Image-Forgery-Detection-CNN

Sejal Chopra
40164708

Praveen Singh
40199511

Elvin Rejimone
40193868

Anushka Sharma
40159259

Abstract

Advancements in digital technology have led to an exponential growth in digital images and user-friendly image manipulation software, which has made image tampering a significant issue in recent years. Various computer vision and deep learning approaches have been proposed to tackle this issue, with some CNN architectures achieving remarkable accuracy rates of over 98%. However, the images used in these studies were easily recognizable by humans, and their performance on more challenging samples is still unknown. This study aims to address this gap by developing a CNN network inspired by a previous study and comparing its performance on two separate datasets, including one with more difficult tampered images. Additionally, the study measures the impact of data augmentation techniques and different hyperparameters on the classification performance. The results of the experiments highlight the significant influence of dataset difficulty on the performance obtained, emphasizing the need to consider more challenging samples in future research.

1. Introduction

In the digital age, the proliferation of devices such as smartphones and tablets has led to an unprecedented growth in the number of digital images available to us. Advances in image manipulation software have made it easier than ever before to alter digital images, with some manipulations being virtually undetectable to the human eye. Social media platforms have made the distribution of these manipulated images a trivial task, raising concerns about the potential for serious business or political issues that could arise from the spread of false information.

One key challenge in combating the spread of manipulated images is the difficulty in detecting them. While some manipulations, such as adjusting image contrast or smoothing skin, may be harmless, others could have serious consequences. For instance, a manipulated image could be used to portray a politician in a negative light or to create a false impression of a company's products or services.

To address this challenge, researchers have developed automated methods for detecting manipulated images. This task is commonly referred to as image forgery detection, and it involves breaking down the manipulation process into smaller sub-tasks. Three of the most common manipulations in literature are copy-move, splicing, and removal.

While there have been numerous approaches to image forgery detection in the literature, there are concerns about the accuracy and robustness of these methods. Traditional computer vision approaches have focused on developing algorithms that tackle some of the sub-tasks in isolation, while more recent research has focused on applying deep learning techniques to the same sub-tasks. Some of these methods have achieved high accuracy rates, but their performance on datasets with more difficult-to-detect manipulated images is unclear.

In this paper, we seek to address this gap in the literature by developing a convolutional neural network (CNN) architecture inspired by previous work, which achieves an impressive accuracy of 98.04% on the CASIA v1.0 dataset. We then apply the same network architecture to the Media Forensics Challenge 2016 dataset, which contains manipulated images that are significantly more difficult to recognize. Finally, we analyze the impact of hyperparameter tuning and data augmentation on the network's performance. Our results shed light on the robustness of CNN-based approaches to image forgery detection and offer insights into how to improve the performance of these methods in detecting more challenging manipulations.

2. Dataset

To develop effective image manipulation detection methods, it is essential to have access to high-quality benchmark datasets that contain a wide range of manipulations that are commonly encountered in real-

world scenarios. In this study, we evaluate the performance of our proposed image manipulation detection method on two publicly available datasets: the Casia v2.0 dataset and the NC16 dataset.

2.1 Casia

The Casia v2.0 [1] dataset is a widely used benchmark dataset for evaluating image forgery detection methods. It contains 12,622 images that are divided into two subsets: authentic and tampered. The authentic subset consists of 5,019 images that are collected from various sources such as Flickr and Google Images. The tampered subset consists of 7,603 images that are created by applying various image manipulation techniques such as copy-move, splicing, and removal.

The tampered images in the Casia v2.0 dataset are designed to be challenging and realistic, with a wide range of manipulations that are commonly encountered in real-world scenarios. The dataset is particularly useful for evaluating the robustness of image forgery detection methods against different types of manipulations.

2.2 Nc16

The NC16 (NIST-CMU) [2] dataset is another widely used benchmark dataset for evaluating image forgery detection methods. It contains 1,124 images that are divided into two subsets: authentic and tampered. The authentic subset consists of 562 images that are collected from various sources such as Flickr and Google Images. The tampered subset consists of 562 images that are created by applying various image manipulation techniques such as copy-move, splicing, and removal.

Compared to the Casia v2.0 dataset, the NC16 dataset is smaller and less challenging. However, it is still a useful benchmark dataset for evaluating image forgery detection methods, particularly for researchers who have limited computational resources or who are interested in evaluating their methods on a smaller scale.

