# Deep Painter: *Painter Classification Using Deep Convolutional Autoencoders*

Project Paper, Neural Networks & Deep Learning

Alex Angus (ala2197) | Gunnar Thorsteinsson (gt2447)


## Abstract

*This project utilizes convolutional autoencoders (CAE) to classify paintings by artists. CAEs have been demonstrated to excel at small dataset classification tasks and are a robust supplementary tool to painting authentication. Mitigating overfitting and general debugging of the models took up a large portion of the students' time, but was eventually resolved. The original paper's accuracy was not perfectly matched (73% versus 96.52%). However, we deem the results satisfactory as the method was also applied to a more extensive dataset (2440 paintings by ten artists), resulting in accuracies that are significantly higher than chance.*


## List of Figures


1 Illustration of a Convolutional Autoencoder 3
2 Original Image vs Autoencoded . . . . . 4
3 Flowchart: High-level . . . . . 4
4 Flowchart: Data Preprocessing . . . . . 4
5 Flowchart: Autoencoder . . . . . 4
6 Flowchart: CAE Classifier . . . . . 4
7 CAE accuracy . . . . . 6
8 CAE accuracy for 3 painters . . . . . 6
9 CAE loss . . . . . 6
10 Confusion matrix of 3 painter classification 6
11 Confusion matrix of 10 fold validation for 3 painter classification . . . . . 7
12 Original image vs CAE . . . . . 7
13 CAE results . . . . . 8


## List of Tables


I Autoencoder Architecture . . . . . 4
II CAE Classifier Architecture . . . . . 4
III Individual Contribution . . . . . 7


## List of Algorithms


1 Image Preprocessing . . . . . 3
2 Autoencoder . . . . . 3
3 CAE Classifier . . . . . 3


## I. Introduction

Painting classification is the challenge of assigning an artist to a particular painting. It is closely related to painting authentication and can, as such, be utilized as a supplementary tool [1].

Convolutional Autoencoders (CAE) have been demonstrated to provide superior validation accuracy over other neural networks where a limited number of training samples is available [1], which is applicable for painting classification as artists generally produce on the order of hundreds of paintings over their career; a prolific painter might have on the order of several thousand.

## II. Original Paper Summary

The original paper by David and Netanyahu [1] proposes a novel approach to painting classification without the use of manual preprocessing or feature extraction. A deep convolutional autoencoder is trained on a set of 120 paintings by Van Gogh, Rembrandt, and Renoir. Then, the trained encoding portion of the autoencoder is used to initialize a supervised convolutional neural network for classification. This novel approach improves the incumbent record accuracy from 90.44% to 96.52%.

### A. Methodology of the Original Paper

Convolutional Neural Networks (CNN) have been proven to be incredibly effective tools in their application to computer vision problems [2], but these networks often need thousands of images for effective training. This poses a problem in using a CNN for the classification of paintings because only a small number of paintings are available per painter. The original paper aims to alleviate this problem by first training a convolutional autoencoder to generate a latent space representation of the images. The purpose of the latent space representation is to condense as many of the essential features as possible into a smaller

representation, with the hopes that the essential features of each image for classification will be captured. The autoencoder is evaluated on how well it can reconstruct the original image from this latent representation; therefore, capturing essential features in the latent representation is a valid assumption if the autoencoder is able to accurately encode and decode images. A diagram of the autoencoding architecture, taken from [1], is shown in Figure 1.

To train their autoencoder, David & Netanyahu used 5000 randomly selected images of paintings by various artists from WikiArt. To train and evaluate their classifier, they used a 120 image dataset with 40 paintings by each of the three considered artists. A table summarizing this test set is found in the original paper [1], making a direct comparison a possibility.

### B. Key Results of Original Paper

Using this novel approach, the authors improved the incumbent standard method's accuracy of painting classification from 90.44% to 96.52% on this limited dataset, which amounts to a 63% error rate reduction.

## III. Students' Project Methodology

This section describes the workflow and pseudo-code of our software, with appropriate references to files in the project repository.

### A. Objectives and Technical Challenges

The objective of this project is twofold; firstly, and most importantly, to replicate the original paper results, and secondly, to generalize the task for a more extensive dataset with ten classes versus the original three.

We anticipated that overfitting would be a problem, which turned out to be the case and ended up being the most time-intensive part of the project.

