Image Forgery Detection
CS4180 Deep Learning - Group 10

Achilleas Vlogiaris Arkajit Bhattacharya Kyriakos Psarakis Panagiotis Soilis Rafail Skoulos
Delft University of Technology

Abstract

The boom of digital images coupled with the development of approachable image manipulation software has made image tampering easier than ever. As a result, serious issues can emerge if the matter is not tackled accordingly. Throughout the years, various computer vision and deep learning approaches have been proposed to solve the issue. In particular, a few of the CNN architectures suggested, manage to predict images with an accuracy of more than 98%. That said, the images used in those studies are easily recognized by humans. This raises a crucial question: how would CNNs perform on more challenging samples? In this study, we develop a CNN network inspired by a previous study and answer this question by comparing its performance on two separate datasets. What is more, we measure the effect of a data augmentation technique and different hyperparameters on classification performance. Our experiments show that dataset difficulty can significantly influence the performance obtained.

1. Introduction

In our day and age, we have witnessed an unprecedented growth of digital images, a trend which has been made possible by the numerous devices around, such as smartphones and tablets [1]. Moreover, the development of user-friendly image manipulation software [2] that is available at reasonable prices, has made the manipulation of such content easier than ever. In particular, some of these images are tampered in such a way that it is impossible to detect even by humans [3]. Over and above, Social media platforms, such as Facebook, Instagram, and Twitter have turned the distribution of those images to the mass into a trivial task [4]. While tampering processes, such as image contrast adjustment and skin smoothing, are harmless [5], there are others that could create serious business or political issues [6].

Based on the aforementioned, it becomes apparent that the development of automated methods that detect such manipulated images is of paramount importance. The term used in literature for this issue is *image forgery detection*. An example set of images is given in Figure 1. More specifically, this task can be broken down into smaller sub-tasks depending on the way that the image has been manipulated. Three of the most common manipulations in literature [6] are:

- *Copy-move*: a specific region from the image is copy pasted within the same image.
- *Splicing*: a region from an authentic image is copied into a different image.
- *Removal*: an image region is removed and the removed part is then in-painted.

Figure 1: The Image Forgery Detection Problem

There have already been numerous approaches in the literature that address the task of image forgery detection. Traditional computer vision approaches have focused mainly on developing algorithms that tackle some of the aforementioned sub-tasks in isolation [7][8]. More recent research has focused on applying deep learning techniques to the same sub-tasks [4][9]. Furthermore, [10] looked into ways in which a Convolutional Neural Network(CNN) can learn how to extract the necessary features irrespective of the manipulation method. However, while some of these

methods seem to solve the image forgery detection problem, the accuracy achieved raises suspicions about their robustness. In particular, [9][10] use CNNs to classify images as tampered or authentic and achieve an accuracy of more than 98%.

When visually inspecting some of the images that were used in these studies, it becomes obvious that the images included in the benchmark datasets used are very easily recognized as tampered by humans. As a result, this raises an important question: how well do these methods perform on datasets where the tampered images are more difficult to recognize by humans? Will they achieve similar accuracy or will their performance degrade?

In this paper, we attempt to answer these key research questions. Firstly, we develop a CNN network architecture which is inspired by [9] that achieves an impressive accuracy of 98.04% on the CASIA v1.0 dataset. Following that, the same network architecture is trained and tested on the Media Forensics Challenge 2016 [11] dataset whose tampered images are significantly more difficult to recognize. Finally, we analyze the effect of hyperparameter tuning and data augmentation on the network performance.

2. Problem Statement

The goal of this study is to evaluate how the performance of an image forgery detection CNN varies based on the sample difficulty. To that end, we develop a classification pipeline inspired by the work of [9]. While their study achieves high accuracy, the CASIA datasets used to test their network on are manipulated in a way that is easy to recognize by humans. An example of an image included in CASIA[1] is available in Figure 2.

Figure 2: CASIA - Example of tampered image

For that reason, we want to test the behavior of such a CNN on a more challenging dataset. Our intuition is that its performance will degrade significantly. To validate this intuition, two datasets are selected which are used to train the CNN on. As an easy dataset, the CASIA v2.0 dataset is chosen over the v1.0 since it is considered more challenging [9] and also offers nearly seven times more samples. This is particularly important given that previous studies have shown that a high number of training samples is a requirement for a reasonably well-trained CNN [3][9]. The CASIA v2.0 dataset contains 12,622 images distributed 60-40 amongst authentic-tampered. Following the results from the CASIA dataset, the same architecture is trained and tested on the Media Forensics Challenge 2016 [11] dataset whose images are significantly more difficult to recognize. The NC16 dataset contains 1,124 images with a 50-50 distribution. An example of a tampered image from NC16 can be found in Figure 3.