3. Related Work

The problem of image forgery detection has been extensively studied in the literature. In recent years, deep learning approaches have shown great promise in detecting image forgeries. In 2016, Bayar and Stamm proposed a method based on convolutional neural networks (CNNs) to detect copy-move forgeries in digital images. Their approach achieved state-of-the-art performance on several benchmark datasets. Similarly, Cozzolino et al. proposed a CNN-based method to detect splicing forgeries in images. Their approach achieved high accuracy even when the spliced regions were small or blended with the background.

Other researchers have explored different types of image manipulation, such as retouching and inpainting. In 2018, Li et al. proposed a method based on generative adversarial networks (GANs) to detect face retouching in portrait photos. Their approach achieved high accuracy even when the retouching was subtle or applied by professionals. In 2019, Zhang et al. proposed a method based on GANs to detect inpainting forgeries in natural scenes. Their approach achieved high accuracy even when the painted regions were large or complex.

Image forgery detection is also an important problem in industry settings, such as social media platforms and forensic investigations. In 2018, Facebook introduced a tool called "Rosetta" that uses machine learning to detect text within images and videos. The tool can identify text in more than 20 languages and has been used to detect misinformation and hate speech on the platform.

Similarly, Adobe has developed several tools for detecting image manipulation within its Creative Cloud suite of products. For example, Photoshop includes a "Content-Aware Fill" feature that can automatically fill in missing parts of an image. However, the tool also leaves behind artifacts that can be detected using Adobe's "Forensic Toolkit" or other third-party software.

In this section, we reviewed some of the most relevant works in the field of image forgery detection, including both academic and industry-related research. Deep learning approaches have shown great promise in detecting various types of image manipulations, and several companies have developed tools to detect such manipulations within their products. However, the problem of image forgery detection is still far from being solved, and there is a need for further research in this area.

4. Methodology

The computer vision pipeline includes two key stages: feature learning and extraction. Initially, a CNN model is pre-trained based on labeled patch samples from training images. The positive patch samples are precisely drawn along tampered region boundaries in forged images, while negative ones are randomly sampled from authentic images. In the second stage, the pre-trained CNN extracts feature from the entire image using a patch-sized sliding

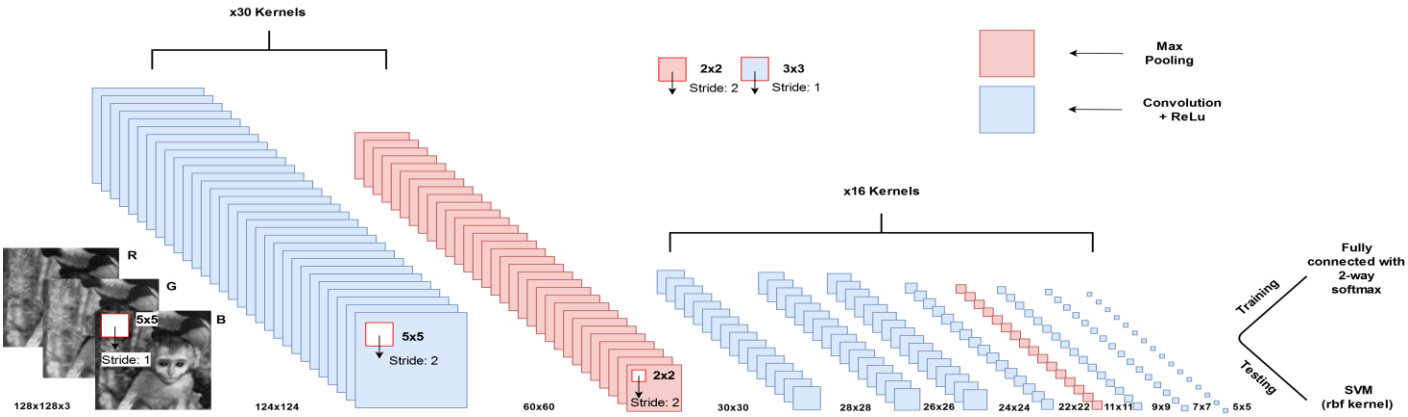

Figure 1: Architecture of the pipeline

window. The extracted features are then used to train an SVM model.

