MINISTRY OF EDUCATION AND RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

SOFTWARE ENGINEERING

ARTISTIC STYLE TRANSFER WEBSITE SOLUTION DESCRIPTION DOCUMENT

Teaching assistant: Eneia Nicolae Traian Todoran
Team members: Badea Cornel-Alexandru, Horvath Andrea-Anett, Mihai Cristina-Mădălina

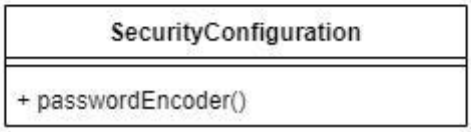
Faculty of Automation and Computer Science
Computer Science Department (En)
3rd year of study, gr. 30434

Abstract

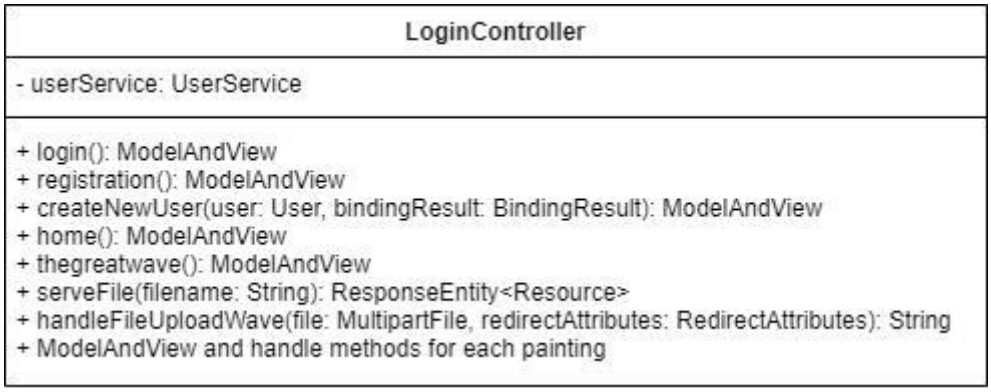
Recent advancements in image processing field has produced results in almost every field, including art, which has been viewed as a pantheon of humanity. Papers like “A Neural Algorithm of Artistic Style[1]” presents techniques for artistic style transfer from a painting to an image using Convolutional Neural Networks. In this project we propose to build a fully deployable website around those techniques using Spring MVC framework. It’s main functionality is allowing the users to upload their images and choose what available painting style they want to transfer to it.

UML diagram

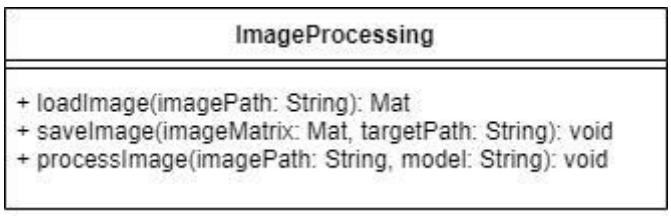
configuration package diagram:



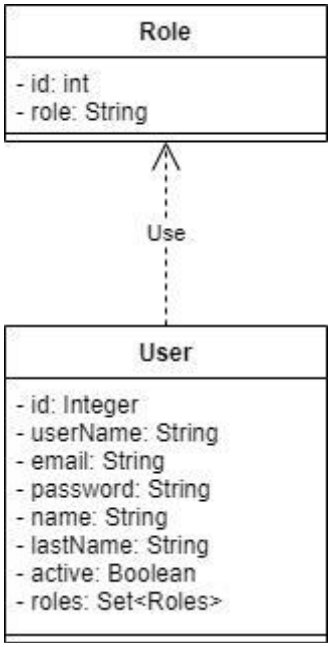
controller package diagram:



imgprocess package diagram:

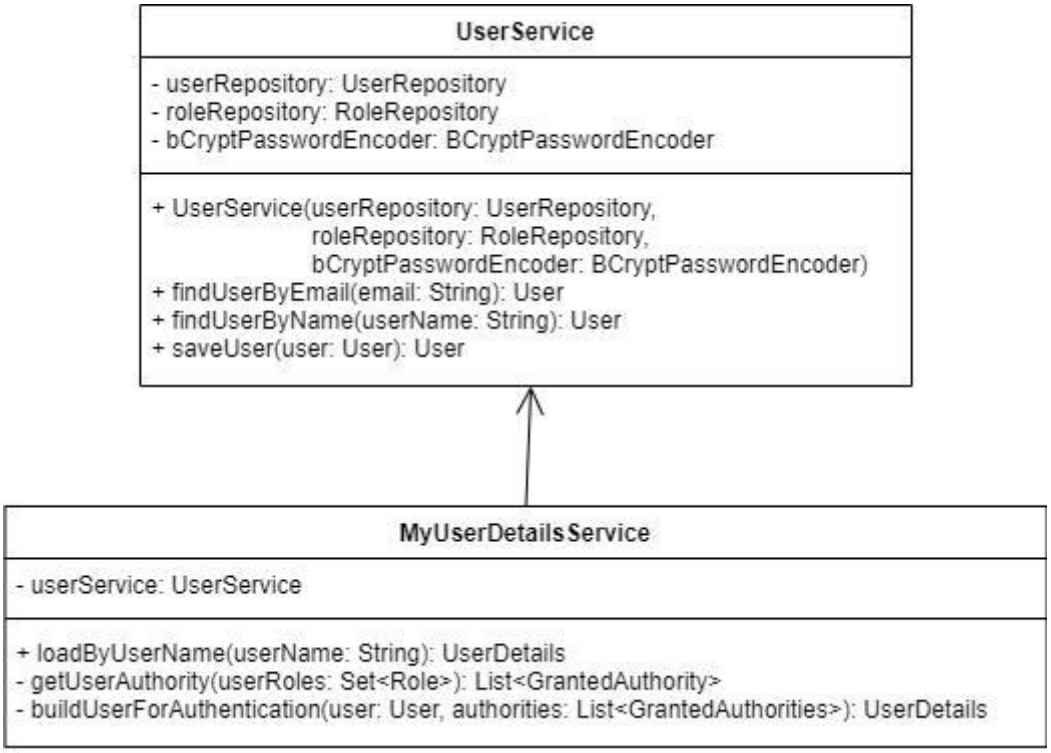


model package diagram:

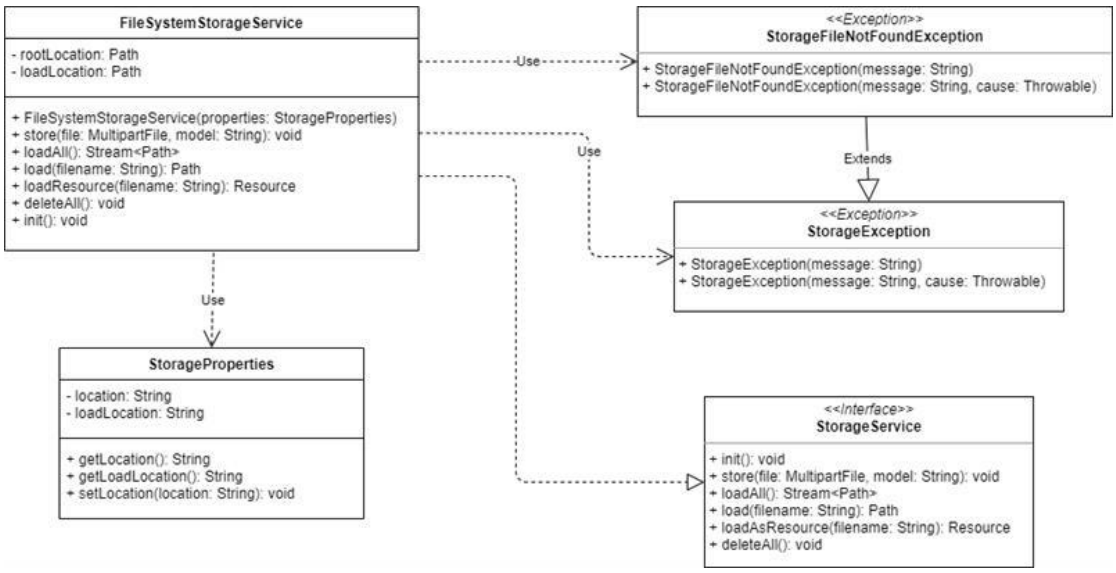


repository package diagram:

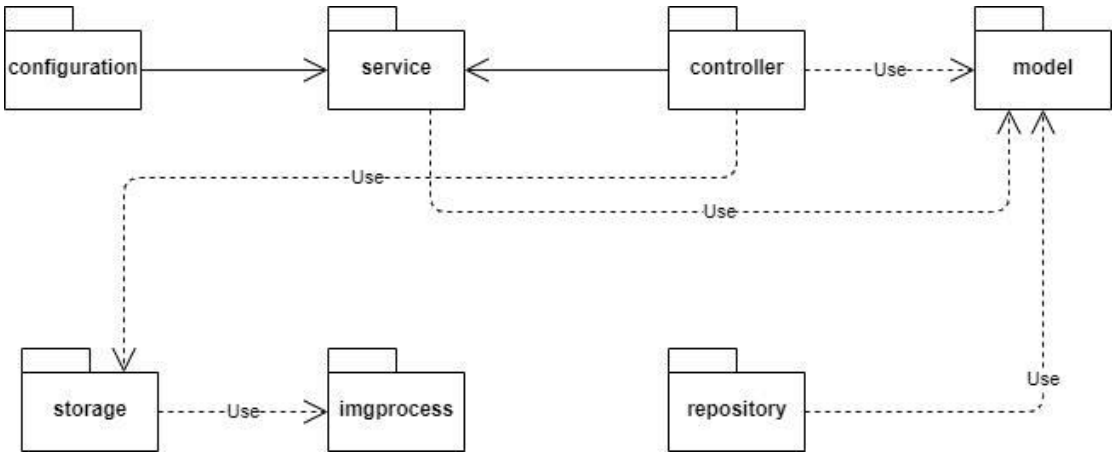
service package diagram:



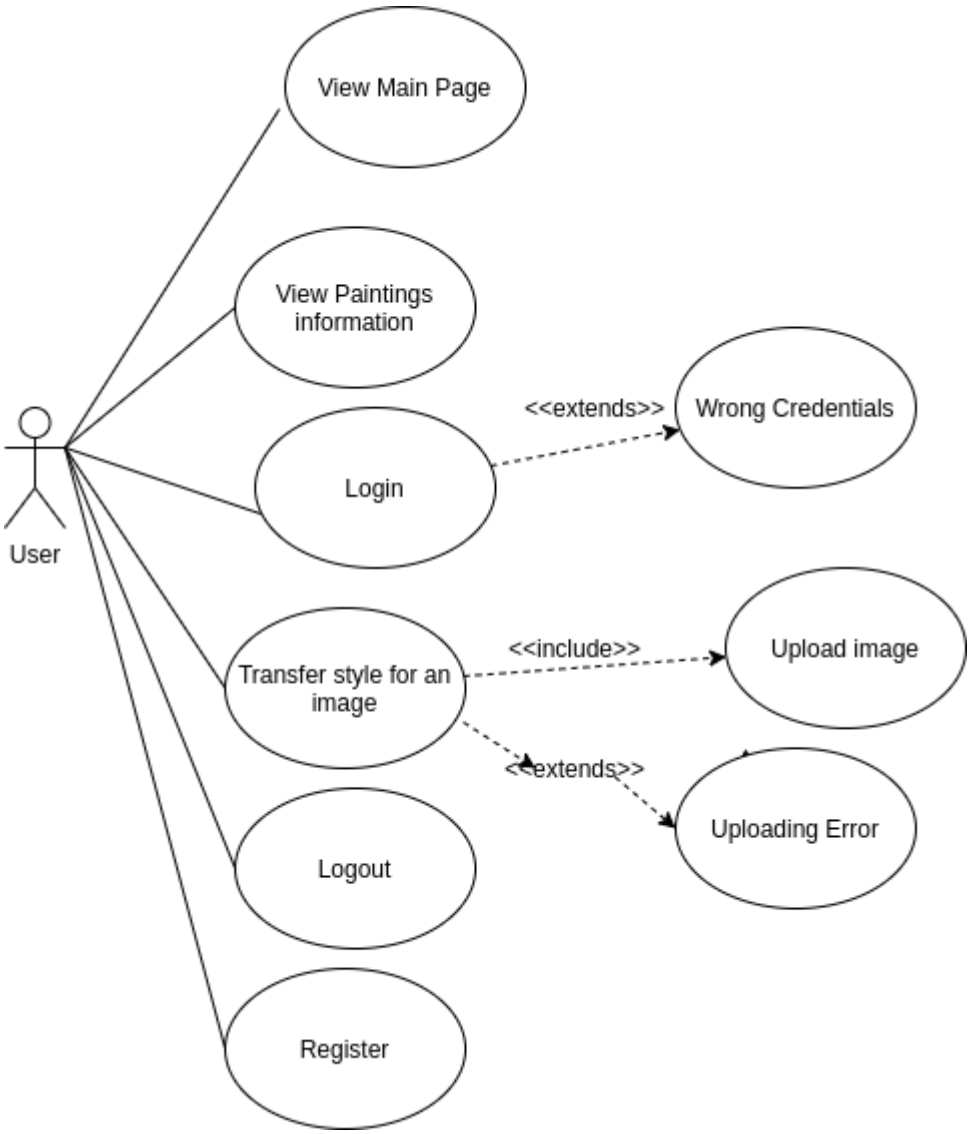
storage package diagram:



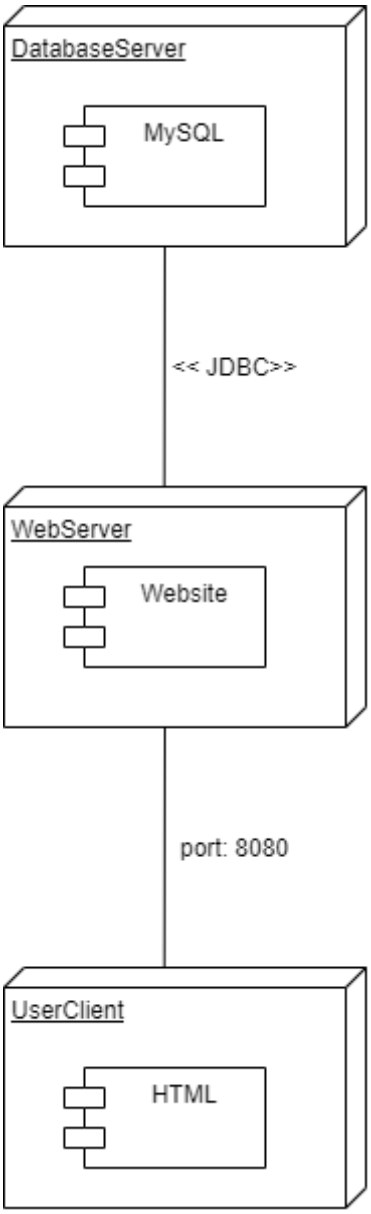
Package diagram:



Use case diagram



Deployment diagram



Source code

com.gpch.login package

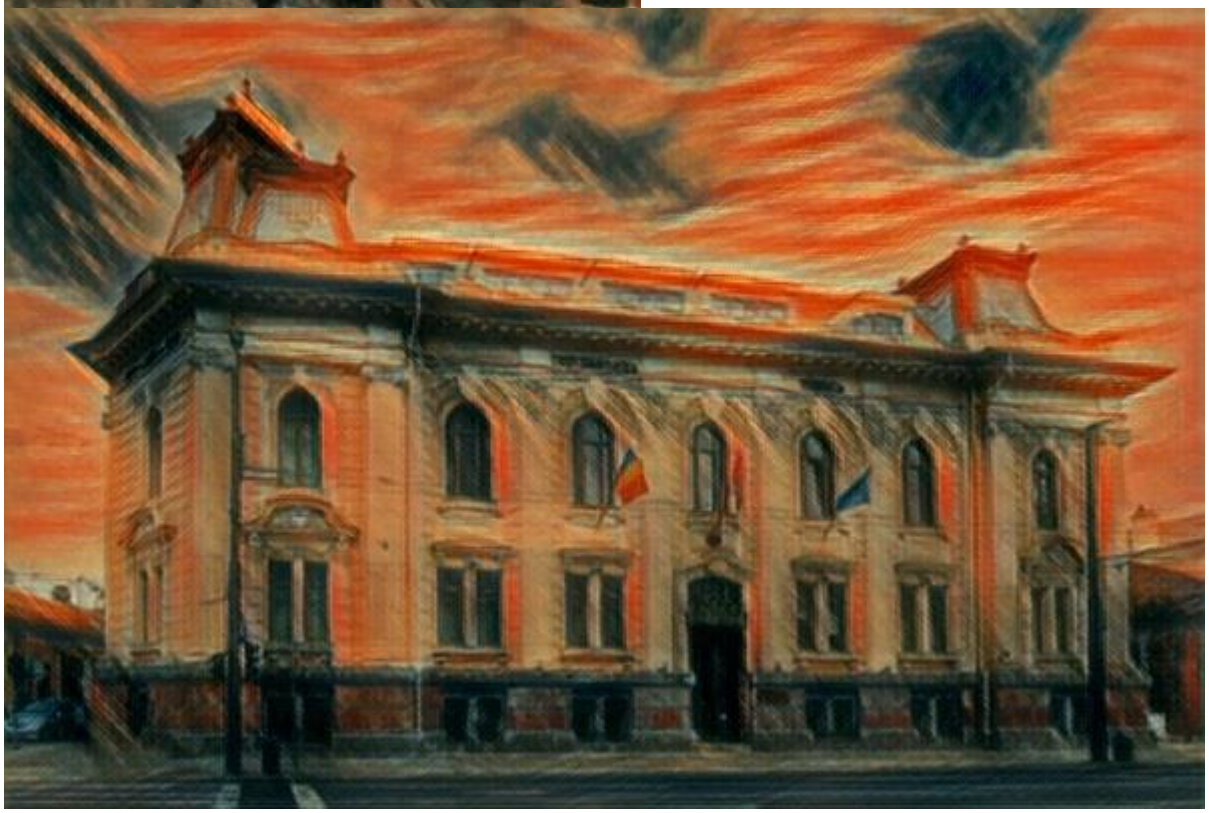
Experimental results

Here is an example of different styles applied on the same image.

The wave painting style:

Starry night painting style:

The Scream painting style:



Alpha/Beta ratio

Into the loss function

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

α and β are the weighting factors for content and style reconstruction respectively. To experiment with the results with different values for those variables in the early stages of the training process we begin from the content image and train our model for 50 iterations (out for 1000 minimum required).


original
(β = 0)


α = 1
$\beta = 10^3$


α = 1
$\beta = 10^7$

As expected even in the early stages of training, the style “noise” is more pervasive as the beta factor increases.

