,:,0] + 0.686 * image[:,:,1] + 0.168 * image[:,:,2]
    tb = 0.272 * image[:,:,0] + 0.534 * image[:,:,1] + 0.131 * image[:,:,2]
    sepia[:,:,0] = np.clip(tr, 0, 255)
    sepia[:,:,1] = np.clip(tg, 0, 255)
    sepia[:,:,2] = np.clip(tb, 0, 255)
    return sepia.astype(np.uint8)

# Inverted filter
def apply_inverted(image):
    inverted = 255 - image
    return inverted.astype(np.uint8)
```

3.4 Data Splitting and Categorization

With all images processed and filtered, we randomly divided the dataset into training, validation, and test sets with a proportion of 70%, 15%, and 15%, respectively. Each subset was labelled with categories (Indian food and landscapes) and filter types (Sepia, Inverted, Grayscale). This setup ensures that the model would be exposed to a diverse range of images across all stages. By splitting the data randomly, we minimized potential biases and achieved an even distribution of classes in each subset.

Each set of filtered images appears in the same split. That is, if the grayscale version of a given image appears in the training set, the sepia and inverted versions of that image are guaranteed to also be in the training set. This ensures that the test and validation sets contain images that have not been seen in any filtered version of the training data. The data is only shuffled randomly according to a set random seed value after the data has been split into training, validation, and testing.

For initial testing and model bring-up, we elected to use only the Indian food datasets. This is because the Indian food dataset alone, with about 12000 filtered images, provides a high degree of colour variation and significantly reduces the time needed for testing and initial model setup compared to using the combined datasets of 33000 images.

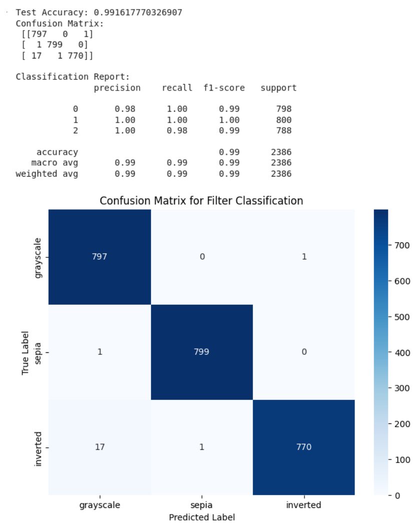
4 Baseline Model

We chose a Support Vector Machine (SVM) with a linear kernel for our baseline model. This choice is based on the SVM's effectiveness in straightforward, linearly separable tasks, making it a strong benchmark for comparison with our more complex Convolutional Neural Network (CNN) model. The SVM is easy to set up and requires minimal tuning, allowing us to establish a baseline quickly.

Our SVM model uses features from preprocessed image data, with each image resized and flattened into a one-dimensional vector. Images are classified into three filter types: grayscale, sepia, and inverted. The model then learns to distinguish between these transformations based on their pixel patterns.

4.1 Quantitative Results

The SVM model demonstrated high classification accuracy, achieving a test accuracy of 99.16% on the test set. This result indicates that the model is effective at distinguishing between the three filters applied to the images. Additionally, the precision, recall, and F1 scores were all close to 1.00 across each class, meaning that the model performs consistently well in identifying each filter type. The confusion matrix revealed a few misclassifications that primarily occurred between grayscale and inverted images, likely due to similar tonal transformations these filters introduce. This strong quantitative performance suggests that the SVM model is a solid baseline for evaluating filter classification.



```
Test Accuracy: 0.991617770326907
Confusion Matrix:
 [[797   0   1]
 [  1 799   0]
 [ 17   1 770]]

Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       798
           1       1.00      1.00      1.00       800
           2       1.00      0.98      0.99       788

    accuracy                           0.99      2386
   macro avg       0.99      0.99      0.99      2386
weighted avg       0.99      0.99      0.99      2386
```

Figure 2: Confusion matrix for SVM model showing filter classification performance.

4.2 Qualitative Results

In terms of qualitative performance, the SVM model classified images consistently across different types, as reflected by the high recall and precision values. However, a pattern emerged where grayscale and inverted images were occasionally confused, indicating that subtle tonal similarities between these filters can create challenges for the model. While reviewing sample images, we observed that the SVM performed well on images with clear, well-defined filter applications. Overall, these qualitative insights highlight the SVM's strengths in simple classifications, while suggesting potential areas for improvement, such as feature tuning or adding texture-based information to improve filter distinction.