Additionally, as the dataset used in [1] was not available, we were tasked with generating our own painting images dataset. To accomplish this we designed a web scraping script to programmatically download painting images from [3]. An overview of the functionality of this data generation script is outlined in the Jupyter Notebook *RetrievingData.ipynb* which can be found in [4]. Also outlined in *RetrievingData.ipynb* are the image standardization functions used for creating sets of images with uniform aspect ratios and pixel resolutions.

### B. Problem Formulation and Design Description

An effort was made to replicate the original paper as closely as possible by manually collecting painting images from the original three painter dataset outlined by the table in [1]. This hand picked set we refer to as the test set, and was used to make the most direct comparison to the original work. A randomly selected dataset of 2440 pictures was used for the unsupervised autoencoder training, or one-half the size of the data used by the original paper. This could not be perfectly copied as the original unsupervised dataset was not listed. Using more than 2400 images resulted in memory issues; see section IV.

## IV. Implementation

This section describes the network architecture in detail, with supplementing diagrams and flow charts. The front-end coding was performed in Jupyter Notebook with back-end utility functions written in Python; see code in supplemented GitHub repository [4]. The models were built on the powerful TensorFlow machine-learning library, version 2.2.0 [5], and hosted on Google Cloud Platform (GCP) to cut down training times by utilizing NVIDIA K-80 GPU.

### A. Software Design

The Autoencoder was set up with the same architecture as the original paper, see Fig. 1 and Table I: Two consecutive layers of convolutional layers with max-pooling after each, followed by two sets of upsampling and deconvolution layers, see code in [6]. The CAE classifier contains the first half of the autoencoder, with the latter half consisting of fully connected dense layers, see in pseudo-code IV-A. Choosing the optimal amount of dense layers took some trial and error.

In terms of the process flow, the first step was to acquire the necessary data by web scraping from the visual art encyclopedia Wikiart [3], see proper code here [7]. The paintings were then resampled and normalized to a 256x256 resolution with downsizing and padding, pseudocode in IV-A, for flowchart see 4, and proper code in [8]. The autoencoder was trained on the unsupervised dataset, pseudocode in IV-A, proper code in [6], for flowchart see Fig. 5, and finally an example of an original image with an autoencoded one in Fig. 2. The resulting model and the latter half of the autoencoder architecture were then used in the CAE classifier, see tabular view in Table II, flowchart in Fig. 6, and proper code in [9]. The high-level process flow listed in this paragraph is visualized in Fig. 3. For a low-level description of the software design, refer to section IV.

There is a known design flaw in Tensorflow where the system does not release the GPU after running,

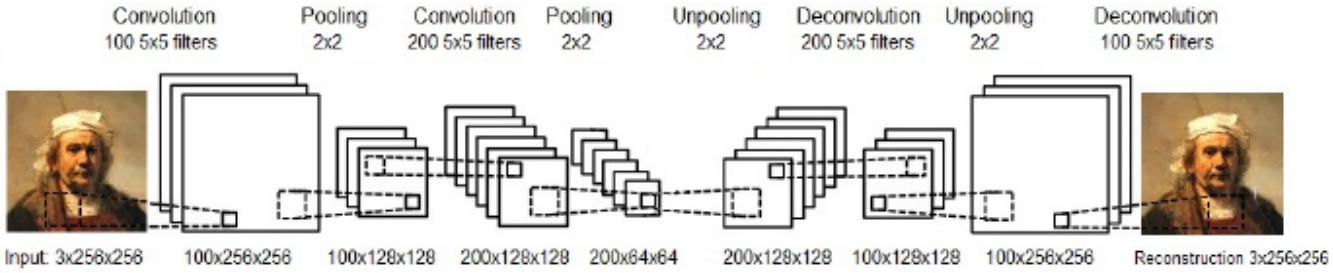


Fig. 1. Illustration of a Convolutional Autoencoder. Schema from original paper [1].

eventually making it run out of memory [10]. This made training a tedious process when training models on a large amount of data, as was the case in this project, as Jupyter had to be manually restarted from the command line after every instance. This problem was somewhat alleviated halfway through the project timeline by upgrading the virtual machine from N1-standard-2 (2 vCPU, 7.5 GB memory) to N1-highmem-8 (8 vCPUs, 52 GB memory).