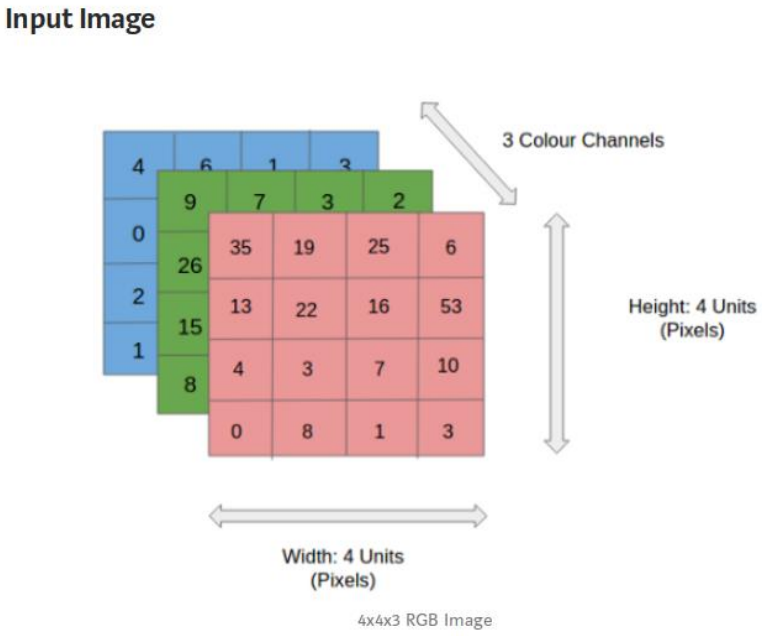
Bibliographic study

What is a Convolutional Neural Network[2]

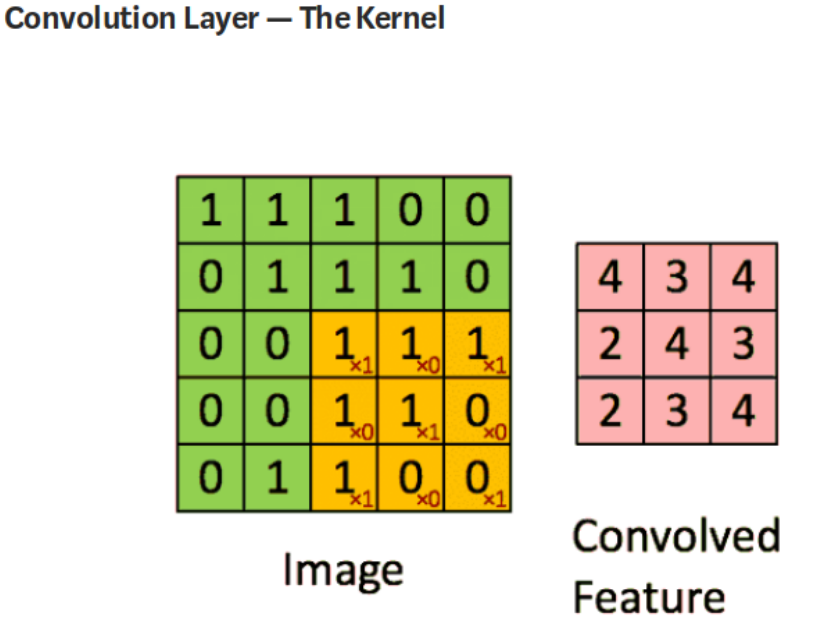
In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.



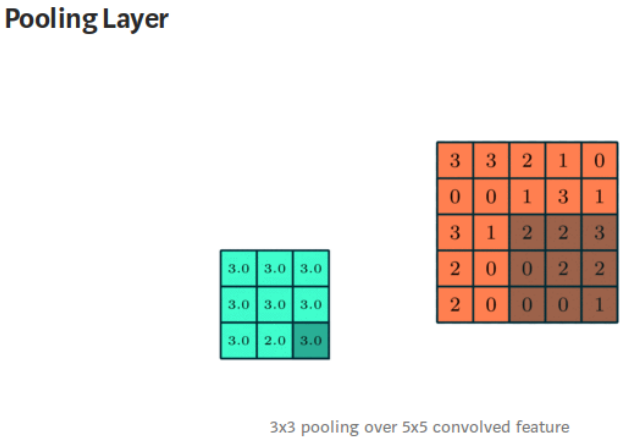
We will apply two steps to the input image: convolution and pooling.[3]



Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB)

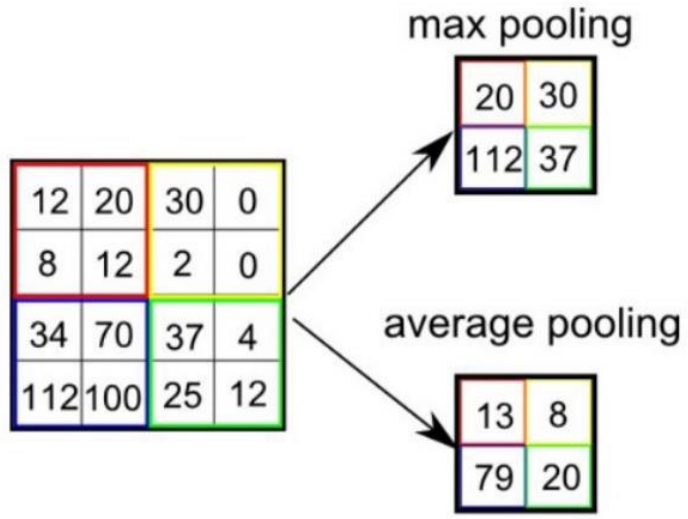
In the above demonstration, the green section resembles our 5x5x1 input image, I. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in the color yellow. We have selected K as a 3x3x1 matrix.



3x3 pooling over 5x5 convolved feature

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.



