B.Tech Major Project Report

CSPE-40

on

CLEANING TEXT DOCUMENTS USING CONVOLUTIONAL AUTOENCODER

By

Vanshika(11710185)

Pratham (11710186)

Mayank (11710194)

Group No.: 17

Under the supervision of

Dr. Anoop Kumar Patel, Asst. Professor

DEPARTMENT OF COMPUTER ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY

KURUKSHETRA-136119, HARYANA (INDIA)

CERTIFICATE

We hereby certify that the work which is being presented in this B.Tech. Major Project (CSPE-40) report entitled "**Cleaning Text Documents using Convolutional Autoencoder**" in partial fulfillment of the requirements for the award of the **Bachelor of Technology in Computer Engineering** is an authentic record of our own work carried out during a period from January, 2021 to April, 2021 under the supervision of Dr. Anoop Kumar Patel, Asst. Professor, Computer Engineering Department.

The matter presented in this project report has not been submitted for the award of any other degree elsewhere.

Signature of Candidates:

VANSHIKA (11710185)

PRATHAM (11710186)

MAYANK (11710194)

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date: **29 Apr 2021**

Signature of supervisor:

Dr. Anoop Kumar Patel

(Assistant Professor)

TABLE OF CONTENTS

ABSTRACT

Text documents are digitised for various purposes like preservation, sharing, reducing waste, accessibility, cost etc. but there is another purpose that is being recognised in recent years due to advancement of technologies and that is 'Restoring'. Physical documents are easily contaminated due to environmental causes and mishandling. This results in deteriorating the document quality and its information. Our solution is based on removing the contaminations and improving the document quality by utilizing an unsupervised learning method which is convolutional autoencoder. The model is trained on custom made datasets which we have created ourselves. Our aim is to take a contaminated and noisy scanned text document as input and produce a clean and denoised document as output.

II. INTRODUCTION

In this project, we present you an approach for dirty document cleaning using a Convolutional Neural Network (CNN) algorithm. We will be using auto-encoders. Autoencoders are a type of unsupervised neural network that seek to accept an input set of data. Then, internally compress the input data into a latent-space representation (i.e., a single vector that compresses and quantifies the input) and reconstruct the input data from this latent representation (i.e., the output).

The dataset being used in this project is a self generated dataset all according to the requirements of our model. The dataset used contains images of smudgy as well as blurry documents and their corresponding clean documents.

The model generated takes an input which is first preprocessed in such a manner that it generates a good result in as low time as possible. Since the CNN model takes a 256px*256px resolution image as an input, the inputted document is first segmented in that manner initially and then each a 256px*256px grid is passed through the model via different threads for prediction of the output. The output is then stitched together in the respective order.

For the prototype that we are generating, we will be using Python programming language for writing the code and generating the CNN model. The libraries being used are Keras, Tensorflow, OpenCV2, Numpy and Matplotlib. Tensorflow is used for modelling deep neural networks, Keras acts as an interface for Tensorflow library, for image manipulation and operations we are using OpenCV2 and numpy and Matplotlib is used for visualising the inputs and outputs.

III. RELATED WORKS & LITERATURE REVIEW

The project contains a few references from different papers that helped us to understand and design our CNN model. Few of the topics that we worked upon are listed below:

1. *Convolutional Neural Networks (CNN):* In deep learning, a **convolutional neural network** (**CNN**, or **ConvNet**) is a class of deep neural networks, most commonly applied to analyzing visual imagery [1].

2. *Autoencoders:* "An autoencoder is a type of neural network that tries to learn an approximation to identify function using backpropagation. [2]". They are also known as auto-associators, are a class of feed-forward neural networks that are designed for performing dimensionality reduction and manifold learning [3].

Typically an autoencoder has two components:

2.1 *Encoder:* Accepts the input data and compresses it into the latent-space. If we denote our input data as ***x*** and encoder as ***E***, then the output latent-space represented by ***s*** would be ***s=E(s)***.

2.2 *Decoder:* Decoder accepts the latent space representation ***s*** and the deconstruct it into the original input. Suppose, we have decoder function ***D*** and output of decoder as ***o*** then we can represent the decoder as ***o=D(s).***

3. Further, a lot of research papers have been published recently using different approaches for tackling it.

Denoising Dirty Document using Autoencoder [1] which we are referring to and are focussing on improving their results.

DIVNOISING [5] which uses a convolutional variational autoencoder (VAEs) on 13 different datasets.

DeepErase [6] which cleans ink strokes over text document images.

Blind Image Denoising and Inpainting Using Robust Hadamard Autoencoders [7].

Training Stacked Denoising Autoencoders for Representation Learning [8].

IV. PROPOSED APPROACH

The method used for image cleaning and denoising is by using an unsupervised machine learning model known as Convolutional Autoencoders. Following is the algorithm to our proposed approach that we will be using for document cleaning.

1. *Dataset*

   We are using self generated datasets using various softwares to train our machine learning model according to our requirements. The dataset contains smudgy documents and their corresponding clean documents for training. Fig. 1 represents a sample of our dataset that is being used in our model.

   Size of the dataset: 1044 images of 256px*256px resolution each.

   Source of dataset: Self generated using Photoshop and MS Word.

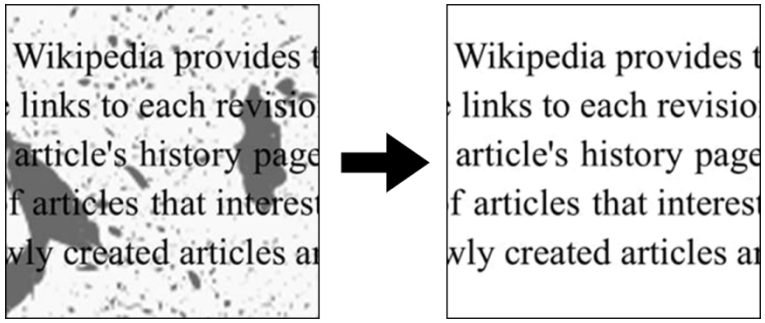


Fig. 1. Smudged to clean document dataset

2. *Model*

   Our algorithm uses a convolutional autoencoder model for smudge removal.

   Layers of our model include:

   Input Layer

   The input layer is the input of the whole CNN. In the neural network of image processing, it generally represents the pixel matrix of the image. [9]

   Since, in this case we are using 256px * 256px resolution input images, input shape of (256,256) is used in the input layer.

Convolutional Layer

The convolutional layer is used to extract image features. Low-level convolutional layer extracts shallow features (such as edges, lines, and corners). High-level convolutional layer further learns abstract features through the input of low-level features. The convolutional layer obtains multiple feature activation maps by convolving the convolution kernel of a specific size with the previous layer. [9]

In the *encoder*, we have used two convolutional layers with output channels 32 and 64 and each are followed by a MaxPooling Layer.