We tried implementing manual pixel dropout for the training set to prevent overfitting, as suggested in the original paper. Nevertheless, overfitting was a severe problem; the training accuracy consistently diverged from the validation accuracy. It turned out that the encoder kept 'memorizing' the manual pixel dropout. We amended this by implementing a *SpatialDropout2D* layer instead of the manual dropout, see architecture in Table II, which dropped pixels randomly for each batch instead of fixed pixel dropouts over the entire training period.

For reasons not understood, the autoencoder trained using Kullback–Leibler divergence resulted in a blurry and green output, see Fig. 12. In comparison, the autoencoder trained with MSE as the loss function did not result in a color change (Fig. 2).

Before training the classifier, the CAE successfully reconstructs the image, but after classifier training, the CAE outputs mostly black images, corresponding to an abundance of encoding weights with values near zero, see Fig. 13. This results from the encoding portion of the CAE continuing to 'learn' during the supervised portion of the classifier training. The areas of the output that are not dark (right) are all the classifier had to infer which painter the painting belongs to; thus the classifier began to learn the specific artifacts (light areas) of the autoencoder rather than learning generalizable features about the paintings. This led to significant overfitting of the model to the training set and was rectified by freezing the weights of the encoding portion of the CAE before supervised classification training.

**Algorithm 1** Image Preprocessing

```
set artist names
set max number of images per artist
for every artist do
    scrape image from Wikiart
end for
for every image do
    standardize
    concatenate
    normalize
end for
```

**Algorithm 2** Autoencoder

```
load preprocessed data
set up architecture, see Table I
train model
visualize results
save model
```

**Algorithm 3** CAE Classifier

```
load preprocessed data
load Autoencoder
split Autoencoder
set up architecture, see Table II
train model
visualize results
save model
```

## V. Project Results

Throughout this project we consistently achieve accuracies that are below those reported in [1]. The reasons for these differences are likely numerous, and are discussed in the following section, but our models do achieve classification accuracies that are significantly

Fig. 2. An example of an autoencoded image (top) versus the original image (bottom). Artwork from [3].

TABLE I

AUTOENCODER ARCHITECTURE

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Input | (N, 256, 256, 200) | 0 |
| Conv2D | (N, 256, 256, 200) | 15200 |
| MaxPool | (N, 128, 128, 200) | 0 |
| Conv2D | (N, 128, 128, 200) | ∼1e6 |
| MaxPool | (N, 64, 64, 200) | 0 |
| UpSample | (N, 128, 128, 200) | 0 |
| DeConvolution2D | (N, 128, 128, 200) | ∼1e6 |
| UpSample | (N, 256, 256, 200) | 0 |
| DeConvolution2D | (N, 256, 256, 200) | ∼1e6 |
| DeConv2D_Reconstr | (N, 256, 256, 3) | 603 |

greater than chance (33% for three artist classification). Additionally, our CAE classifier models outperform other simpler machine learning painting classification models discussed in [1] such as nearest neighbor, SVM,

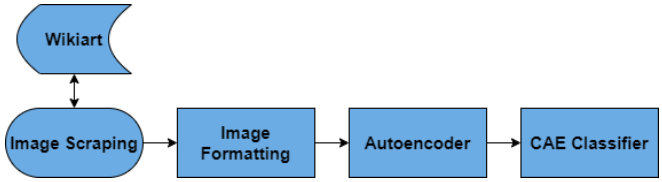


Fig. 3. High-level project flowchart.

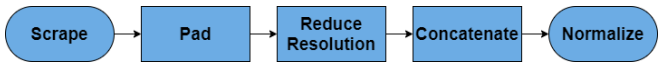


Fig. 4. Flowchart demonstrating the data preprocessing before the Autoencoder and CAE Classifier. See pseudo-code in IV-A

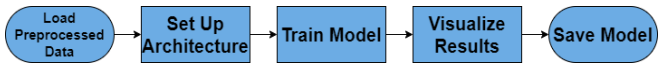


Fig. 5. Flowchart demonstrating the Autoencoder training process. See pseudo-code in IV-A and detailed architecture in Table I