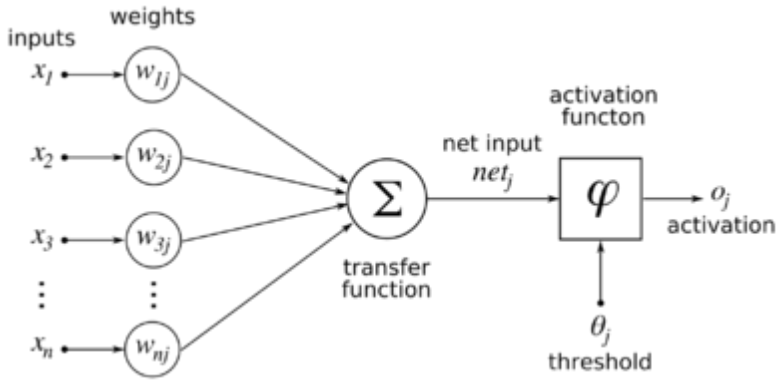
Types of Pooling

After going through the above process, we have successfully enabled the model to understand the features. Moving on, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes.

Backpropagation[4]

Backpropagation is a method to adjust the connection weights to compensate for each error found during learning. The error amount is effectively divided among the connections. Technically, backprop calculates the gradient (the derivative) of the cost function associated with a given state with respect to the weights. The weight updates can be done via stochastic gradient descent or other methods.

Finding the derivative of the error:



Activation function is simply a function that tells the newron when to output data.

For the simple perceptron example the activation function was the output to be greater than 1.

Diagram of an artificial neural network to illustrate the notation used here.

Calculating the partial derivative of the error with respect to a weight wij i*s done using the chain rule* twice:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial \mathrm{net}_j}\frac{\partial \mathrm{net}_j}{\partial w_{ij}}$$

The error depends on the output of the network, that in turn depends on the output of a neuron that is also dependent on the assigned weights.

Style Transfer

All the information in this section is taken from the paper "A Neural Algorithm of Artistic Style"[5]

When Convolutional Neural Networks are trained on object recognition, they develop a representation of the image that makes object information increasingly explicit along the processing hierarchy. Therefore, along the processing hierarchy of the network, the input image is transformed into representations that increasingly care about the actual content of the image compared to its detailed pixel values.
To obtain a representation of the style of an input image, we use a feature space originally designed to capture texture information.

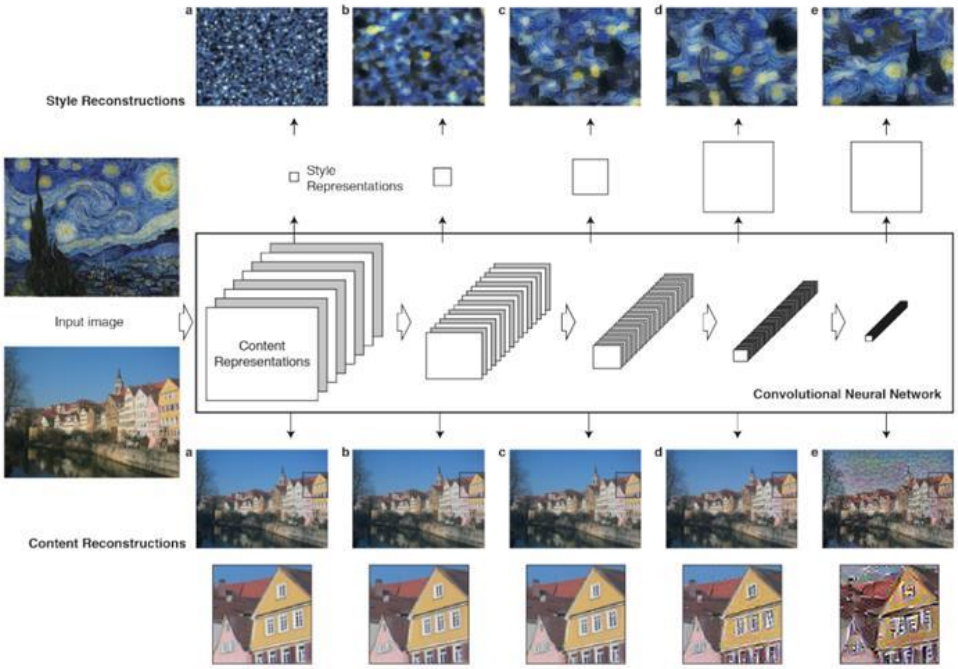


Figure 1[5]:Convolutional Neural Network (CNN). A given input image is represented as a set of filtered images at each processing stage in the CNN. Content Reconstructions. We can visualise the information at different processing stages in the CNN by reconstructing the input image from only knowing the network's responses in a particular layer. We reconstruct the input image from from layers 'conv11' (a), 'conv21' (b), 'conv31' (c), 'conv41' (d) and 'conv51' (e) of the original VGG-Network. We find that reconstruction from lower layers is almost perfect (a,b,c). In higher layers of the network, detailed pixel information is lost while the high-level content of the image is preserved (d,e).Style Reconstructions. On top of the original CNN representations we built a new feature space that captures the style of an input image. The style representation computes correlations between the different features in different layers of the CNN.

