

Group Name: **Gallivanters**

Team: **Chitranjan Gupta** (Team Leader)

**Umesh Tikhe** (Coder)

**Lavanya Paspuleti** (Coder)

**Riya Vachhani** (Planning & Co-ordination)

**Amit Purswani** (R.&D. & Presentation)

Topic: **Neural Style Transfer**

Underlying Concept: Convolutional Neural Networks

Problem Statement:

Develop a codebase to perform Neural Style Transfer given a content image and a style image

Challenges: 1. Neural Style Transfer

2. Web Application

3. Deployment on a Platform

The Concepts used:

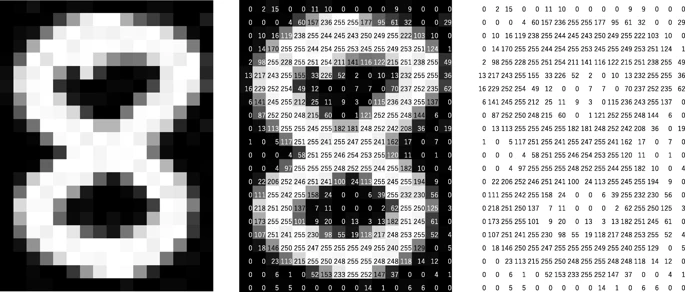
**Convolutional Neural Networks** –

CNN is a type of deep learning model for processing data that has a grid pattern, such as images.

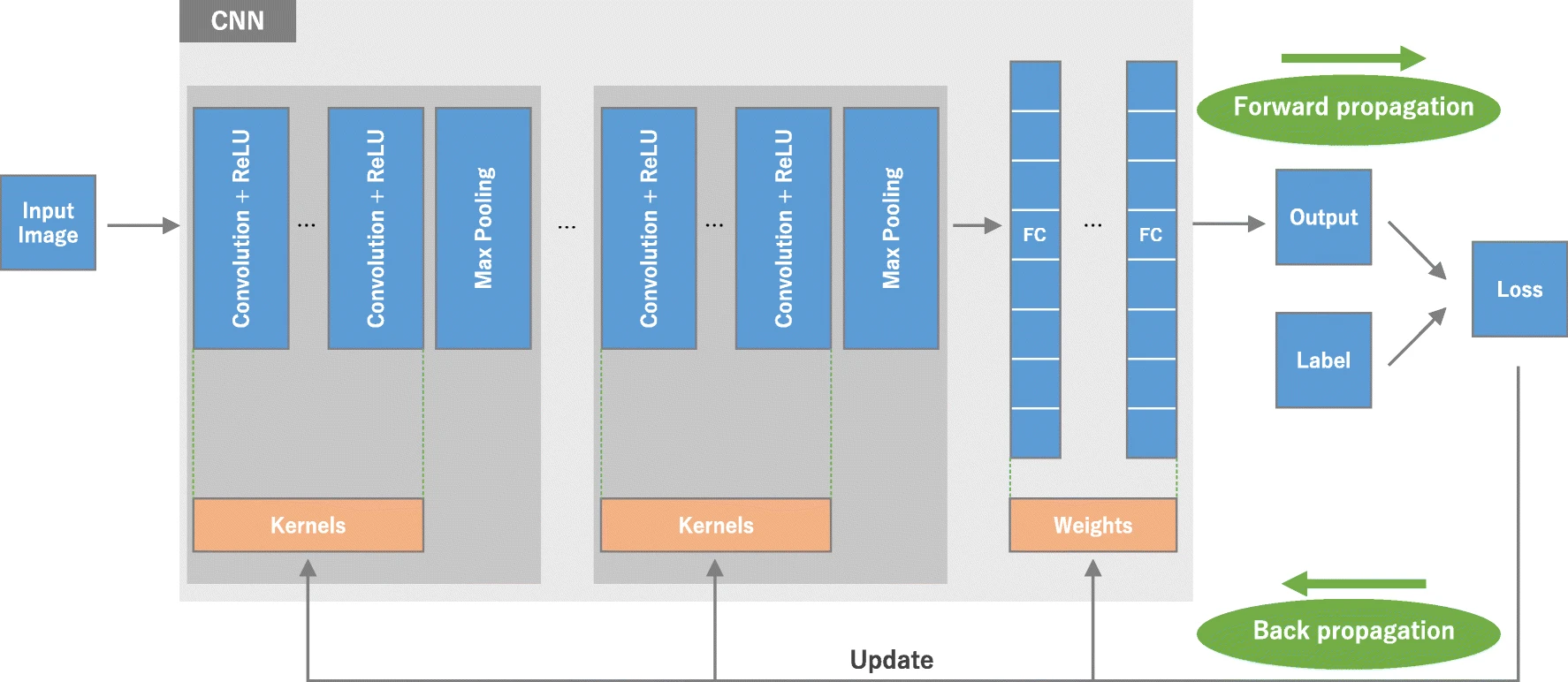
CNN is a mathematical construct that is typically composed of three types of layers (or building blocks): convolution, pooling, and fully connected layers.