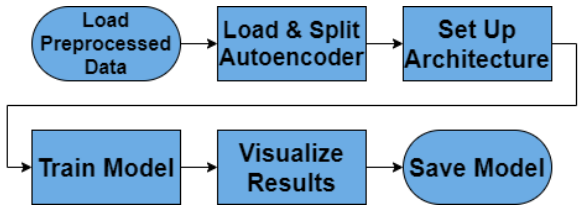


Fig. 6. Flowchart demonstrating the CAE classifier training setup. See pseudo-code in IV-A and detailed architecture in Table II

TABLE II

CAE CLASSIFIER ARCHITECTURE

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Input | (N, 256, 256, 200) | 0 |
| Conv2D | (N, 256, 256, 200) | 15200 |
| MaxPool | (N, 128, 128, 200) | 0 |
| Conv2D | (N, 128, 128, 200) | ∼1e6 |
| MaxPool | (N, 64, 64, 200) | 0 |
| Spatial_Dropout2D | (N, 64, 64, 200) | 0 |
| Dense_Flatten | (N, 819200) | 0 |
| Dropout | (N, 819200) | 0 |
| Dense | (N, 300) | 0 |
| Dropout | (N, 819200) | 0 |
| Dense | (N, 300) | 0 |
| Dense (Softmax) | (N, 3) | 903 |

and certain variations of genetic algorithms. To assess the performance of our classifier we used the scikit-learn 'classification_report()' function, which calculates the precision, recall, and F1 score of a set of predictions made by a classifier. Here we focus mainly on the F1 score which is defined as the harmonic mean of precision and recall.

Figure 7 shows the unsupervised training process of the CAE. Satisfactory encoding and decoding function-

ality was produced, and a greater than 90% decrease in validation loss was observed within 4 training epochs. Figure 8 shows the supervised training process of the CAE classifier for three painter classification of the test set and Figure 9 shows the mean squared error (MSE) loss of the same training process. In this instance an early stopping technique was implemented, where training would stop if validation accuracy surpassed 80%. This was done to prevent divergence of the validation loss and overfitting of the model. The confusion matrix in Figure 10 shows how well this particular model classified paintings of each artist. The matrix shows a perfect classification score for this model and validation set, but these results were not consistent. A better evaluation of how well a classifier performs is 10-fold cross validation, which was implemented similarly in [1]. In our 10-fold cross validation, 10 independent models were trained using 10 random 90% train, 10% validation splits of the test set. The models were then each evaluated in terms of precision, recall, and F1 score. Each metric was then averaged across all models for a more representative representation of the model's performance. Our model achieved precision of 73%, recall of 72%, and an F1 score of 72% under 10-fold cross validation. The confusion matrix representing these results is illustrated in Figure 11.

### *A. Comparison of the Results Between Original Paper and Students' Project*

The original paper had an accuracy of 96.52% for the three-artist, 120 image dataset. Our best performing model had an accuracy of about 73% for three artist classification. There are many probable reasons for this discrepancy, with different data being the most important. As mentioned previously, the original dataset was not available, thus we accumulated a new training dataset using a web scraping script and manually constructed, to the best of our ability, a test set closely resembling the one described in [1]. As the performance of computer vision models are highly dependent on the data used for training, the standardization of this newly constructed dataset is likely the biggest culprit in the relatively lower performance of our models. Specifically, it is not indicated in [1] how the dimensions of various painting images were standardized. We opted for a padding approach in which the painting image is surrounded by zero valued pixels such that the dimensions of the whole image is square (examples of this padding can be seen in Figures 2, 12, and 13). We chose this standardization method over cropping or augmentation of aspect ratios so as to preserve as much original information in the image as possible. Cropping would remove sections of the painting, and augmentation of aspect ratios would alter potentially important features in the paintings by changing the spatial relationships of different shapes. We hypothesized that the padding method would result in the best classification performance, but this hypothesis was not explicitly tested and it is certainly possible that a different standardization method was used in [1] with more success.

Similarly, while we attempted to replicate the CAE architecture described by [1] as closely as possible, we were constrained by the functional API of TensorFlow 2.2. A crucial component of the autoencoding architecture (Figure 1) is the unpooling layer. Unpooling layers are those in which pooling operations performed in the encoding portion of the CAE are 'undone'. In [1] a max unpooling layer is implemented, but no such layer exists in TensorFlow as of yet. We instead used the *UpSampling2D* layer with bi-linear interpolation to function as an unpooling analog. More information about the *UpSampling2D* layer can be found in the TensorFlow documentation [5]. It is not immediately evident what the effects of using upsampling instead of unpooling are on the quality of latent space representation generated by the autoencoder, but it is possible that this small architectural difference led to a large impact in terms of classification performance.