A layer with Nl distinct filters has Nl feature maps each of size Ml, where Ml is the height times the width of the feature map. So the responses in a layer l can be stored in a matrix Fl∈ RNl×Ml where Flij is the activation of the ith filter at position j in layer l.

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^l - P_{ij}^l\right)^2$$

The derivative of this loss with respect to the activations in layer l equals

$$\frac{\partial \mathcal{L}_{content}}{\partial F_{ij}^l} = \begin{cases} \left(F^l - P^l\right)_{ij} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0 \,. \end{cases}$$

On top of the CNN responses in each layer of the network we built a style representation that computes the correlations between the different filter responses, where the expectation is taken over the spatial extend of the input image. These feature correlations are given by the Gram matrix Gl∈ RNl×Nl, where Glij is the inner product between the vectorised feature map i and j in layer l:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l.$$

The contribution of that layer to th etotal loss is then

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2$$

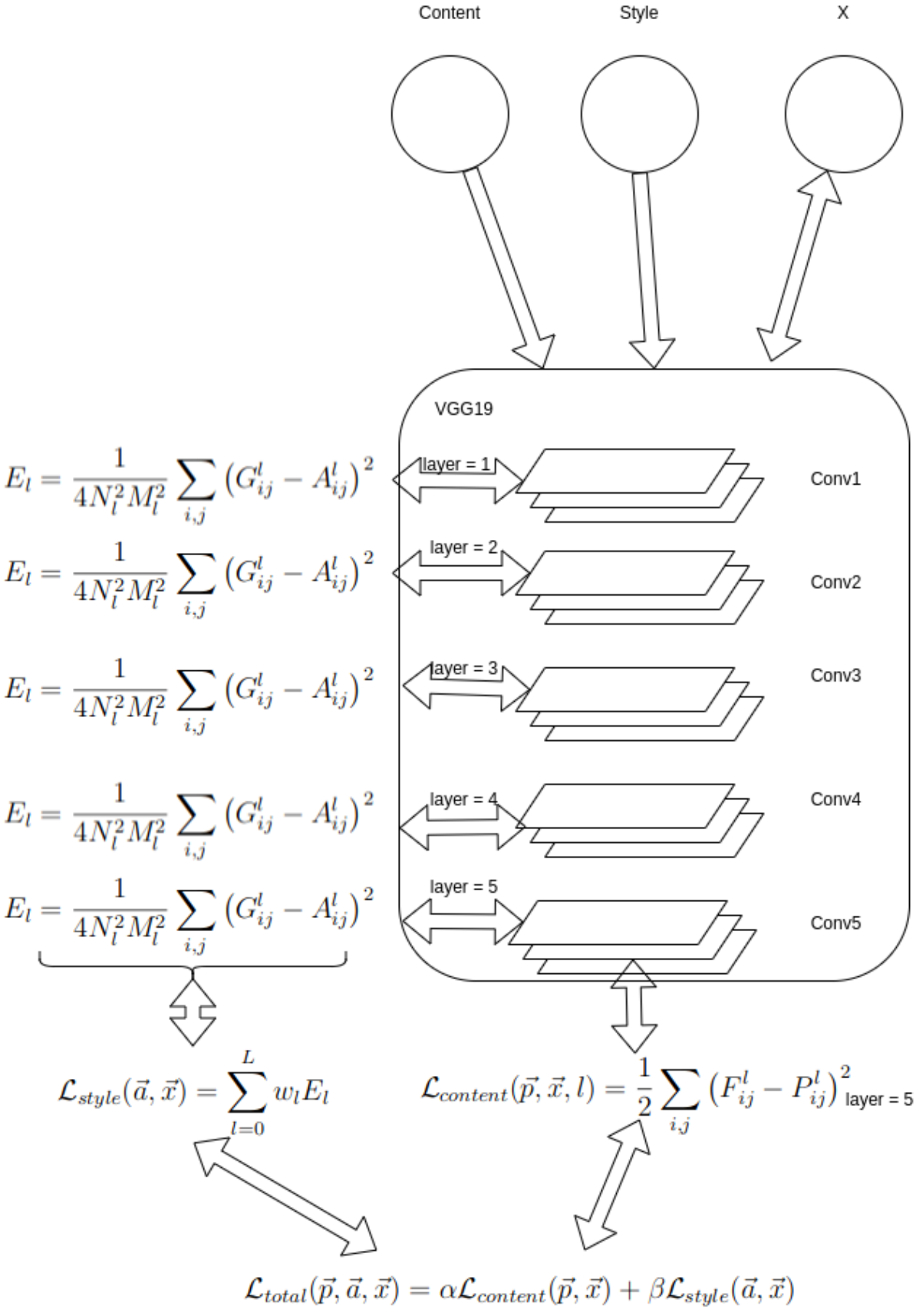
and the total loss is

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

The loss function we minimise is

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

Architecture VGG19 for style transfer


Content
Style
X
VGG19
layer = 1
Conv1
layer = 2
Conv2
layer = 3
Conv3
layer = 4
Conv4
layer = 5
Conv5
$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2$
$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$
$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^l - P_{ij}^l\right)^2$ layer = 5
$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$