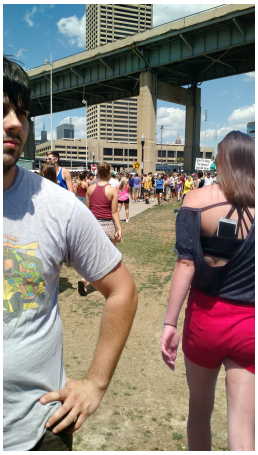


Figure 3: NC16 - Example of tampered image

Moving on to the evaluation of the model, the accuracy of the classifier during the test phase is the main evaluation metric. The reason behind this choice is that this is the main metric used by past deep learning studies [3][4][9][10] when evaluating the image forgery detection performance. Moreover, analyzing the confusion matrix of the model predictions and visually inspecting the misclassified images gives us more insights into its behavior.

3. Technical Approach

One of the main tasks of our study is to create a pipeline that is able to recognize tampered from authentic images. For that reason, we took inspiration from the architecture proposed by Y. Rao et al [9]. In particular, they propose a CNN that is used as a feature extractor that takes an image patch $X \in R^{p \times p}$ as input and outputs a feature representation $Y = f(X) \in R^K$, where K is the number of dimensions. This feature representation is then fed into an SVM classifier that predicts whether the features correspond to an original or tampered image. The network architecture and the procedure via which the CNN and the SVM are trained are detailed in the following sections.

[1] https://www.kaggle.com/sophatvathana/casia-dataset

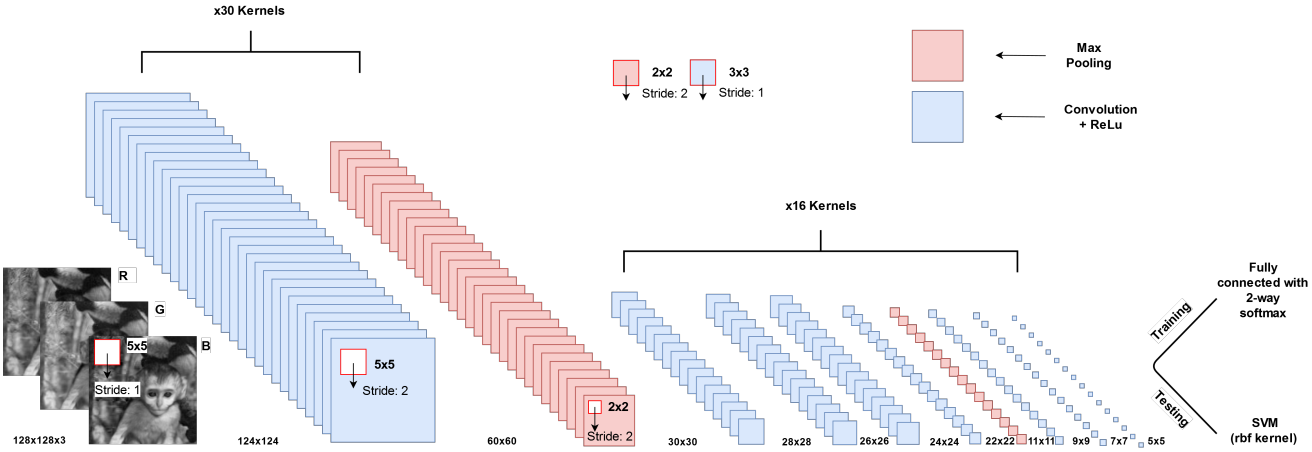


Figure 4: CNN network structure

3.1. Network architecture

A Convolution Neural Network (CNN) is a category of deep neural networks mainly used for image analysis. The basic structure of a CNN contains several convolutional layers followed by a fully connected layer(s) and a softmax classifier. Each convolutional layer is composed by a convolution, a non-linear activation, and a pooling. The input and the output of convolution layers are arrays which are called feature maps. If we denote $F^n(X)$ the feature map in the convolution layer n, with the kernel and bias defined by W^n and B^n respectively, the convolutional layer can be computed using the following formula [9]:

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

where $F^0(X) = X$ the input data, $f^n(\cdot)$ a non-linear activation function applied to each element of its input and $pooling(\cdot)$ the pooling operation which reduces the dimensions of the data via a max or mean operation.