The first two, convolution and pooling layers, perform feature extraction, whereas the third, a fully connected layer, maps the extracted features into final output, such as classification.

In digital images, pixel values are stored in a two-dimensional (2D) grid, i.e., an array of numbers, and a small grid of parameters called kernel, an optimizable feature extractor, is applied at each image position, which makes CNNs highly efficient for image processing, since a feature may occur anywhere in the image.



A computer sees an image as an array of numbers. The matrix on the right contains numbers between 0 and 255, each of which corresponds to the pixel brightness in the left image. Both are overlaid in the middle image.



**An overview of a convolutional neural network (CNN) architecture**

**Convolution layer**

A convolution layer is a fundamental component of the CNN architecture that performs feature extraction, which typically consists of a combination of linear and nonlinear operations, i.e., convolution operation and activation function.

**Pooling layer**

A pooling layer provides a typical down-sampling operation which reduces the in-plane dimensionality of the feature maps in order to introduce a translation invariance to small shifts and distortions, and decrease the number of subsequent learnable parameters.

**Max pooling**

The most popular form of pooling operation is max pooling, which extracts patches from the input feature maps, outputs the maximum value in each patch, and discards all the other values.

**Fully connected layer**

The output feature maps of the final convolution or pooling layer is typically flattened, i.e., transformed into a one-dimensional (1D) array of numbers (or vector), and connected to one or more fully connected layers, also known as dense layers, in which every input is connected to every output by a learnable weight.

Once the features extracted by the convolution layers and down-sampled by the pooling layers are created, they are mapped by a subset of fully connected layers to the final outputs of the network, such as the probabilities for each class in classification tasks.

The final fully connected layer typically has the same number of output nodes as the number of classes. Each fully connected layer is followed by a nonlinear function, such as ReLU (Rectified Linear Units), as described above.

Training a network

Backpropagation algorithm is the method commonly used for training neural networks where **loss function** and **gradient descent** optimization algorithm play essential roles.

**Loss function**

A loss function, is also referred to as a cost function.

We define two loss functions; the content loss function and the style loss function.

The **content loss function** ensures that the activations of the higher layers are similar between the content image and the generated image.

The **style loss function** makes sure that the correlation of activations in all the layers are similar between the style image and the generated image.

**Gradient descent**

Gradient descent is an iterative optimization algorithm used in machine learning to minimize a loss function. The loss function describes how well the model will perform given the current set of parameters (weights and biases), and gradient descent is used to find the best set of parameters.

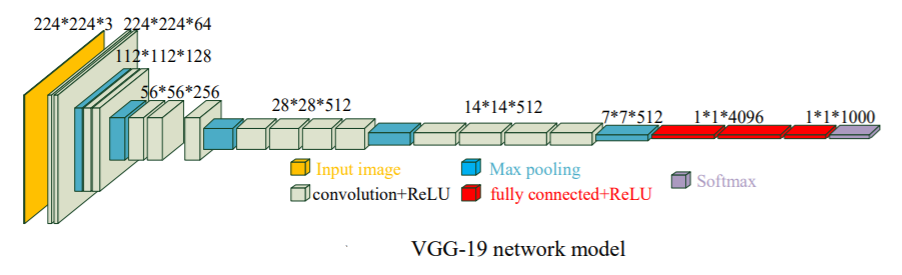
**What is VGG-19**

VGG-19 is a deep convolutional neural network that is 19 layers deep.

VGG19 is a variant of VGG model which in short consists of 19 layers (16 convolution layers, 3 Fully connected layer, 5 MaxPool layers and 1 SoftMax layer). There are other variants of VGG like VGG11, VGG16 and others.

It was created by a group named as **Visual Geometry Group** at **Oxford's** and hence the name VGG.

VGG CNN has six main structures, each of which is mainly composed of multiple connected convolutional layers and full-connected layers. The size of the convolutional kernel is 3\*3, and the input size is 224\*224\*3. The number of layers is generally concentrated at 16~19. The VGG-19 model structure is shown below:



**Applications of CNN / VGG-19:**

Image recognition

Recommender Engines

Handwriting analysis

Climate Change Studies

Data-driven personalized advertising

**PyTorch**

PyTorch is an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab (FAIR).

It is free and open-source software released under the Modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface.

**Modules**

**Autograd module**

PyTorch uses a method called automatic differentiation.

A recorder records what operations have performed, and then it replays it backward to compute the gradients. This method is especially powerful when building neural networks to save time on one epoch by calculating differentiation of the parameters at the forward pass.

**Optim module**

torch.optim is a module that implements various optimization algorithms used for building neural networks. Most of the commonly used methods are already supported, so there is no need to build them from scratch.

**nn module**

PyTorch autograd makes it easy to define computational graphs and take gradients, but raw autograd can be a bit too low-level for defining complex neural networks.

Project Uploaded at:

<https://github.com/Chitranjan806/tvarit_neural_style_transfer>

End of Document