4.3 Challenges and Feasibility

One of the main challenges was handling potential misclassifications, particularly between grayscale and inverted filters, which share some visual similarities. Despite this, the SVM model demonstrated that non-deep learning approaches could be feasible for simple filter detection. However, the baseline's performance may decline with more complex or mixed filters, reinforcing the need for a more sophisticated CNN model to tackle intricate transformations.

5 Primary Model

For our primary model, we have implemented a Convolutional Neural Network (CNN) with an architecture tailored to detect image transformations, specifically filters like grayscale, sepia, and inversion. Our current architecture is simpler than what we initially outlined in the project proposal, but it has been effective in capturing the desired patterns within the dataset.

The flowchart below illustrates the architecture and workflow of our primary Convolutional Neural Network (CNN) model designed for filter classification and transformation estimation. Our process begins with data preparation, where a large dataset of images is collected, and resized, and various filters (grayscale, sepia, inverted) are applied to create a diverse training set. Each image is paired with a ground truth (unfiltered original) and labelled based on the applied filter.

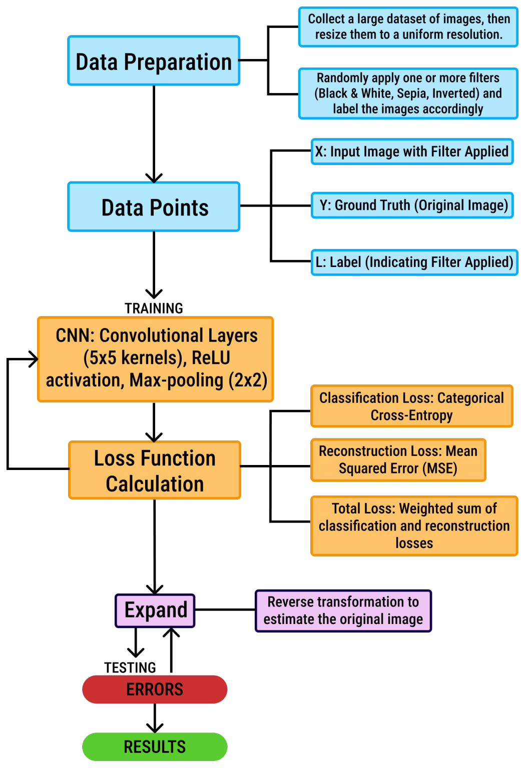

Data Preparation
Collect a large dataset of images, then resize them to a uniform resolution.
Randomly apply one or more filters (Black & White, Sepia, Inverted) and label the images accordingly
Data Points
X: Input Image with Filter Applied
Y: Ground Truth (Original Image)
L: Label (Indicating Filter Applied)
TRAINING
CNN: Convolutional Layers (5x5 kernels), ReLU activation, Max-pooling (2x2)
Loss Function Calculation
Classification Loss: Categorical Cross-Entropy
Reconstruction Loss: Mean Squared Error (MSE)
Total Loss: Weighted sum of classification and reconstruction losses
Expand
Reverse transformation to estimate the original image
TESTING
ERRORS
RESULTS

Figure 3: Project Summary Illustration

5.1 Model Architecture

The CNN model architecture consists of the following layers:

- **Convolutional Layer 1:** This layer uses 5 filters (kernels) with a size of 5x5. A kernel of this size is chosen to capture larger areas of the image, as we are more interested in broader color and intensity patterns rather than fine-grained features.

- **Max Pooling Layer 1:** A max-pooling layer with a kernel size of 2x2 follows the first convolutional layer. Max pooling reduces the spatial dimensions of the output, helping to retain only the most significant features and discard irrelevant parts of the image, such as uniform backgrounds in food and landscape images.

- **Convolutional Layer 2:** Another convolutional layer with 10 filters of size 5x5 further extracts complex patterns across the RGB channels that may indicate the applied filter.

- **Max Pooling Layer 2:** This max-pooling layer, also with a 2x2 kernel, further reduces the spatial dimensions, simplifying the feature maps and focusing on the most salient patterns.