The architecture implemented in this study is a CNN comprising of nine convolutional and two max-pooling layers as shown in Figure 4. The input size of the network is a 128x128x3 patch, where 3 represents the RGB color channels. The first two convolutions have a 5x5 kernel size and output 3 and 30 kernels respectively. These layers are followed by a pooling operation with a 2x2 filter. The next eight layers have 16 kernels, with a 3x3 kernel size for the convolutions and a 2x2 filter for the max pooling. These seven convolutions have stride one whereas the pooling operation has stride two. Additionally, every convolutional layer uses the ReLU activation function. It has to be mentioned that local response normalization is applied to every feature map before the pooling operation to improve generalization [9]. In particular, the central value in each neighborhood is normalized by the values of its three neighboring channels.

3.2. Network weight initialization

Apart from the second convolution, every other convolutional layer in our network is initialized using Xavier initialization [12]. The main concept is that it avoids high values or values that vanish to zero. This is achieved by keeping the variance the same with each passing layer.

Similarly to [9], the kernels of the second convolutional layer are initialized using thirty SRM high-pass filters proposed in [13]. The SRM filters used are eight first, four second and eight third order filters. As the order rises the more sensitive the filter is in detecting changes on the edges. Apart from the aforementioned, two square high pass filters of size 3x3 and 5x5 are used which can detect pixels that have different values from their neighbors. Finally, we use eight edge detection filters 3x3 (4) and 5x5 (4) that are the best in finding edges. The main reason behind their selection is that the tampered images could have abnormal edges which do not blend in with their surroundings. As a result, the change between one-pixel-region to the other would be drastic, thus enabling our model to better detect forged images.

In our work, we follow the suggestion of [9] to 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. To elaborate, given $c = [c1, c2, \ldots, c30]$ as the 30 filters, the weights can be calculated as:

$$W_j = [W_j^{3k-2} W_j^{3k-1} W_j^{3k}]$$

where $j \in [1, 30]$ featuremap, $k = ((j-1)\, mod\, 10) + 1$ and W_j^i corresponds to the i^{th} filter and the j^{th} featuremap. It has to be underlined that the W_j^i are zero filled until they reach a size of 5x5. An output example of these filters is showcased in Appendix A.

3.3. CNN training

In order to train the aforementioned CNN architecture in a way that it can focus on the local regions of the artifacts and learn to recognize them, image patches have to be extracted from the dataset used. The size of the extracted patches is 128x128x3, meaning that there is a 128x128 patch for every color channel. The extraction was performed by applying a patched-size sliding window with stride equal to eight for the whole image. Following that, the tampered patches are discriminated from the non-tampered ones. As far as the tampered patches are concerned, we compare each patch with the equivalent patch (from the same region of the image) of the mask of this image and keep the ones that contain part of the tampered region, as demonstrated in Figure 5. Moreover, we only keep two random tampered patches per image, as training the CNN with a huge amount of extracted patches would be computationally expensive. When it comes to the non-tampered patches, we apply the same technique but now on the equivalent authentic image and randomly select two of these patches. Finally, in order to improve the generalization ability of CNN and avoid overfitting, we augment the patches extracted by rotating them four times by a step of 90 degrees.

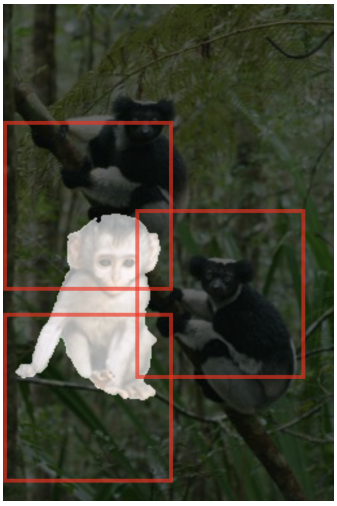


Figure 5: Patch extraction example

Following the aforementioned procedure, the patches are fed into the CNN which extracts a 400-D (5x5x16) feature representation of the patches. These features are then passed to a fully-connected layer with a 2-way softmax classifier that uses dropout [14]. More specifically, the neurons of the fully-connected layer are set to zero with a probability of 50%. Similarly to [9], we used only one fully-connected layer to reduce the number of parameters.

3.4. SVM training

After the training of the CNN network, the next step is to train the SVM classifier. For that purpose, we extract every possible $p \times p$ patch from both the original and the tampered images using a sliding-window with stride s to scan the whole image. This process results in n new patches per image which are passed through the CNN resulting in n feature representations Y_i (400-D). That said, these representations need to be fused into a single $\hat{Y}[k]$ representation for each image before being passed as an input to the SVM. Similarly to [9], max or mean pooling is applied on each dimension of Y_i over all the n patches extracted from each image:

$$\hat{Y}[k] = Mean\, or\, Max\, \{Y_1[k]...Y_n[k]\}$$

where k $\in$ [1, 400] dimensions. The resulting 400-D feature vector is then used by the SVM to classify images either as original or tampered.