In the *decoder*, we have again used two convolutional layers but with output channels 64 and 32 and each are followed by a UpSampling Layer.

We are using ReLu (Rectified Linear Unit) as the activation function of each convolutional layer.

Pooling Layer

The introduction of a pooling layer is to reduce dimension and abstract the input image by imitating the human visual system. By sampling the convolved feature maps, the useful information of the image is preserved and the redundant data is removed, thus effectively preventing the overfitting problem and speeding up the computation speed. What's more, the pooling layer has feature invariance which can make the model more concerned with the presence of certain features rather than the specific location of the features and tolerate some small displacement of features. There are generally two kinds of operations: maximum pooling and average pooling. [9]

We have used maximum pooling operation after the convolutional layers of the encoder part.

Upsampling Layer

The Upsampling layer is a simple layer with no weights that will double the dimensions of input and can be used in a generative model when followed by a traditional

convolutional layer.

Therefore, it is used in a decoder and used after the two convolutional layers of the decoder.

Output Layer

The Output layer is responsible for producing the final predicted/constructed image. The output shape of the image is taken same as the input shape in the Input Layer i.e. (256,256). Sigmoid function is used as the activation function in this layer.

Fig. 2. shows the model architecture of the CNN autoencoder used in the project.

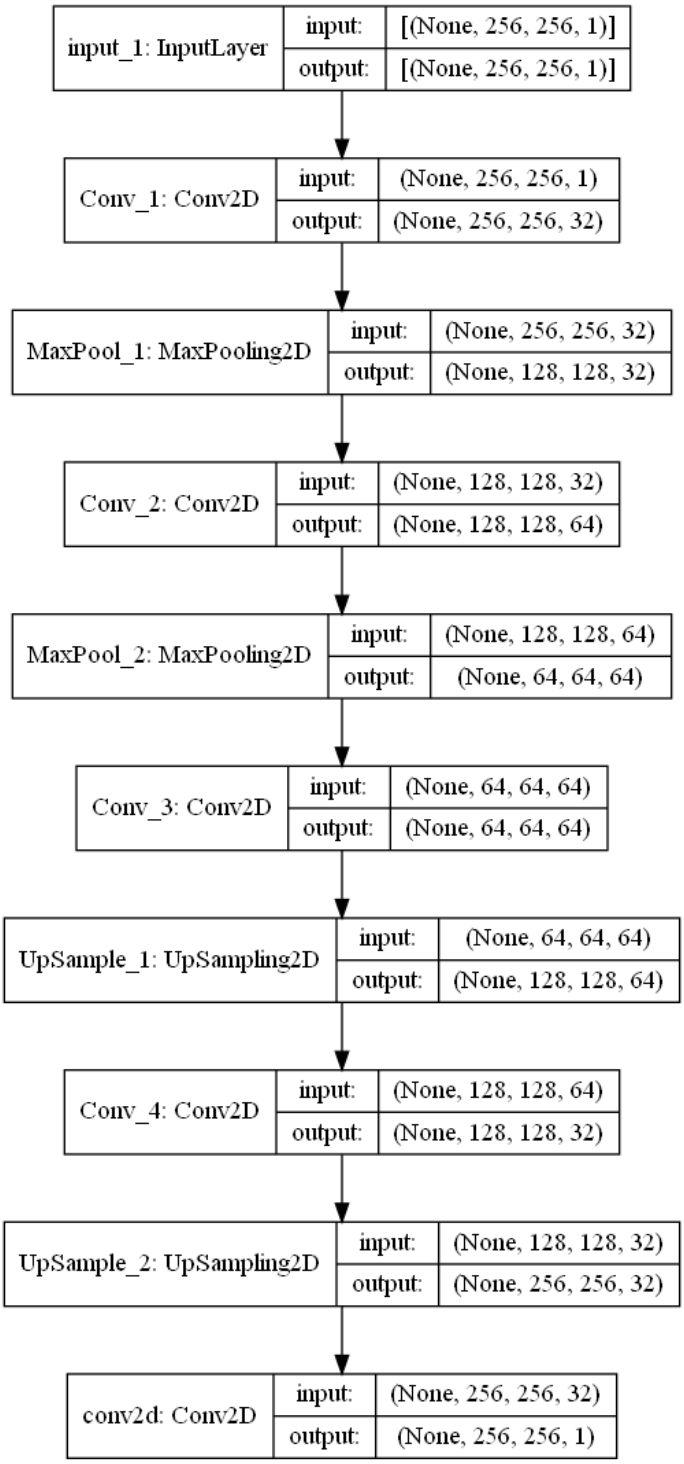


Fig. 2. CNN Autoencoder Model Architecture

For evaluating the model we've used the Mean Squared Error method to find the error. Given a noise-free $m \times n$ monochrome image I and its noisy approximation K, MSE is defined as:

$$MSE = \frac{1}{m\,n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

Equation 1. Mean squared error is used as the loss function for fitting our model.

Another metric that we've used for evaluation is accuracy.

To find the accuracy we've used the pixel accuracy formula, i.e. In this case, for a given class X:

- **True Positive (TP):** pixel classified correctly as X
- **False Positive (FP):** pixel classified incorrectly as X
- **True Negative (TN):** pixel classified correctly as not X
- **False Negative (FN):** pixel classified incorrectly as not X

Pixel accuracy is given by:

$$Pixel\ Accuracy = \frac{\#TP + \#TN}{\#TP + \#TN + \#FP + \#FN}$$

Equation 2. Accuracy formula to find accuracy of our model

3. *Input*

The input provided to our data is preprocessed by first resizing the image to the closest multiple of 256. Then, the resultant image is divided and stored into grids of 256px*256px for prediction. The stored grids are then passed through the model generated for prediction on different threads for improved performance.

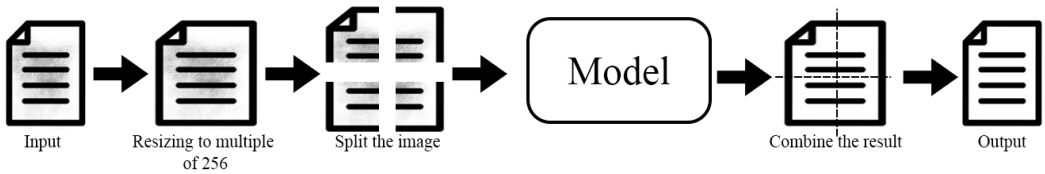
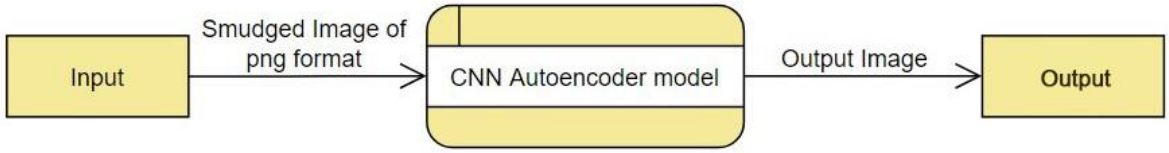


