NEW HORIZON
COLLEGE OF ENGINEERING

Module 3

CONVOLUTIONAL NEURAL NETWORK

Syllabus

- CNN Architectures
- Convolution
- Pooling Layers
- Transfer Learning
- Image Classification using Transfer Learning

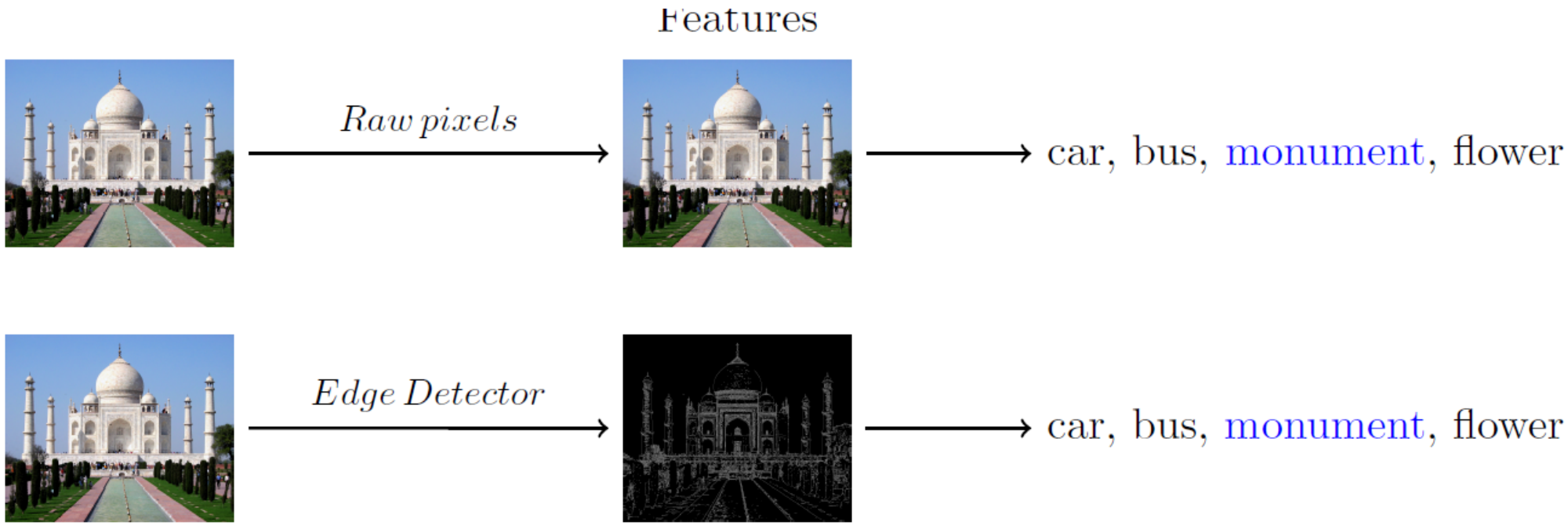
Convolution Neural Network

- One major problem with computer vision problems is that the input data can get really big.
- Suppose an image is of the size 68 X 68 X 3. The input feature dimension then becomes 12,288. This will be even bigger if we have larger images (say, of size 720 X 720 X 3).
- Now, if we pass such a big input to a neural network, the number of parameters will swell up to a HUGE number (depending on the number of hidden layers and hidden units).
- This will result in more computational and memory requirements – not something most of us can deal with.
- The advancements in Computer Vision with Deep Learning lead to one particular algorithm — a Convolutional Neural Network. CNNs have become a very important part of many Computer Vision applications
- Convolutional neural networks are biologically inspired networks that are used in computer vision for image classification and object detection.
- They are used widely for image recognition, object detection/localization, and even text processing.