4. Experiments

Having laid out the technical details of our implementation, we now move on to the tests performed to answer the research questions of this study. The pipeline of our study was implemented in Python 3.7 using PyTorch[2] to implement the CNN and the SVM[3] implementation provided by scikit learn. The code developed is publicly available on GitHub[4]. After the system implementation, the following experimental pipeline was laid out. To be more exact, first we extract the CNN training patches according to Section 3.3 and use them to train the neural network. Then, we extract new patches and the corresponding image features with the use of mean fusion as described in Section 3.4. Having obtained the features, the SVM is trained and tested using stratified 10-fold cross-validation. Before moving on, it has to be mentioned that we encountered issues while denoising the extracted masks for CASIA v2.0. As a result, we resorted to using the masks provided on a public GitHub repository[5]. The NC16 came with the masks included so no such problems were faced.

As for the hyperparameters used in our experiments, the patches required to extract the image features are obtained using a stride s of 128 and 1024 for CASIA v2.0 and NC16 respectively. The difference in stride stems from the fact that the image size in NC16 is approximately ten times larger than CASIA v2.0. Thus, we selected a larger stride to keep the number of patches extracted per image to be

[2] https://pytorch.org/

[3] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[4] https://github.com/kPsarakis/Image-Forgery-Detection-CNN

[5] https://github.com/namtpham/casia2groundtruth

same. All the CNNs are trained for 250 epochs using the cross-entropy loss and optimizing the network via Stochastic Gradient Descent (SGD). The SGD implementation uses momentum equal to 0.99, a weight decay of $5x10^{-4}$ and a decaying learning rate that is reduced by 10% for every ten epochs, similar to [9]. These parameters were selected for every CNN trained since they were found to improve its convergence during early tests. However, the batch size and the initial learning rate were tweaked accordingly for each experiment since they greatly influenced the network's ability to train. Moreover, while dropout worked for the NC16, it caused issues when training for the CASIA v2.0 dataset. Therefore, it was used for the former but not for the latter. The network training was carried out on Google Cloud using 4 vCPUs with 26 GB of RAM and the NVIDIA TESLA K80 GPU. This setup resulted in a training time of 25sec/epoch for the CASIA v2.0 dataset without the data augmentation which contains around 20,000 patches. Last but not the least, regarding the SVM we used the RBF kernel for every run and optimized the C and γ hyperparameters accordingly performing exhaustive grid search. The values that were tried out for each experiment were the following: C: [0.001, 0.01, 0.1, 1, 10, 100, 1000] and γ: [0.001, 0.0001].

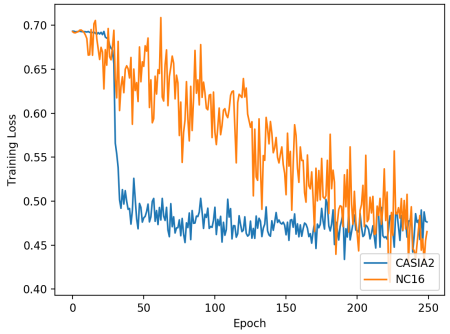
The rest of the section is structured as follows. First, we describe the different CNN networks that were trained. Following that, we perform a series of experiments that help us answer the question of how the network performs on datasets of varying difficulty and also its generalization ability. Then, we experiment with different hyperparameters and their effect on the classification. Finally, we visually inspect the misclassified samples and reason about why the network failed for these cases.

4.1. Network Training

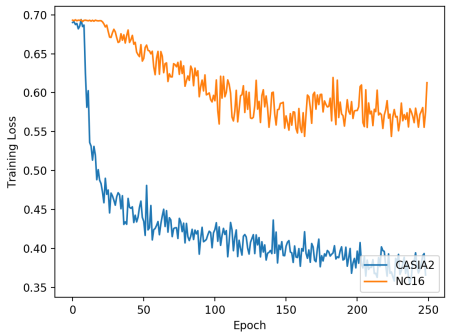
For our experiments, we used the network architecture described in Section 3 to train four different networks and compare their classification performance. More specifically, we trained a network with the patches from each of the two datasets, CASIA v2.0 and NC16, both with augmented (four rotations) and non-augmented data.