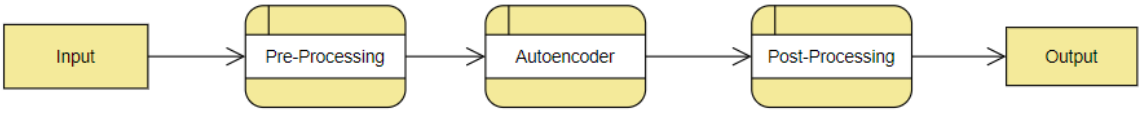
Fig. 3. Flow chart for predicting the output

IV. DATA FLOW DIAGRAMS

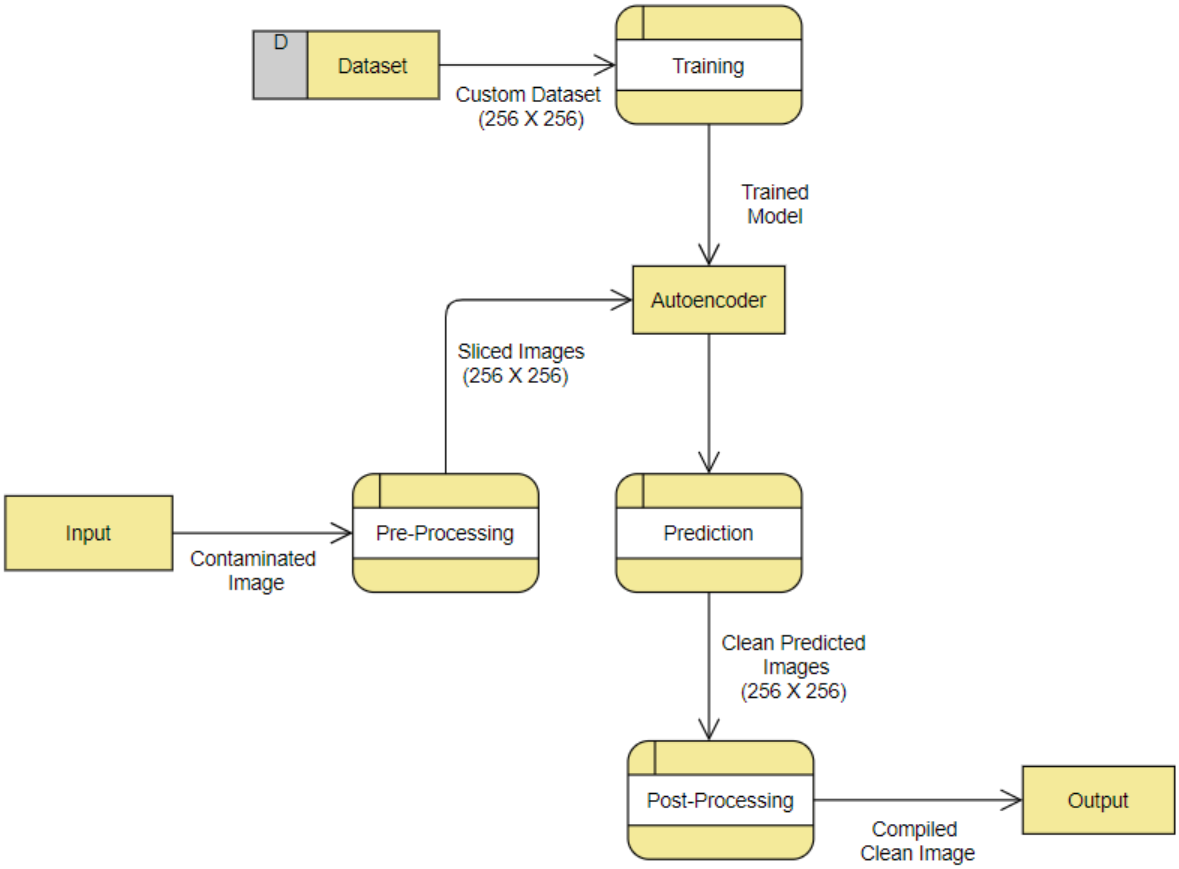
IV.1 Level 0 DFD



IV.2 Level 1 DFD



IV.3 Level 2 DFD


D
Dataset
Custom Dataset
(256 X 256)
Training
Trained
Model
Autoencoder
Sliced Images
(256 X 256)
Input
Contaminated
Image
Pre-Processing
Prediction
Clean Predicted
Images
(256 X 256)
Post-Processing
Compiled
Clean Image
Output

V. RESULTS AND OBSERVATIONS

V.I Training Phase:

a. Architecture used

For finding the best architecture that we could use for our model we ran a script consisting of different architectures in a loop such that it consists of all the filter combinations in range of 32, 64 and 128 for 5 epochs.

Then the architecture with the minimum loss value was used for developing our model. Table 1 lists the different architectures and their corresponding loss values sorted in an ascending order w.r.t loss. The loss function used here is *Mean Squared Error*.

The code and results for finding the optimal architecture is given in APPENDIX A.

MODEL	LOSS	FILTERS
model_1	0.0184	32 64 64 32
model_2	0.0194	32 128 128 32
model_8	0.0268	128 128 128 128
model_4	0.0383	64 64 64 64
model_3	0.0403	64 32 32 64
model_7	0.0414	128 64 64 128
model_5	0.0439	64 128 128 64
model	0.0445	32 32 32 32
model_6	0.0445	128 32 32 128
model_26	0.0926	64 128 128 128 128 64
model_35	0.0995	128 128 128 128 128 128
model_14	0.1013	32 64 128 128 64 32
model_34	0.1018	128 128 64 64 128 128
model_32	0.1058	128 64 128 128 64 128
model_17	0.1095	32 128 128 128 128 32
model_25	0.1112	64 128 64 64 128 64
model_23	0.1122	64 64 128 128 64 64

model_29	0.1138	128 32 128 128 32 128
model_13	0.1162	32 64 64 64 64 32
model_16	0.1175	32 128 64 64 128 32
model_33	0.1179	128 128 32 32 128 128
model_24	0.1195	64 128 32 32 128 64
model_22	0.1297	64 64 64 64 64 64
model_11	0.1354	32 32 128 128 32 32
model_31	0.1366	128 64 64 64 64 128
model_30	0.1442	128 64 32 32 64 128
model_19	0.1457	64 32 64 64 32 64
model_20	0.1482	64 32 128 128 32 64
model_21	0.1691	64 64 32 32 64 64
model_15	0.1726	32 128 32 32 128 32
model_28	0.1773	128 32 64 64 32 128
model_10	0.1817	32 32 64 64 32 32
model_27	0.1835	128 32 32 32 32 128
model_12	0.184	32 64 32 32 64 32
model_18	0.1949	64 32 32 32 32 64
model_9	0.2209	32 32 32 32 32 32

Table 1: Loss value of different models generated with different architectures.