4.1. Network architecture.

The implemented CNN architecture in this study consists of nine convolutional and two max-pooling layers, illustrated in Figure 1.
If we denote $F^n(X)$ the feature map in the convolution layer n, with the kernel and bias defined by Wn and Bn respectively, the convolutional layer can be computed using the following formula (equation 1):

$$F^n(X) = pooling(f^n(F^{n-1}(X) * W^n + B^n)) \quad \dots(1)$$

Where $F^0(X) = X$ the input data, $f^n(.)$ a non-linear activation function applied to each element of its input and $pooling(.)$ the pooling operation which reduces the dimensions of the data via a max or mean operation.
The network takes a 128x128x3 patch as input, where 3 represents the RGB color channels. The first two convolutions output 3 and 30 kernels, respectively, with a 5x5 kernel size, followed by a pooling operation with a 2x2 filter. The subsequent eight layers contain 16 kernels each, using a 3x3 kernel size for the convolutions and a 2x2 filter for the max pooling. The convolutions have stride one, while the pooling operation has stride two. All convolutional layers use the ReLU activation function, and local response normalization is applied to each feature map before the pooling operation to enhance generalization. Specifically, the central value in each neighborhood is normalized by the values of its three neighboring channels.

4.2 Network weight initialization.

All the convolutional layers, except the second one, are initialized using Xavier initialization technique to avoid high values or values that vanish to zero, which helps to keep the variance the same with each passing layer. The second convolutional layer kernels are initialized using thirty SRM (steganalysis-rich model) high-pass filters, consisting of eight first, four second, and eight third-order filters. The higher the order, the more sensitive the filter becomes in detecting changes on the edges. Additionally, two square high-pass filters of size 3x3 and 5x5 are used to detect pixels with different values from their neighbors. Moreover, eight edge detection filters of size 3x3 (4) and 5x5 (4) are used to detect abnormal edges that do not blend in with their surroundings in the tampered images. This selection of filters helps to detect forged images better, as the tampered images may have abnormal edges that have drastic changes between one-pixel-region to another.

We shuffle the filters on all channels leading every RGB channel to have a different filter in every dimension. This has been empirically proven to increase performance due to its regularization effect.

4.3. CNN training

To enable the CNN architecture to identify and focus on local artifact regions and learn to recognize them, image patches were extracted from the dataset using a patch size of 128x128x3 for each color channel. The patches were extracted by applying a sliding window with a stride of eight over the whole image. Tampered patches were discriminated against from non-tampered ones by comparing each patch with the equivalent patch from the image mask and selecting those that contained part of the tampered region. Two random tampered patches per image were kept avoiding computational expense, and the same technique was applied to the equivalent authentic image to select two non-tampered patches randomly. To improve the generalization ability of the CNN and prevent overfitting, patches were augmented by rotating them four times at intervals of 90 degrees. This approach enables CNN to learn to recognize local artifact regions, improving its ability to detect tampered images.

After feature extraction, the resulting 400-dimensional features (5x5x16) are fed into a fully connected layer with a 2-way SoftMax classifier, using dropout with a probability of 0.5 to prevent overfitting. Unlike traditional CNN models with multiple fully connected layers, our architecture uses only one necessary fully connected layer at the end of the network. This is due to the large number of parameters required to train multiple fully connected layers, which can lead to overfitting, especially when the training set is relatively small, as in our task.

4.4. SVM training

Once the CNN network has been trained, the subsequent step is to train the SVM classifier. To achieve this, a sliding window with a stride of s is employed to scan the entire original and tampered images, extracting every possible (p x p) patch. This results in n new patches per image that are processed through the CNN, generating n feature representations Y_i (400-D) for each patch. However, these representations must be combined into a single $\hat{Y}[k]$ representation for each image before being inputted to the SVM. Max or mean pooling is performed on each dimension of Y_i across all n patches extracted from each image.

$$\hat{Y}[k] = Mean\ or\ Max\ \{Y_i[k] \dots Y_n[k]\}$$

Where $k \in [1{,}400]$ dimensions.
The SVM then uses the resulting 400-D feature vector to classify images as original or tampered.