- **Fully Connected Layers:** The flattened output from the second pooling layer is passed through a fully connected layer with 32 units, followed by the final output layer with 3 units, each representing one of the filter classes (grayscale, sepia, inverted).

- **Activation Function:** We apply the ReLU activation function after each convolutional layer. Although our current filters are linear transformations, we anticipate expanding the model to recognize non-linear transformations, making ReLU a suitable choice as it also helps avoid vanishing gradient issues.

Loss Function: We use categorical cross-entropy loss, appropriate for our multiclass classification task with three filter classes. Categorical cross-entropy allows the model to optimize for the probability of correctly predicting each filter type.

5.2 Complexity and Reproducibility

Our model has two convolutional layers, each followed by max pooling, and two fully connected layers. With this architecture, the model is lightweight and efficient enough to be trained with moderate computational resources while still capturing significant patterns related to filter transformations. This setup is straightforward to reproduce using standard deep learning frameworks like PyTorch or TensorFlow.

5.3 Quantitative Results

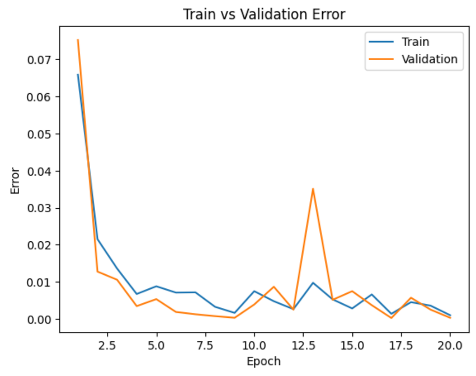
Our primary model achieved strong quantitative results, indicating that it is effectively learning the filter classification task. Key metrics are as follows:

- **Test Accuracy:** 99.97%, which demonstrates that the model performs extremely well in correctly classifying the filters applied to the test images.

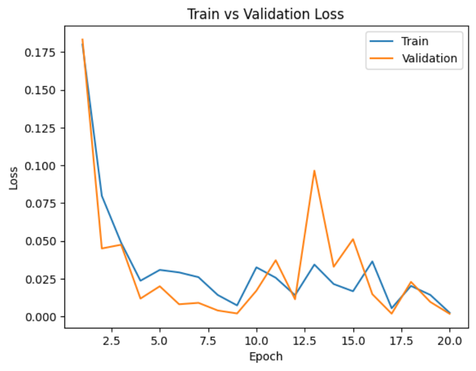
- **Test Error:** 0.03%, showing minimal misclassification in the test set.

- **Test Loss:** 0.00175, which is low, suggesting that the model has a high confidence in its predictions with little deviation from the true labels.

The training and validation error and loss curves also indicate good model performance. The training error and loss decrease consistently over epochs, while the validation error and loss closely track the training values, indicating minimal overfitting. This steady convergence demonstrates that the model generalizes well to unseen data.



(a) Training vs Validation Error



(b) Training vs Validation Loss

Figure 4: Training curves showing error and loss over epochs

5.4 QUALITATIVE RESULTS

The qualitative results are similarly promising. The model consistently performs well on images with distinct filter transformations, like grayscale, sepia, and inverted filters. It shows high confidence in these classifications, as reflected in the minimal error and low variance in the loss.

- **Error and Loss Curves:** The attached training curves show that both error and loss for validation and training decrease over time, with some minor fluctuations that stabilize toward the end. This pattern suggests that the model is learning effectively, balancing bias and variance well.
- **Confusion Patterns:** Given the high accuracy and low error rate, the model exhibits minimal confusion between filters, demonstrating that it can reliably distinguish even visually similar transformations, like grayscale and inverted. This robustness is crucial for applications where precise filter identification is necessary.

5.5 CHALLENGES AND IMPROVEMENTS

One of the challenges we encountered was tuning the architecture to retain only relevant features, as large portions of our images (e.g., the sky or background in landscapes) do not contribute meaningfully to filter classification. The max-pooling layers help mitigate this issue by reducing unnecessary background details. We also found that using larger kernel sizes allows the model to capture broad, colour-based transformations effectively.

In future iterations, we may consider adding more convolutional layers or experimenting with different pooling strategies to capture a wider range of filter types, especially if we expand to non-linear filters. We also plan to continue monitoring the loss curves and qualitative performance to refine the architecture as needed.