From Table 1 we got the best architecture for our model. i.e. model_1 with the loss of 1.84% and it reduces the input image of size 256px*256px to 128px*128px latent space representation using 32px*32px filter and then to 64px*64px latent space representation using a 64px*64px filter.

b. Training the model

Epochs= 50

Batch Size= 16

Dataset size= 1044 Images of 256px*256px

Mean Squared Error is taken as the loss function.

Epochs	10	20	30	40	50
Loss	0.0025	0.0016	0.0012	0.0010	0.0008

Table 2: Loss function values for corresponding Epoch values upto 50.

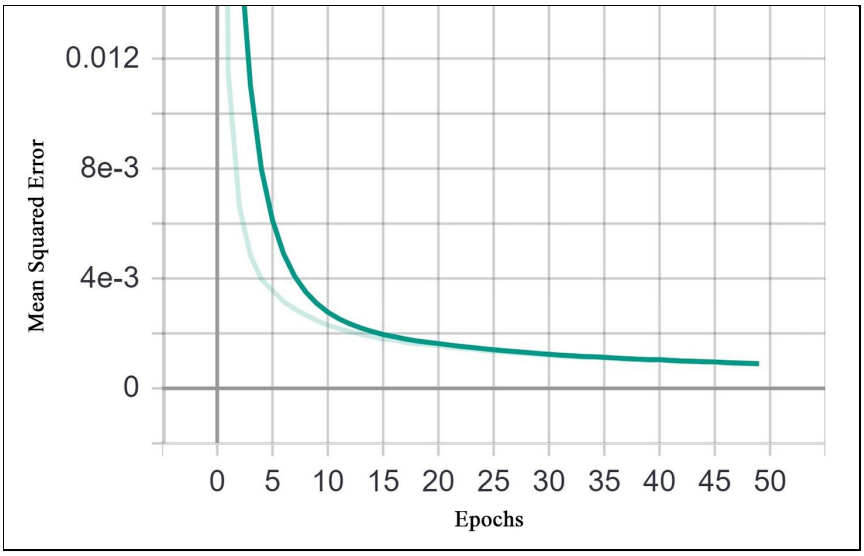


Fig 3: Mean Squared Error w.r.t. each epoch graph.

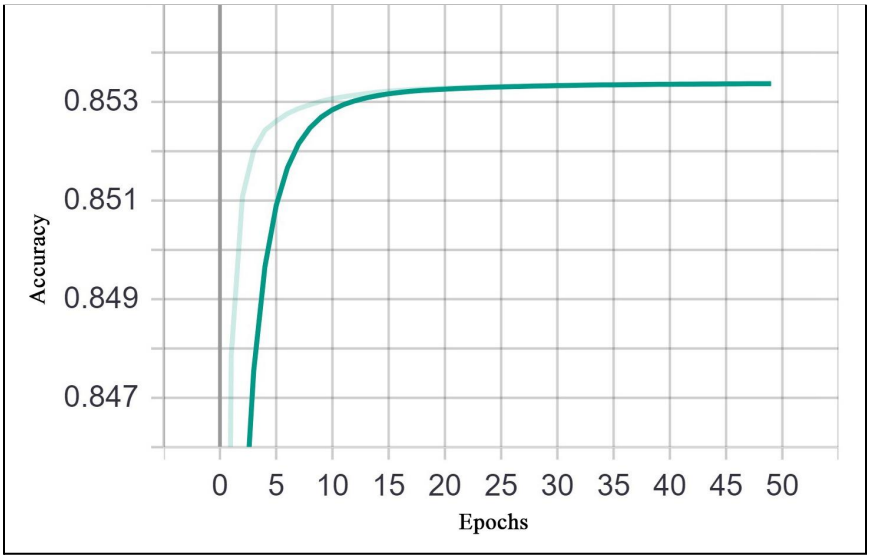


Fig 4: Accuracy w.r.t. each epoch graph.

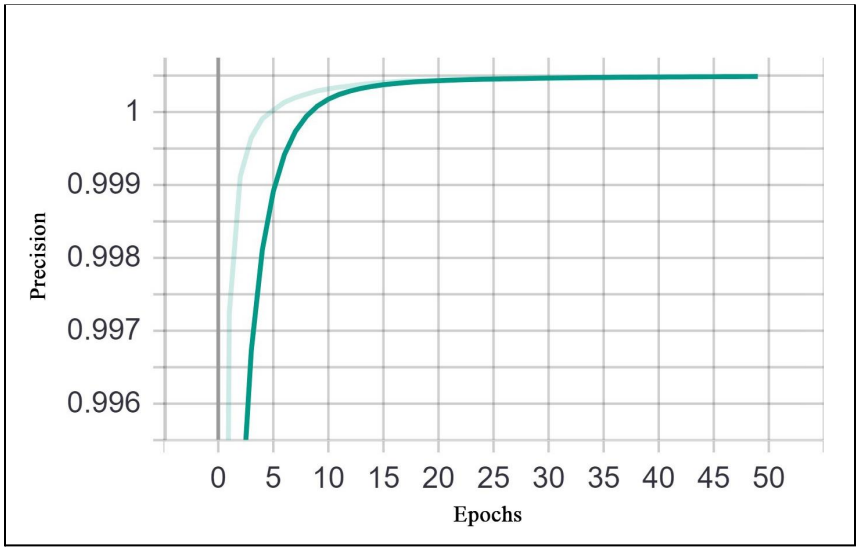


Fig 5: Precision w.r.t. each epoch graph.

The code and results for generating and training the model is given in APPENDIX B.

c. Model evaluation and metrics

Other than visual inspection, we used two different metrics to determine the performance of our model.

Mean squared error and Accuracy

For model evaluation we had a newly generated validation dataset of size approximately one-third the size of the training dataset.

To find the metrics we did some preprocessing by applying different filters and used the algorithm with the best accuracy.

The formula for finding the mean squared error is given in Equation 1 and the to find the pixel accuracy we've used the formula given in Equation 2.

Table 3 lists the result for the above evaluation.

FILTER	MEAN SQUARED ERROR	ACCURACY
None	0.0007	83.48%

Unsharpen mask	0.0090	83.43%
Gaussian Blur then Unsharpen mask	0.0051	83.33%
Laplacian mask	0.0023	83.47%
Gamma filter	0.0025	83.48%

Table 3: Metrics for our model evaluation.

Results in Table 3 shows that the two algorithms have the maximum and same accuracy. So, after a few visual inspections we decided to go with the Gamma filter model since it gives better results visually for images having text with higher pixel values.