Details regarding the specific training methods used are also absent in the original work. The loss function and optimizer algorithm used in training the autoencoder are not discussed, thus we were tasked with finding a combination that successfully trained our version of the CAE. This process was largely done by informed (through reading various papers and articles) trial and error. We experimented with using Kullback–Leibler divergence (KLD: a measure of the difference between two probability distributions) as a loss function, but the results were unsatisfactory. Although a relatively low KLD loss was achieved, classification results using this encoder were poor. Upon inspection, we discovered the output of the CAE trained with KLD to be extremely blurry and green (Figure 12). After replacing the KLD loss function with a MSE loss function, the autoencoder output improved significantly (Example in Figure 2). The reasons for the KLD color shift are not understood, but this example highlights the importance of choosing the correct loss function for a given application, and makes it clear that a small difference such as this could

explain the discrepancies between the results of [1] and our own.

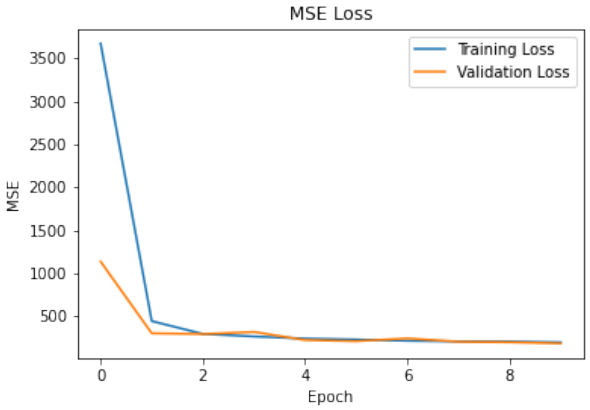


Fig. 7. Training and validation loss of the CAE.

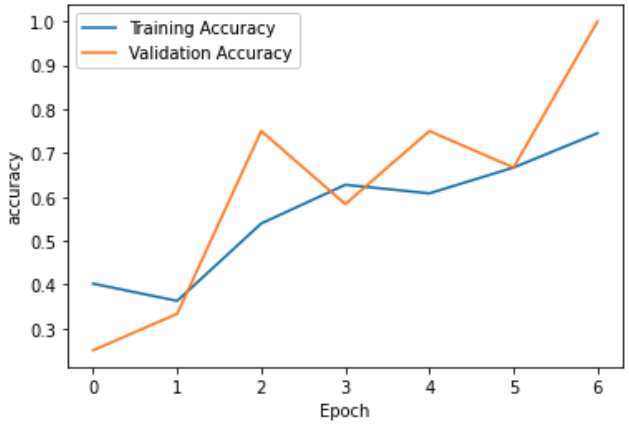


Fig. 8. Training and validation accuracy of the CAE classifier for 3 painter classification.

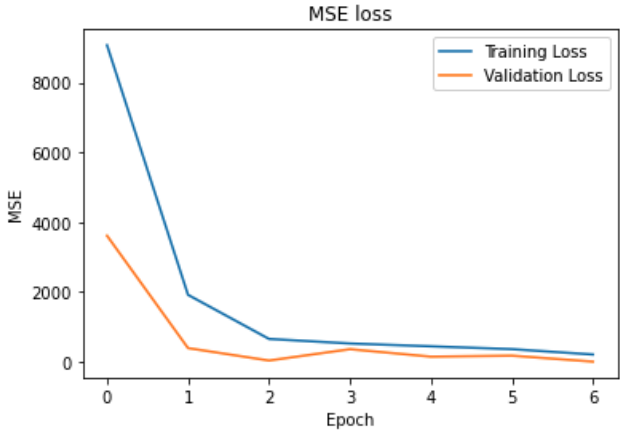


Fig. 9. Training and validation loss of the CAE classifier for 3 painter classification.