The training loss for each of the four aforementioned configurations is depicted in Figure 6. Regarding the training loss with the non-augmented data (6a), it is clear that after 250 epochs, the loss for both the two datasets is almost the same (≈ 0.45). However, the one of NC16 has gradually decreased while the CASIA v2.0 one has sharply decreased following the first epoch and already reached its minimum value after roughly 50 epochs. This behavior suggests that we could have applied early stopping for the latter, while we could have achieved better results for NC16 dataset if we had trained the network for more epochs.

When it comes to the networks with the augmented data, the loss for CASIA v2.0 after 250 epochs is much lower (≈ 0.4) than the equivalent of NC16 (≈ 0.6). In particular, the loss of CASIA v2.0 rapidly decreases in the first epochs and then it keeps decreasing at a lower rate until it becomes stable on the last epochs. On the other hand, the loss for NC16 gradually decreases for the first 100, until it reaches its lowest value (≈ 0.6). Thus, an early termination strategy may have reduced significantly the training time for NC16 dataset without affecting its performance.



(a) Non-augmented data - CASIA v2.0 vs NC16



(b) Augmented data - CASIA v2.0 vs NC16

Figure 6: Training loss comparison of CASIA v2.0 and NC16 with and non-augmented vs augmented data

The hyperparameters chosen for each different dataset are presented in Table 1. To elaborate, we tweaked the batch size and the learning rate for each CNN model to enable it to train properly. The values selected were such that the SGD made a similar amount of gradient steps for all the networks.

4.2. Dataset difficulty

In this section, we make a comparison of the classification performance between the two datasets that were used in this project.

Dataset	Batch size	Learning rate
CASIA v2.0	200	0.0005
NC16	32	0.001
CASIA v2.0 augmented	128	0.001
NC16 augmented	128	0.001

Table 1: CNN hyperparameters

4.2.1 CASIA v2.0 vs NC16

First, we trained the CNN by using the CASIA v2.0 dataset with an initial learning rate of 0.0005 and a batch size of 200 images. The best SVM hyperparameters that we trained on were $C = 1$ and $\gamma = 0.0001$. The previous settings resulted in a classification accuracy of $92.54 \pm 3.8\%$. Moreover, the corresponding confusion matrix was computed using a random 80-20 split and can be found in Table 2. In particular, the SVM managed to correctly classify 1,013 tampered and 1,308 original images, while it misclassified only 12 tampered images and 190 original images. Following that, we trained the feature extractor (CNN) with the second dataset with a learning rate of 0.001 and a batch size of 32 images. The optimal SVM parameters chosen after the grid search were $C = 100$ and $\gamma = 0.001$, resulting in an accuracy of $83.29 \pm 5.59\%$. An interesting thing is that the classification accuracy of our system on the more difficult NC16 is 10% lower than CASIA v2.0. By computing the confusion matrix in a similar fashion to CASIA v2.0, the output of Table 3 was obtained. More specifically, 100 tampered and 88 original images were correctly classified, while there were 13 false negatives (FN) and 24 false positives (FP). Both of the previous confusion matrices indicate that with this network architecture we obtain more FP than FN. However, given the nature of the image forgery detection problem we cannot conclude which of the two is more important since it depends entirely on the case study. The rest of the tests performed, pointed towards the same FP/FN behavior. Therefore, there is no need to provide the confusion matrix in each of the following experiments.

CASIA v2.0	Predicted Auth.	Predicted Tamp.
Actual Auth.	1,308	**190**
Actual Tamp.	12	1,013

Table 2: Confusion matrix - CASIA v2.0

NC16	Predicted Auth.	Predicted Tamp.
Actual Auth.	88	**24**
Actual Tamp.	13	100

Table 3: Confusion matrix - NC16

4.2.2 CASIA v2.0 vs NC16 - Augmented

Following that, we performed the same tests using the augmented datasets, where the images are rotated four times by a step of 90 degrees. The use of the augmentation improved classification accuracy for both datasets. In particular, for CASIA v2.0, accuracy increased from $92.54 \pm 3.8\%$ to $96.82 \pm 1.19\%$ while NC16 went up by a smaller amount from $83.29 \pm 5.59\%$ to $84.89 \pm 6.06\%$. The results of both the non-augmented and augmented tests are available in Table 4.