Mean squared error and Accuracy with input as clean image

The following evaluation technique is inspired by the DeepErase[6] research paper. In this technique we pass our clean dataset for prediction to ensure that it is not removing the useful information by recognising it as a smudge. Table 4 represents the result of this evaluation.

FILTER	MEAN SQUARED ERROR	ACCURACY
None	0.0007	83.48%

Table 4: Metrics for our model evaluation with clean dataset.

The code for finding the metrics is given in APPENDIX C.

d. Sample outputs

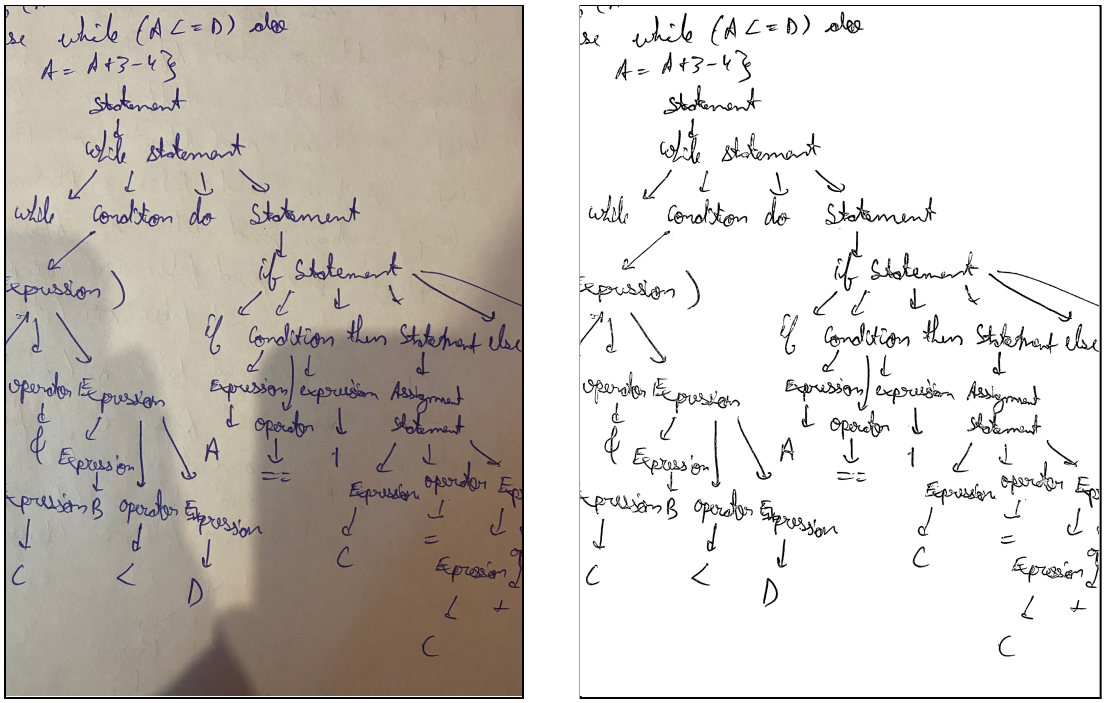


Fig 6: Handwritten text with shadows

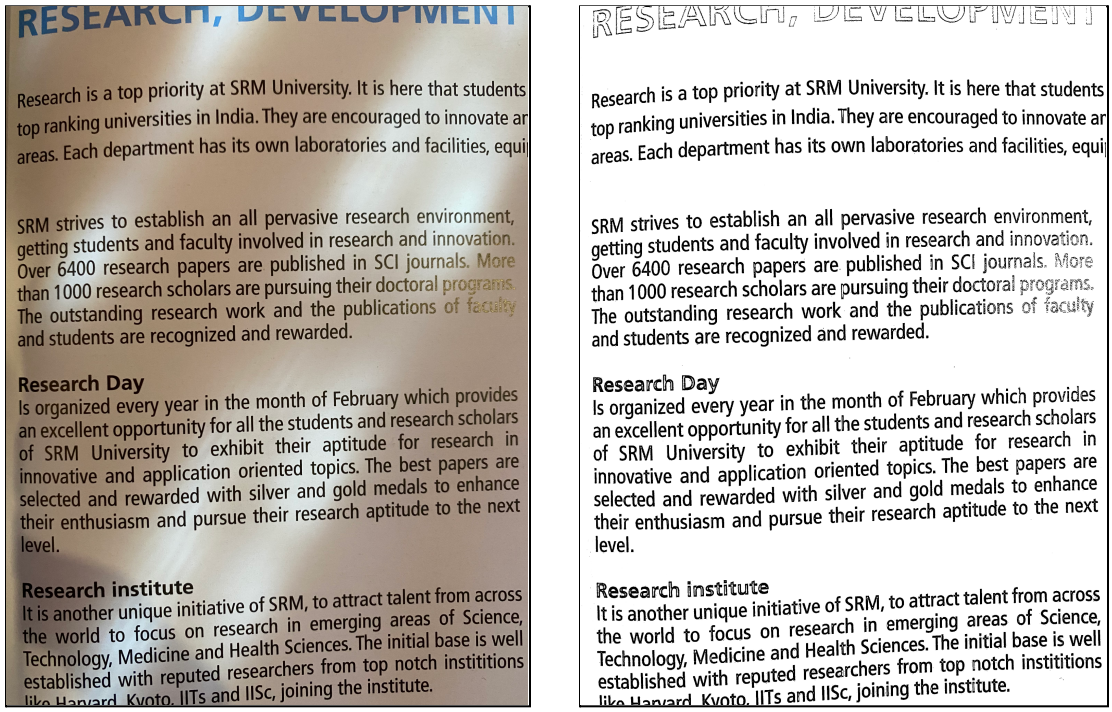


Fig 7: Printed text with shadows and reflections

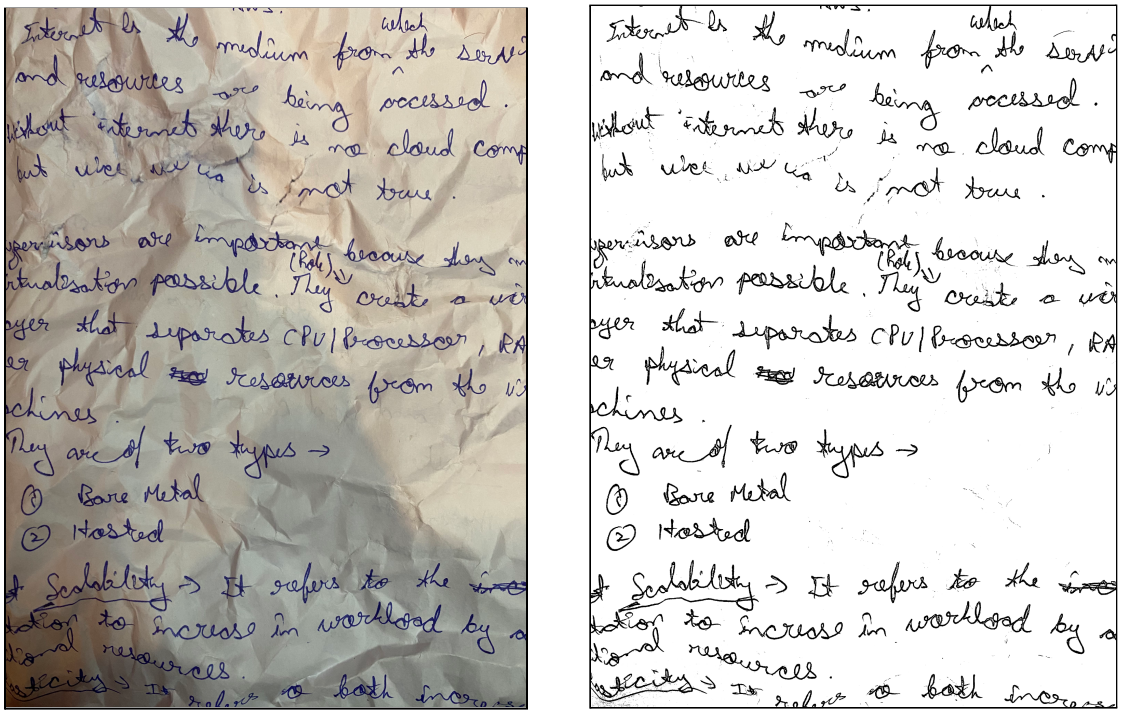


Fig 8: Handwritten text on a crushed paper

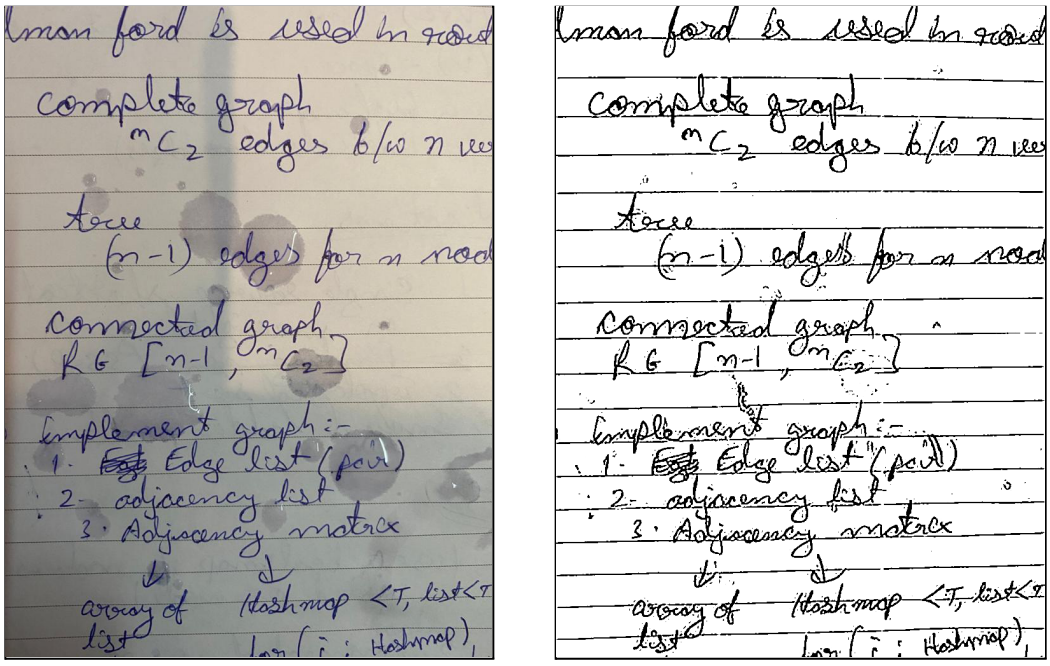


Fig 9: Handwritten text with water droplets

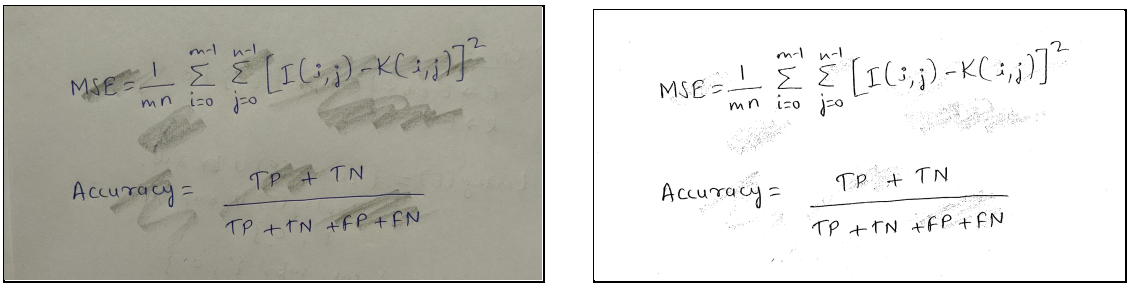


Fig 10: Mathematical formula with pencil smudges

In Fig 6-10, left hand side images are full size images captured by mobile cameras of approx. 8-12 MP resolution and the right side is their corresponding clean images.

VI. CONCLUSION AND FUTURE PLAN

In this project, we have developed a solution which removes contamination from a text document. We have trained a convolutional autoencoder with an original dataset created by ourselves. It takes a contaminated text image as input and produces a clean image as output i.e. image to image translation. Our project has a vast number of use cases like removing spilled water/oil like substances, restoring a crushed document, removing traces of background text, removing shadows, recovering a torned document etc.

We are looking forward to training our model to be able to clean not only text documents but also documents containing different shapes and diagrams. Currently, the model processes only gray scale images, it needs to be compatible with RGB images too and should provide similar results. We need to optimise the prediction time for high resolution images. A huge variety of features can be added to our dataset to improve the model accuracy and reduce loss.

REFERENCES

[1] Valueva, M.V., Nagornov, N.N., Lyakhov, P.A., Valuev, G.V. and Chervyakov, N.I., 2020. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation*, *177*, pp.232-243.

[2] Gondara, L., 2016, December. Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)* (pp. 241-246). IEEE.

[3] Hinton, G.E. and Salakhutdinov, R.R., 2006. Reducing the dimensionality of data with neural networks. *science*, *313*(5786), pp.504-507.

[4] Mohammad Imran, T. Sita Mahalakshmi, M.D. Venkata Prasad, V. Kumar Kopparty, "Denoising Dirty Document using Autoencoder," *International Journal of Computer Sciences and Engineering*, Vol.7, Issue.10, pp.21-26, 2019

[5] Prakash, M., Krull, A. and Jug, F., 2020. Divnoising: diversity denoising with fully convolutional variational autoencoders. *arXiv preprint arXiv:2006.06072*.