5. Experiment

With the technical aspects of our implementation established, we proceed to lay down the workflow of the project followed by the tests performed on the results obtained. Our study's pipeline is developed using Python 3.9 and employed PyTorch[3] for implementing the CNN, along with scikit-learn's SVM[4] implementation. The entire code is available as a GitHub repository [5] which can be cloned and used. Thereafter, we conducted the subsequent experimental steps. Specifically, we first extracted CNN training patches according to Section 4.3 and utilized them for training the neural network. Next, we acquired new patches and their corresponding image features using mean fusion. Once we obtained the features, we trained and tested the SVM through stratified 10-fold cross-validation. We implemented the same steps for both of our datasets CASIA2 and NC2016, which are available online. But we experienced difficulties denoising the extracted masks for CASIA2, leading us to use masks from a public GitHub [6] repository. Luckily, no such issues were encountered with the NC16 dataset since masks were already included in the dataset.
Regarding the hyperparameters used in our experiments, we extracted image feature patches using a stride (s) of 128 and 1024 for CASIA v2.0 and NC16, respectively. The variation in stride is due to NC16's image size being about ten times larger than CASIA v2.0, necessitating a larger stride to maintain a consistent number of patches per image. All CNNs underwent training for 250 epochs, employing cross-entropy loss and Stochastic Gradient Descent (SGD) optimization. The SGD implementation utilized a momentum of 0.99, weight decay of 5x10^-4, and a decaying learning rate reduced by 10% every ten epochs. These parameters were chosen for every trained CNN as they enhanced convergence during preliminary tests. However, batch size and initial learning rate were adjusted for each experiment due to their significant impact on the network's training capabilities. Dropout was effective for the NC16 dataset but problematic for CASIA v2.0, hence it was only applied to the former. Network training occurred on Google Cloud, employing 4 vCPUs, 26 GB of RAM, and the NVIDIA TESLA K80 GPU. This configuration led to a training duration of 25 seconds per epoch for the CASIA v2.0 dataset, which contained approximately 20,000 patches without data augmentation. Lastly, regarding the SVM, we employed the RBF kernel for each run and optimized the C and gamma hyperparameters using exhaustive grid search. The tested values for each experiment were as follows: C: [0.001, 0.01, 0.1, 1, 10, 100, 1000] and gamma: [0.001, 0.0001]. The results obtained from the computation are displayed in the next section.

6. Results

The algorithm was run on both the datasets. We see different performance in the case of each of the dataset with the model performing slightly better on the CASIA2 dataset as compared to the NC 2016 dataset.
The best hyperparameters after grid search for CASIA2 and NC16 are {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
The table below shows the 10-fold cross validation accuracy scores:

CASIA2	**NC16**
0.839	0.761
0.869	0.752
0.941	0.769
0.871	0.920
0.877	0.892
0.896	0.857
0.853	0.901
0.899	0.901
0.937	0.857

Based on the above table we achieved a mean accuracy score of 0.887 with a standard deviation of 0.03 for

CASIA2 and a mean accuracy score 0.848 with a standard deviation of 0.06 for NC16.

The table below shows the confusion matrix for both the datasets.

CASIA2:

921 (TP)	194 (FP)
104 (FN)	1304(TN)

NC16:

100 (TP)	18 (FP)
13 (FN)	94 (TN)

While training both the models we could see that the accuracy for the model trained on CASIA2 dataset is gradually increasing and loss is gradually decreasing with each epoch but that is not the case for NC16 dataset. The model trained on NC16 shows a constant trend in both the accuracy and loss:

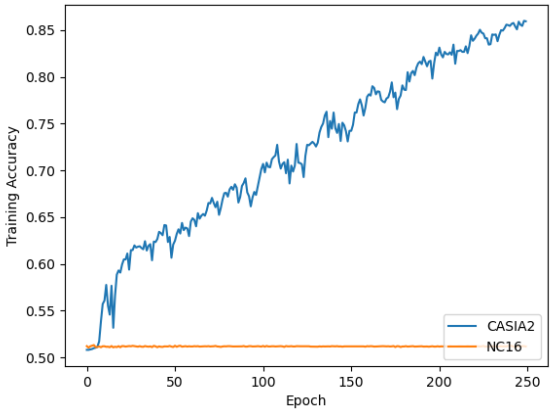

Figure 2: Accuracy trend

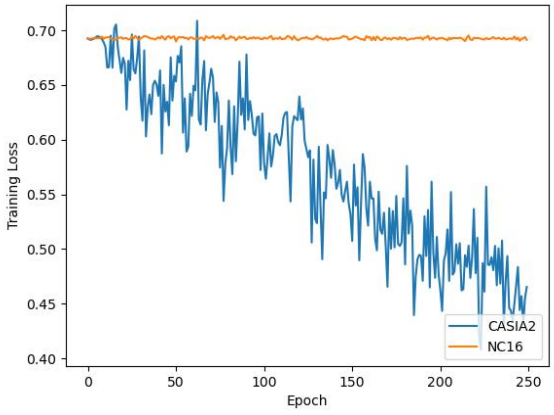

Figure 3: Loss Trend

7. Conclusion:

In our research, we conducted experiments using CNN to detect image forgery. Specifically, we utilized a CNN network to extract features from two datasets with varying levels of difficulty: CASIA v2.0 and NC16. The features we extracted were then used to train and test an SVM, which achieved an accuracy of 88.8% and 84.89% on CASIA v2.0 and NC16 respectively. These results support our hypothesis that the classification performance decreases as the samples become more challenging. However, our findings suggest that the implemented architecture may not be easily transferable to datasets with different underlying distributions. In conclusion, although there is still much work to be done in the field of image forgery detection, we believe that neural networks will soon be able to detect tampered images regardless of their difficulty.

8. Repository

The repository for this project can be found at ***repo***. This link will provide access to all the project's files, documentation, and version history. By visiting the repository, interested parties can view the codebase, contribute to the project, and gain a deeper understanding of its inner workings. We encourage anyone interested in the project to explore the repository and get involved in its development.

References

[1] "Casia2 dataset," www.kaggle.com. https://www.kaggle.com/datasets/sophatvathana/casia-dataset

[2] NIST Nimble 2016 dataset - National Institute of Standards and Technology (NIST), "The 2016 nimble challenge evaluation dataset,"https://www.nist.gov/itl/iad/mig/nimble-challenge-2017-evaluation, Jan. 2016 https://mfc.nist.gov/

[3] FirstName Alpher, Frobnication. *IEEE TPAMI*, 12(1):234–778, 2002.

[4] FirstName Alpher and FirstName Fotheringham-Smythe. Frobnication revisited. *Journal of Foo*, 13(1):234–778, 2003.

[5] FirstName Alpher, FirstName Fotheringham-Smythe, and FirstName Gamow. Can a machine frobnicate? *Journal of Foo*, 14(1):234–778, 2004.

[6] FirstName Alpher and FirstName Gamow. Can a computer frobnicate? In *CVPR*, pages 234–778, 2005.

[7] FirstName LastName. The frobnicatable foo filter, 2014. Face and Gesture submission ID 324. Supplied as supplemental material `fg324.pdf`.

[8] FirstName LastName. Frobnication tutorial, 2014. Supplied as supplemental material `tr.pdf`.

[7] Bayar, B., & Stamm, M. C. (2016). A deep learning approach to universal image manipulation detection using a new convolutional layer. In Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security (pp. 5-10).

[8] Cozzolino, D., Poggi, G., Verdoliva, L., & Riess, C. (2017). Recasting residual-based local descriptors as convolutional neural networks: an application to image forgery detection. In Proceedings of the IEEE International Workshop on Information Forensics and Security (pp. 1-6).

[9] Li, Y., Yang, X., Sun, P., & Qi, H. (2018). Celeb-DF: A large-scale challenging dataset for deepfake forensics. arXiv preprint arXiv:1811.00656.

[10] Zhang, Y., Liu, S., Dong, C., & Yang, Y. (2019). Detecting GAN-generated fake images using co-occurrence matrices of gradient magnitude and phase congruency features. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (pp. 0-0).

[11] Facebook Engineering Blog. (2018). Rosetta: Understanding text in images and videos with machine learning [Blog post]. Retrieved from https://engineering.fb.com/ai-research/rosetta-understanding-text-in-images-and-videos-with-machine-learning/

[12] Adobe Systems Incorporated. (n.d.). Adobe Forensic Toolkit [Software]. Retrieved from https://www.adobe.com/products/forensic-toolkit.html