Data Augmentation	CASIA2	NC16
With	$96.82 \pm 1.19\%$	$84.89 \pm 6.06\%$
Without	$92.54 \pm 3.8\%$	$83.29 \pm 5.59\%$

Table 4: Augmentation effect - CASIA v2.0 & NC16

To conclude, based on the previous results the image forgery detection system behaved reasonably when we trained and tuned it for each corresponding dataset. What is more, the NC16 achieves a lower accuracy than the CASIA v2.0 dataset, regardless of the use of augmentation, as it contains samples whose tampered areas are carefully manipulated. Moreover, by comparing the classification with and without the use of rotations it becomes apparent that augmenting the data boosts the performance of the system irrespective of the dataset used. That said, its use increases the CASIA v2.0 accuracy by 4.28% while for the NC16 the increase was only 1.60%.

4.2.3 Pre-trained versions

Another interesting test performed was to train the CNN on the CASIA v2.0 augmented dataset and test it with the NC16, as well as the other way around. These two tests enables us to reason about the generalization performance of the proposed system. To be more specific, we extracted the features for the NC16 dataset with the CASIA v2.0 pretrained CNN and then classified the images using the SVM. The results obtained clearly show that the proposed model generalizes poorly to new unseen data that have a different underlying distribution. In particular, the NC16 accuracy with the CNN network trained on the CASIA v2.0 was equal to 67.54% while the CASIA v2.0 using the NC16 network yielded an accuracy of 59.81%. That said, the NC16 accuracy had a standard deviation of 16.01% within the 10 folds, ranging from 37.16% to 86.60% meaning that its performance heavily depends on the test fold selected.

4.3. Hyperparameters effect

In this section we experiment with three different hyperparameters, namely the initial learning rate of the CNN, the

stride used to extract the patches for the feature extraction and the feature fusion technique used.

4.3.1 Different learning rates

First off, to investigate the effect of the CNN learning rate in the system performance, we train three networks with different learning rates. It is important to mention that for these experiments we used the patches extracted from CASIA v2.0 without data augmentation. The main reasons behind this choice are that the network performs better on CASIA v2.0 and that it is computationally cheaper to train it for this version than the augmented one since it requires almost 75% less time. The results for learning rate values of 0.0001, 0.0005 and 0.001 are demonstrated in Table 5.

LR	CNN accuracy (%)	Test accuracy (%)
0.0001	88.14	92.42 ± 3.79
0.0005	84.53	92.54 ± 3.81
0.001	84.65	92.35 ± 3.48

Table 5: Learning rate comparison - CASIA v2.0

Based on the previous results, we conclude that despite the difference on the training accuracy of CNN for the three different learning rates, the test accuracy obtained by the SVM has insignificant differences.

4.3.2 Mean vs Max feature fusion

The next parameter we experimented with, is the method we use to combine the values of all the patches for each of the 400 features. In particular, the two methods we tried are mean and max fusion, as explained in Section 3.4. The dataset we used is CASIA v2.0 with augmentation given that it is the setup that achieves the highest accuracy so far. To add to that, the same hyperparameters used to obtain the best test accuracy are used. By comparing the performance of the mean vs max fusion, the results in Table 6 were obtained.

Fusion method	Test accuracy (%)
Mean	96.82 ± 1.19
Max	96.90 ± 0.40

Table 6: Mean vs max fusion - CASIA v2.0 augmented

As we can observe, the difference in terms of accuracy is negligible (0.08%). However, the standard deviation for max fusion method goes down by a significant amount.

4.3.3 64 vs 128 feature fusion stride

Finally, we also experimented with different stride sizes when we extracted the patches for the testing phase. Similar to the previous experiment, we used the augmented patches of CASIA v2.0 and the model hyperparameters that provided the best results so far. As as result, we selected the version which uses max fusion and compared two different stride values, namely 64 and 128. The classification results obtained are depicted in Table 7.

Stride size	Test accuracy (%)
64	96.83 ± 0.56
128	96.90 ± 0.40

Table 7: Stride 64 vs 128 - patch extraction during testing

Based on the previous results, we can conclude that these two stride values do not influence the performance of the classifier significantly since the accuracy and standard deviation are almost the same for both cases.

4.4. Misclassified samples - Visual inspection

In order to understand which samples the network has trouble in classifying, we observe the false negative and false positive annotations on both datasets. To begin with, the false positives in both datasets have some common properties, namely: motion blur, blurry depth of field, fog, spider webs, sun reflection on the lens and reflections on non-smooth materials. These lead us to assume that the model classifies an image as tampered if a blured sub-region exists. On the other hand, the false negatives share one common trait which is the effort done to smooth the edges of tampered regions. To elaborate, the shading and color scheme match the rest of the image so well that it is impossible to tell the regions apart without looking at the masks provided. The aforementioned findings can be clearly seen in Figure 7. In particular, the image on the left is a FP where the ball is depicted with motion blur and causes the image to be classified as tampered. On the other hand, the one on the right is a well forged image with smoothing on the edges, color correction and proper shading. These factors cause the network to have issues in recognizing this image as tampered.