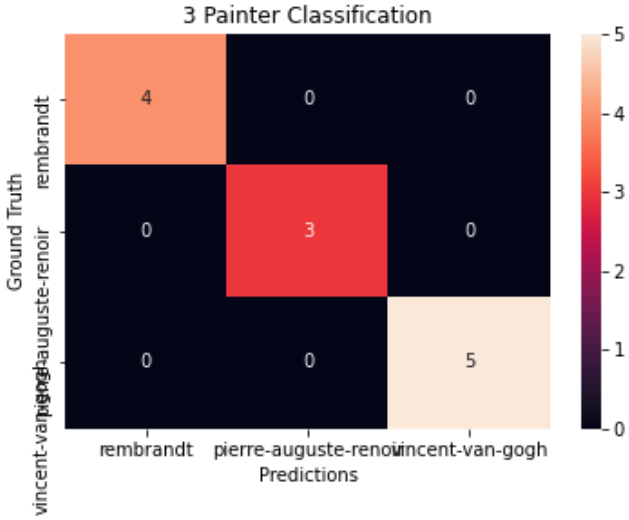


Fig. 10. Confusion matrix of 3 painter classification. Training was conducted on the 120 image test set where a random 90% was used for training and the other 10% used for validation.

### B. Discussion of Insights Gained

We weren't able to further reduce the difference between our validation accuracy and the original paper's, despite numerous hours of modifying hyperparameters. We hypothesize that this difference is due to the data used for training the autoencoder; by not having access to the 5000 image dataset, we cannot determine if the mismatch inaccuracy results from the training data mismatch, our inadequacies, or a combination of the reasons discussed in the previous section.

## VI. Conclusion

An autoencoder and a CAE classifier were trained to classify paintings, based on an approach developed by David & Netanyahu [1]. The CAE classification results, assessed using 10-fold cross validation, yielded an accuracy of 73%, using a comparable dataset as the original paper, which resulted an accuracy of 96.52%. This difference is partly attributed to the difference in the datasets used for training the autoencoder. We also applied the ten-artist, 2440 image dataset on the CAE classifier, resulting in validation accuracies of about 35%, roughly three and a half times better than chance.

We interpret these results as satisfactory, albeit not perfect. We consider this method ideal for small datasets with limits on augmentation and suggest fur-

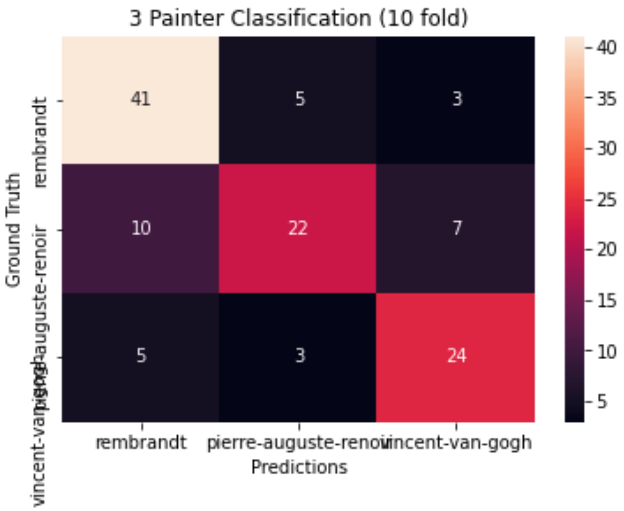


Fig. 11. Confusion matrix of 10 fold validation for 3 painter classification. Ten independent CAE classifiers were trained, each with a different random 90% of the test set. The remaining 10% was used to assess the performance of each classifier.

ther research on more data by more artists for optimization.

On a personal level, there are two things we would like to highlight as key takeaways. Firstly, patience is a virtue when debugging a piece of software one is doing for the first time. We had to spend numerous hours configuring the CAE classifier. Secondly, it was a great lesson in cooperating from idea to 'product' on software, as navigating the different project parts requires a good deal of communication and delegation.

*Acknowledgement*

This project was carried out without any outside assistance from TAs, colleagues, or other specialists. We consulted Tensorflow, and Numpy documentation [5, 11] and various Stack Overflow threads (see e.g., [12–15]) during setup and debugging, respectively, as is to be expected.