[6] Qi, Y., Huang, W.R., Li, Q. and Degange, J., 2020. DeepErase: Weakly Supervised Ink Artifact Removal in Document Text Images. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (pp. 3522-3530).

[7] Karkare, R., Paffenroth, R. and Mahindre, G., 2021. Blind Image Denoising and Inpainting Using Robust Hadamard Autoencoders. *arXiv preprint arXiv:2101.10876*.

[8] Liang, J. and Kelly, K., 2021. Training stacked denoising autoencoders for representation learning. *arXiv preprint arXiv:2102.08012*.

[9] Zhang, X., Wang, Y., Zhang, N., Xu, D. and Chen, B., 2019. Research on Scene Classification Method of High-Resolution Remote Sensing Images Based on RFPNet. *Applied Sciences*, *9*(10), p.2028.

APPENDIX A

Code for finding the optimal architecture for the model

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten,
Reshape, add
from tensorflow.keras.layers import Conv2D, MaxPooling2D, UpSampling2D,
Conv2DTranspose
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import TensorBoard
from skimage.util import random_noise
import cv2
import keras
import os
from PIL import Image
from time import time
from keras.utils.vis_utils import plot_model
from keras.utils.vis_utils import model_to_dot
# from wand.image import Image

PATH=r'B:\Major\finalDataset\clean'
PATH2=r'B:\Major\finalDataset\smudged'
directory=os.listdir(PATH)
directory2=os.listdir(PATH2)
directory.sort()
directory2.sort()

X_train=[]
for file in directory:
    img=cv2.imread("{}/{}".format(PATH,file),0)
    # print(img)
    try:
      # img=cv2.resize(img)
      X_train.append(img)
    except:
      pass
```

```
X_train=np.array(X_train)
X_train.shape

y_train=[]
for file in directory2:
    # print(file)
    img=cv2.imread("{}/{}".format(PATH2,file),0)
    # print(img)
    try:
      # img=cv2.resize(img)
      y_train.append(img)
    except:
      pass

y_train=np.array(y_train)
y_train.shape

X_train = X_train/255.0
y_train=y_train/255.0

shape=X_train.shape

X_train=X_train.reshape(shape[0],shape[1],shape[2],1)
y_train=y_train.reshape(shape[0],shape[1],shape[2],1)



layers=[2,3]
nodes=[32,64,128]

for l in layers:
  model=Sequential()
  input_img = keras.Input(shape=(256, 256, 1))
    #encoder
  for n1 in range(0,3):
    x = Conv2D(nodes[n1], (3, 3), activation='relu',
padding='same')(input_img)
    x = MaxPooling2D((2, 2), padding='same')(x)
    # print(nodes[n1])
    for n2 in range(0,3):
```

```
      y = Conv2D(nodes[n2], (3, 3), activation='relu', padding='same')(x)
      y = MaxPooling2D((2, 2), padding='same')(y)
      if l==3:
        for n3 in range(0,3):
          z = Conv2D(nodes[n3], (3, 3), activation='relu',
padding='same')(y)
          z = MaxPooling2D((2, 2), padding='same')(z)
          z = Conv2D(nodes[n3], (3, 3), activation='relu',
padding='same')(z)
          z = UpSampling2D((2, 2))(z)
          z = Conv2D(nodes[n2], (3, 3), activation='relu',
padding='same')(z)
          z = UpSampling2D((2, 2))(z)
          z = Conv2D(nodes[n1], (3, 3), activation='relu',
padding='same')(z)
          z = UpSampling2D((2, 2))(z)
          decoded = Conv2D(1, (3, 3),
activation='sigmoid',padding='same')(z)

          autoencoder = keras.Model(input_img, decoded)
          autoencoder.compile(optimizer='adamax',
loss='binary_crossentropy')
          autoencoder.summary()
          autoencoder.fit(y_train[350:500],
X_train[350:500],epochs=5,batch_size=16,shuffle=True)
      else:
        z = Conv2D(nodes[n2], (3, 3), activation='relu',
padding='same')(y)
        z = UpSampling2D((2, 2))(z)
        a = Conv2D(nodes[n1], (3, 3), activation='relu',
padding='same')(z)
        a = UpSampling2D((2, 2))(a)
        decoded = Conv2D(1, (3, 3),
activation='sigmoid',padding='same')(a)

        autoencoder = keras.Model(input_img, decoded)
        autoencoder.compile(optimizer='adamax', loss='mse')
        autoencoder.summary()
        autoencoder.fit(y_train[350:500],
X_train[350:500],epochs=5,batch_size=16,shuffle=True)
```

```
print(".")
```

APPENDIX B

Code for generating and training the model

```
tf.__version__

PATH=r'B:\Major\finalDataset\clean'
PATH2=r'B:\Major\finalDataset\smudged'
directory=os.listdir(PATH)
directory2=os.listdir(PATH2)
directory.sort()
directory2.sort()

directory=os.listdir(PATH)
directory2=os.listdir(PATH2)

directory.sort()
directory2.sort()

X_train=[]
for file in directory:
    img=cv2.imread("{}/{}".format(PATH,file),0)
    # print(img)
    try:
      # img=cv2.resize(img)
      X_train.append(img)
    except:
      pass

X_train=np.array(X_train)
X_train.shape

y_train=[]
for file in directory2:
    # print(file)
    img=cv2.imread("{}/{}".format(PATH2,file),0)
    # print(img)
    try:
      # img=cv2.resize(img)
      y_train.append(img)
```

```
    except:
      pass

y_train=np.array(y_train)
y_train.shape

plt.imshow(X_train[56],cmap='gray')

plt.imshow(y_train[56],cmap='gray')

X_train = X_train/255.0
y_train=y_train/255.0

shape=X_train.shape
X_train.shape

X_train=X_train.reshape(shape[0],shape[1],shape[2],1)
# noise=noise.reshape(shape[0],shape[1],shape[2],1)
y_train=y_train.reshape(shape[0],shape[1],shape[2],1)

X_train.shape

y_train.shape

model=Sequential()

tensorboard = TensorBoard(log_dir='B:\Major\git
code\ImageProcessing\logs\{}'.format(time()))

input_img = keras.Input(shape=(256, 256, 1))
    #encoder
x = Conv2D(32, (3, 3), activation='relu', padding='same', name =
'Conv_1')(input_img)
x = MaxPooling2D((2, 2), padding='same', name = 'MaxPool_1')(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same', name =
'Conv_2')(x)
encoded = MaxPooling2D((2, 2), padding='same', name = 'MaxPool_2')(x)
#decoder
x = Conv2D(64, (3, 3), activation='relu', padding='same', name =
'Conv_3')(encoded)
```