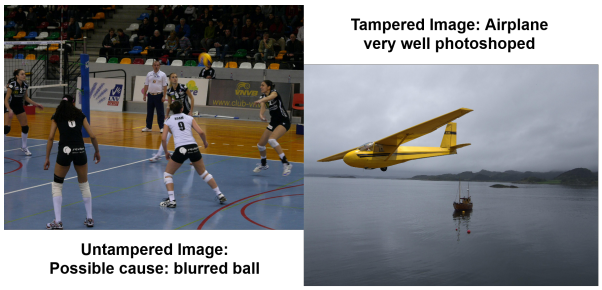


Figure 7: Misclassification examples - left: FP, right: FN

5. Discussion

Based on the experimental results provided, there are several questions that can be answered. First and foremost, how does a CNN perform on tampered images that are more difficult to recognize by humans? The answer to that question is that its performance degrades significantly. Based on the results from Section 4.2, the classification performance decreases substantially both when using augmented and non-augmented datasets. To be more specific, the accuracy on the more challenging NC16 dataset is 9.25% lower compared to CASIA v2.0 with their difference extending to 11.93% when augmenting the datasets with rotated images. This finding is in line with our intuition that the performance should be worse when the tampered regions are smoother and more difficult to recognize.

It has to be underlined that the CASIA v2.0 and NC16 have a couple of significant differences that might skew the results of our study. To be more specific, the NC16 dataset only contains 1,124 images compared to the 12,622 of CASIA v2.0. Given that neural networks are known to require a lot of data to yield a high accuracy, this could be part of the reason for the performance difference noticed. Moreover, the image size of the two datasets is significantly different. More specifically, the images in NC16 are around 10 times larger than those in CASIA v2.0. Given that the patch size extracted is the same in both datasets (128x128px), the patches in NC16 might only focus on local tampered regions which do not contain enough information about the overall manipulation to achieve a high accuracy. The aforementioned reasons could also partially explain why the network fails to generalize, as shown in Section 4.2.3.

Another interesting observation in our study is that data augmentation does indeed increase the performance of the network regardless of the dataset used. This finding is in line with past studies which use such techniques to artificially create more training data. However, designers have to also weight the extra time and cost required to train the CNN against the performance increase achieved. For instance, in the case of the NC16 dataset where the accuracy only increases by 1.60%, one could decide whether it is worth spending the extra time to train the network in a real life scenario.

Moving on to the system hyperparameters, we realized first hand their crucial importance on the classification performance. What became apparent during our efforts to train the neural networks is that without selecting the proper hyperparameters, the CNN either intially converges and then diverges or that it does not converge at all. To elaborate, settings such as batch size, learning rate and the use of dropout created serious issues when trying to train the aforementioned CNN since small changes in their values resulted in different behaviors. On the contrary, as shown in Section 4.3 the stride and the fusion method selected have an insignificant effect in the classification accuracy. Moreover, the hyperparameters of the SVM were also shown to significantly influence the classification performance. For an instance, when using the SVM to classify NC16 images with features extracted from a CNN trained on non-augmented images, a change of the γ value from 0.0001 to 0.001 increased the accuracy from 76.44% to 83.29%.

To conclude, we would like to lay out the future work arising from our study. Firstly, the CASIA v2.0 CNNs trained underlined the need to implement early stopping during the network training. To be more specific, all of them converged fairly quickly and plateaued at a specific cost value, suggesting that the use of early stopping would save precious training time/cost while achieving the same or even better performance by avoiding overfitting. Over and above, the use of two datasets with varying difficulty but similar sample and image sizes would assist in validating the results presented in this study. Another option would be to combine different datasets with forged and original images to create a larger training dataset for the CNN. As a result, more data with varying underlying distributions would be available, thus improving the generalization ability of the CNN model. That said, the differences in image sizes and image tampering techniques pose a challenge in creating a "one size fits all" network that can learn the feature representations for all types of sample variations. Last but not least, it would be interesting to look into ways to extend CNNs such that they can detect tampered images irrespective of the manipulation technique applied to them. An example of one such study is the one performed by [10].