TABLE III

INDIVIDUAL CONTRIBUTION

| UNI | ala2197 | gt2447 |
|---|---|---|
| Last Name | Angus | Thorsteinsson |
| Scraping | 100 | 0 |
| Utils | 70 | 30 |
| Architecture | 70 | 30 |
| Training | 50 | 50 |
| Debugging | 50 | 50 |
| Report | 25 | 75 |

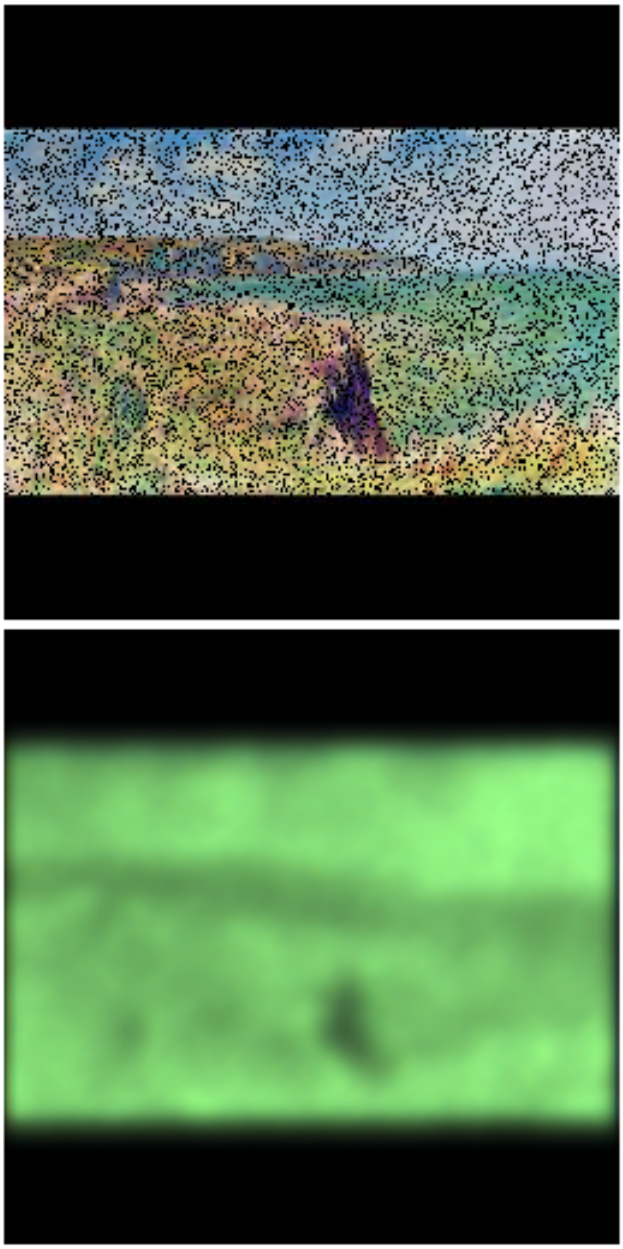

Fig. 12. Comparison between original image (top) and output of CAE (bottom) trained with Kullback–Leibler divergence as the loss function.