```
x = UpSampling2D((2, 2), name = 'UpSample_1')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same', name =
'Conv_4')(x)
x = UpSampling2D((2, 2), name = 'UpSample_2')(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adamax', loss='binary_crossentropy',
metrics=['accuracy', 'mse', 'mae','Precision'])
autoencoder.summary()

plot_model(autoencoder, to_file=r'B:\Major\git
code\ImageProcessing\Models\model_plot.png', show_shapes=True,
show_layer_names=True)

autoencoder.fit(y_train, X_train,
                epochs=50,
                batch_size=16,
                shuffle=True,
                callbacks=[tensorboard])

json_model = autoencoder.to_json()
json_file = open(r'B:\Major\git
code\ImageProcessing\Models\SmudgeRemovalV6\smudge_autoencoderV7.json',
'w')
json_file.write(json_model)
# saving model weights
autoencoder.save_weights(r'B:\Major\git
code\ImageProcessing\Models\SmudgeRemovalV6\smudge_autoencoder_weightsV7.h
5', 'w')
```

APPENDIX C

Code for evaluating metrics

```
PATH=r'B:\Major\validationDataset\validationDataset\clean'
PATH2=r'B:\Major\validationDataset\validationDataset\smudged'
directory=os.listdir(PATH)
directory2=os.listdir(PATH2)
directory.sort()
directory2.sort()

X_train=[]
for file in directory:
    img=cv2.imread("{}/{}".format(PATH,file),0)
    try:
      X_train.append(img)
    except:
      pass

y_train=[]
for file in directory:
    img=cv2.imread("{}/{}".format(PATH2,file),0)
    try:
      y_train.append(img)
    except:
      pass

plt.imshow(X_train[0])

plt.imshow(y_train[0])

def apply_unsharpen(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    image = Image.fromarray(image.astype('uint8'))
    new_image = image.filter(ImageFilter.UnsharpMask(radius=1,
percent=150))
    new_image = np.array(new_image)
    return new_image

def apply_gaussian_unsharpen(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

```
    figure_size = 9
    image = cv2.GaussianBlur(image, (figure_size, figure_size),0)
    image2 = cv2.cvtColor(image, cv2.COLOR_HSV2BGR)
#     image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)
    image2 = Image.fromarray(image2.astype('uint8'))
    new_image = image2.filter(ImageFilter.UnsharpMask(radius=2,
percent=150))
    return np.array(new_image)

def apply_laplacian(image):
    kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
    image = cv2.filter2D(image, -1, kernel)
    return image

def apply_gamma(image):
    k=1.8
    image = 255 * (image/255)**k
    return image

# for i in range(0, len(y_train)):
#     y_train[i] = apply_unsharpen(y_train[i])
#     y_train[i] = apply_gaussian_unsharpen(y_train[i])
#     y_train[i] = apply_laplacian(y_train[i])
#     y_train[i] = apply_gamma(y_train[i])
#     y_train[i] = cv2.cvtColor(y_train[i], cv2.COLOR_BGR2GRAY)
plt.imshow(y_train[0])

X_train=np.array(X_train)
y_train=np.array(y_train)
y_train.shape

X_train = X_train/255.0
y_train = y_train/255.0

shape=X_train.shape
X_train=X_train.reshape(shape[0],shape[1],shape[2],1)
y_train=y_train.reshape(shape[0],shape[1],shape[2],1)

PATH3=r'B:\Major\model'
model = 'smudge_autoencoderV7.json'
```

```
weight = 'smudge_autoencoder_weightsV7.h5'

json_file = open(r'{}\{}'.format(PATH3, model), 'r')
json_model = model_from_json(json_file.read())
json_model.load_weights(r'{}\{}'.format(PATH3, weight))

json_model.compile(optimizer='adamax', loss='binary_crossentropy',
metrics=['accuracy', 'mse', 'mae','Precision'])

result = json_model.evaluate(y_train, X_train, batch_size=1)

test_image =
json_model.predict(y_train[0].reshape(1,256,256,1)).reshape(256, 256)

cv2.imwrite(r'B:\Major\datast\filter_test\vt1.png'.format(PATH),
test_image*255)
plt.imshow(test_image)
```

APPENDIX D

Code for generating a clean image

```
PATH=r'B:\Major\model'
os.listdir(PATH)

PATH2=r'B:\Major\datast'
os.listdir(PATH2)

PATH3=r'B:\Major\datast\output'

model = 'smudge_autoencoderV2.json'
weight = 'smudge_autoencoder_weightsV2.h5'

slice_size = 256
input_file_name = 'test3.png'
print("{}\{}".format(PATH2, input_file_name))

img = cv2.imread("{}\{}".format(PATH2, input_file_name), 0)
plt.imshow(img)
v_res = img.shape[0]
h_res = img.shape[1]
img.shape

def apply_gamma(image):
    k=1.8
    image = 255 * (image/255)**k
    return image

img = apply_gamma(img)
img

def count_blocks(res, slice_size):
  blocks = 0
  if res % slice_size == 0:
    blocks = int(res/slice_size)
  else:
    blocks = (int(res/slice_size)) + 1
  return blocks
```

```
def resizeImage(img, v_res, h_res, slice_size):
  v_res = count_blocks(v_res, slice_size) * slice_size
  h_res = count_blocks(h_res, slice_size) * slice_size
  img = cv2.resize(img, (h_res, v_res))
  print(v_res)
  print(h_res)
  return img

img = resizeImage(img, v_res, h_res, slice_size)
plt.imshow(img)
img.shape

json_file = open(r'{}\{}'.format(PATH, model), 'r')
json_model = model_from_json(json_file.read())
json_model.load_weights(r'{}\{}'.format(PATH, weight))

# json_file.seek(0)
# json_file.read()

column = count_blocks(h_res, slice_size)
rows = count_blocks(v_res, slice_size)
print(column)
print(rows)

normalImg = img
newImg = None

hor_split_image = np.hsplit(normalImg, column)
for i in hor_split_image:
    v_split = np.vsplit(i, rows)
    cleanHor = None
    for j in v_split:
        smallClean =
json_model.predict(j.reshape(1,256,256,1)).reshape(256, 256)
        if cleanHor is None:
            cleanHor = smallClean
        else:
            cleanHor = np.concatenate((cleanHor, smallClean), axis=0)
    if newImg is None:
```

```
        newImg = cleanHor
    else:
        newImg = np.concatenate((newImg, cleanHor), axis=1)

newImg = newImg.reshape(img.shape[0], img.shape[1])
newImg = cv2.resize(newImg, (h_res, v_res))
plt.imshow(newImg, cmap='gray')

cv2.imwrite(r'{}\output_{}'.format(PATH3, input_file_name), newImg*255)
```