6. Conclusion

In this work we experimented with using a CNN in the image forgery detection task. More specifically, we used a CNN network to extract features from two datasets of varying difficulty, namely CASIA v2.0 and NC16. Furthermore, the extracted features were then used to train and test an SVM, achieving an accuracy of 96.82% and 84.89% on CASIA v2.0 and NC16 respectively. These results validate our intuition that the classification performance decreases the more challenging the samples are. What is more, our study has shown that image tampering can be detected with an accuracy of more than 84% even if done by professionals. However, according to our findings the implemented architecture does not easily generalize to datasets with different underlying distributions. To conclude, while there is surely a lot of work still to be done in the image forgery detection domain, we believe that neural networks will be able to detect tampered images regardless of their difficulty in the near future.

References

[1] Jason Bunk, Jawadul H Bappy, Tajuddin Manhar Mohammed, Lakshmanan Nataraj, Arjuna Flenner, BS Manjunath, Shivkumar Chandrasekaran, Amit K Roy-Chowdhury, and Lawrence Peterson. Detection and localization of image forgeries using resampling features and deep learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1881–1889. IEEE, 2017.

[2] Ghulam Muhammad, Munner H Al-Hammadi, Muhammad Hussain, and George Bebis. Image forgery detection using steerable pyramid transform and local binary pattern. *Machine Vision and Applications*, 25(4):985–995, 2014.

[3] Jawadul H Bappy, Cody Simons, Lakshmanan Nataraj, BS Manjunath, and Amit K Roy-Chowdhury. Hybrid lstm and encoder-decoder architecture for detection of image forgeries. *IEEE Transactions on Image Processing*, 2019.

[4] Ying Zhang, Jonathan Goh, Lei Lei Win, and Vrizlynn LL Thing. Image region forgery detection: A deep learning approach. In *SG-CRC*, pages 1–11, 2016.

[5] Xudong Zhao, Shilin Wang, Shenghong Li, and Jianhua Li. Passive image-splicing detection by a 2-d noncausal markov model. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(2):185–199, 2014.

[6] Peng Zhou, Xintong Han, Vlad I Morariu, and Larry S Davis. Learning rich features for image manipulation detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1053–1061, 2018.

[7] A Jessica Fridrich, B David Soukal, and A Jan Lukáš. Detection of copy-move forgery in digital images. In *in Proceedings of Digital Forensic Research Workshop*. Citeseer, 2003.

[8] Weiqi Luo, Jiwu Huang, and Guoping Qiu. Robust detection of region-duplication forgery in digital image. In *Proceedings of the 18th International Conference on Pattern Recognition-Volume 04*, pages 746–749. IEEE Computer Society, 2006.

[9] Yuan Rao and Jiangqun Ni. A deep learning approach to detection of splicing and copy-move forgeries in images. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6. IEEE, 2016.

[10] Belhassen Bayar and Matthew C Stamm. 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*, pages 5–10. ACM, 2016.

[11] Haiying Guan, Mark Kozak, Eric Robertson, Yooyoung Lee, Amy N Yates, Andrew Delgado, Daniel Zhou, Timothee Kheyrkhah, Jeff Smith, and Jonathan Fiscus. Mfc datasets: Large-scale benchmark datasets for media forensic challenge evaluation. In *2019 IEEE Winter Applications of Computer Vision Workshops (WACVW)*, pages 63–72. IEEE, 2019.

[12] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[13] Jessica Fridrich and Jan Kodovsky. Rich models for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security*, 7(3):868–882, 2012.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

A. SRM Filters

In Figure 8 we showcase 2 out of the 30 high pass SRM filters that were used in our project. These filters are able to detect lines, edges, gradients of multiple orders.

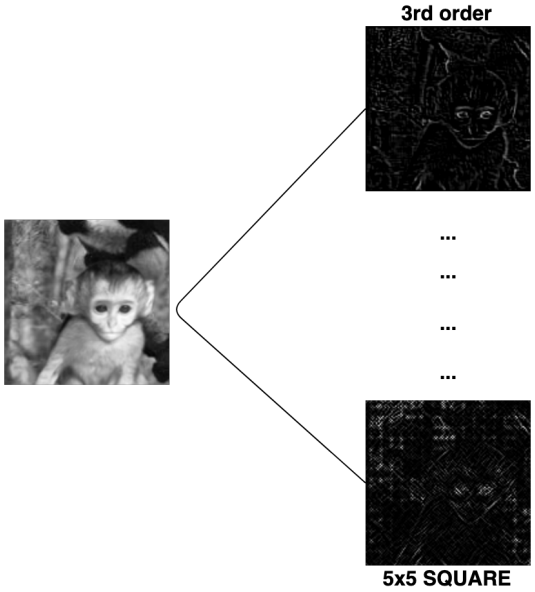


Figure 8: Example output of two SRM filters