Code

```
import collections
import tensorflow as tf
import numpy as np
import utils
import vgg19
import argparse


parser = argparse.ArgumentParser()


parser.add_argument('--content', type=str, default='images/mona.jpg')
parser.add_argument('--style', type=str, default='images/starry-night.jpg')
parser.add_argument('--output', type=str, default='output.jpg')
parser.add_argument('--image_size', type=int, default=512)
parser.add_argument('--num_iter', type=int, default=1000)


def add_batch_dim(image):
shape = (1,) + image.shape
return np.reshape(image, shape)


def main():
args = parser.parse_args()

model_file_path = "pre_trained_model/" + vgg19.MODEL_FILE_NAME
vgg_net = vgg19.VGG19(model_file_path)


content_image = utils.load_image(args.content, max_size=args.max_size)
style_image = utils.load_image(args.style,
shape=(content_image.shape[1],content_image.shape[0]))

# initial image to be transformed
#init_image = np.random.normal(size=content_image.shape,
scale=np.std(content_image))
init_image = content_image

init_image = add_batch_dim(init_image)
```

```
content_image = add_batch_dim(content_image)
style_image = add_batch_dim(style_image)

#relevant layers and their weighting
CONTENT_LAYERS =   collections.OrderedDict({'conv4_2' : 1.0})
STYLE_LAYERS =collections.OrderedDict({'relu1_1':.2, 'relu2_1':.2, 'relu3_1':.2,
'relu4_1':.2, 'relu5_1':.2})


# open session
sess = tf.Session(config=tf.ConfigProto(allow_soft_placement=True))

num_iter = args.num_iter

p0 = np.float32(vgg_net.preprocess(content_image))
a0 = np.float32(vgg_net.preprocess(style_image))
x0 = np.float32(vgg_net.preprocess(init_image))

x = tf.Variable(x0, trainable=True, dtype=tf.float32)

p = tf.placeholder(tf.float32, shape=p0.shape, name='content')
a = tf.placeholder(tf.float32, shape=a0.shape, name='style')

content_layers = vgg_net.feed_forward(p, scope='content')

Pn = {}
for id in CONTENT_LAYERS:
    Pn[id] = content_layers[id]

style_layers = vgg_net.feed_forward(a, scope='style')
An = {}
for id in STYLE_LAYERS:
    An[id] = gram_matrix(style_layers[id])

Fn = vgg_net.feed_forward(x, scope="x")

L_content = 0
L_style = 0
for id in Fn:
    if id in CONTENT_LAYERS:

        F = Fn[id]
        P = Pn[id]

        _, h, w, d = F.get_shape()

        w = CONTENT_LAYERS[id]
```

```
            L_content += w * tf.reduce_sum(tf.pow((F - P), 2)) / 2   # original paper

        elif id in STYLE_LAYERS:

            F = Fn[id]

            _, h, w, d = F.get_shape()
            N = h.value * w.value

            w = STYLE_LAYERS[id]

            G = gram_matrix(F)
            A = An[id]

            L_style += w * (1. / (4 * N ** 2 * M ** 2)) * tf.reduce_sum(tf.pow((G - A), 2))

alpha = 1e-3
beta = 1

L_content = L_content
L_style = L_style
L_total = alpha * L_content + beta * L_style

global iter
iter = 0

def callback(lt):
    global iter
    print('iteration : %d, ' % iter, 'L_total : %g' % (lt))
    iter += 1

optimizer = tf.contrib.opt.ScipyOptimizerInterface(L_total, method='L-BFGS-B',
                                                   options={'maxiter': num_iter})


init_op = tf.global_variables_initializer()
sess.run(init_op)


optimizer.minimize(sess, feed_dict={a: a0, p: p0},
                   fetches=[L_total, L_content, L_style], loss_callback=callback)


final_image = sess.run(x)
saver = tf.train.Saver()
saver.save(sess, 'my_test_model.t7')

# pixel-values between 0 and 255
```

```
final_image = np.clip(vgg_net.undo_preprocess(final_image), 0.0, 255.0)


sess.close()

# remove batch dimension
shape = final_image.shape
result_image = np.reshape(final_image,shape[1:])


utils.save_image(result_image,args.output)


def gram_matrix(tensor):

shape = tensor.get_shape()

num_channels = int(shape[3])

matrix = tf.reshape(tensor, shape=[-1, num_channels])

gram = tf.matmul(tf.transpose(matrix), matrix)

return gram


if __name__ == '__main__':
main()
```

References:

[1] https://answers.opencv.org/question/205824/dnn-forward-result-has-rows-1-colums-1/
[2]https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/
[3]https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
[4]https://en.wikipedia.org/wiki/Backpropagation
[5]https://arxiv.org/pdf/1508.06576.pdf