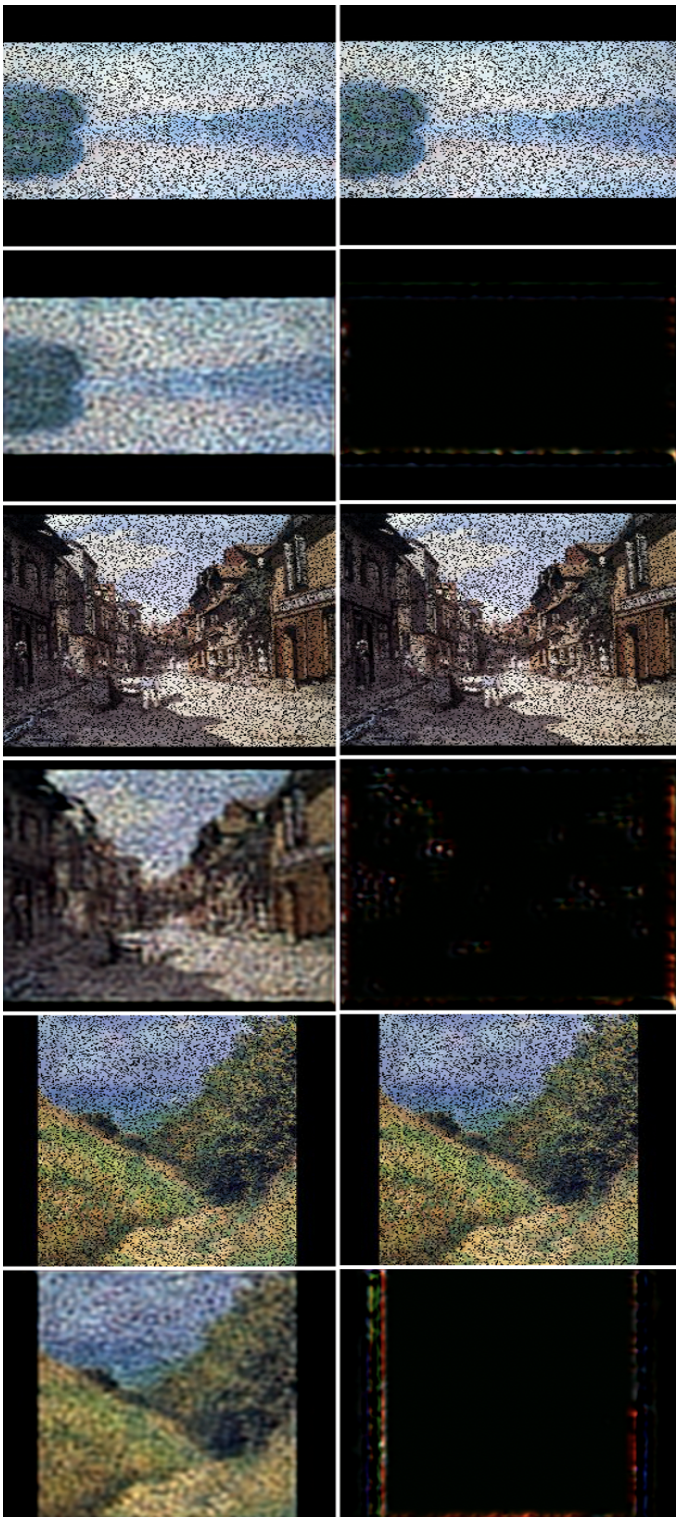

Fig. 13. Input (top) and output (bottom) of CAE. The left column shows the output of the CAE before training of the CAE classifier, and the right column shows the output of the CAE after training of the CAE classifier.

## REFERENCES


[1] Eli David and Nathan S. Netanyahu. "DeepPainter: Painter Classification Using Deep Convolutional Autoencoders". In: vol. 9887 LNCS. Springer Verlag, Nov. 2017, pp. 20–28. DOI: 10.1007/978-3-319-44781-0_3.

[2] Ian Goodfellow and Yoshua Bengio and Aaron Courville. "Deep Learning". In: MIT Press, 2016.

[3] *WikiArt.org - Visual Art Encyclopedia*.

[4] Alex Angus and Gunnar Thorsteinsson. GitHub Repository: *e4040-2021spring-project-aagt-gt2447-ala2197*.

[5] Google Inc. *TensorFlow Core API Documentation, v2.4.1*.

[6] Alex Angus and Gunnar Thorsteinsson. *e4040-2021spring-project-aagt-gt2447-ala2197/Autoencoder.ipynb*.

[7] Alex Angus and Gunnar Thorsteinsson. *e4040-2021spring-project-aagt-gt2447-ala2197/image_scrape.py*.

[8] Alex Angus and Gunnar Thorsteinsson. *e4040-2021spring-project-aagt-gt2447-ala2197/image_formatting.py*.

[9] Alex Angus and Gunnar Thorsteinsson. *e4040-2021spring-project-aagt-gt2447-ala2197/CaeClassifier.ipynb*.

[10] GitHub. *Tensorflow Issue #36465*.

[11] Numpy Community. *NumPy Reference Manual, v. 1.20.0*. 2021.

[12] Stack Overflow. *python - Clearing Tensorflow GPU memory after model execution*.

[13] Stack Overflow. *machine learning - ResNet: 100% accuracy during training, but 33% prediction accuracy with the same data*.

[14] Stack Overflow. *tensorflow - Understanding the ResourceExhaustedError: OOM when allocating tensor with shape*.

[15] Stack Overflow. *python - jupyter notebook's kernel keeps dying when I run the code*.