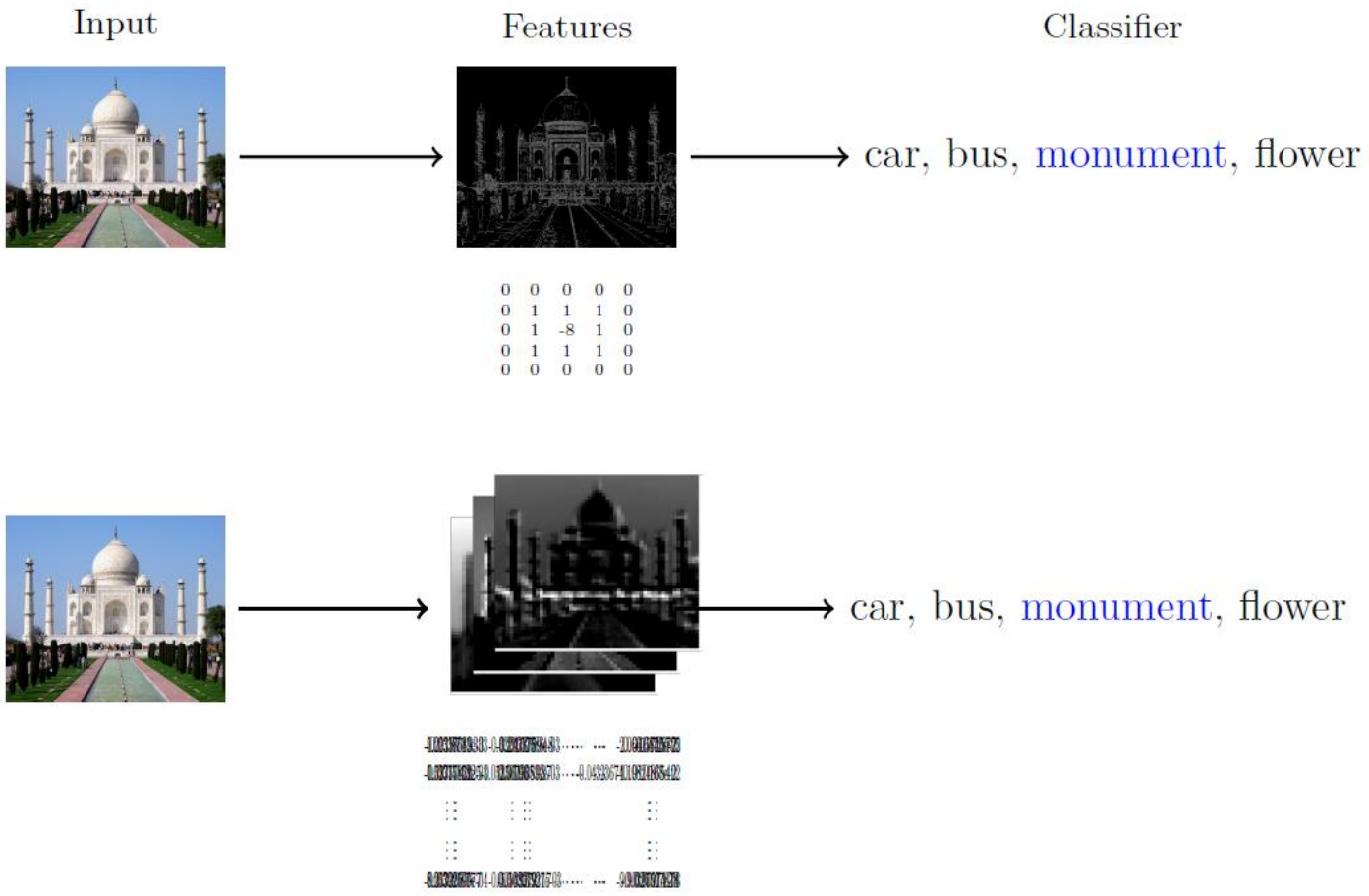
Convolution Neural Network

- CNN's were first developed and used around the 1980s.
- It was mostly used in the postal sectors to read zip codes, pin codes(recognize handwritten digits).
- **Convolutional neural network** (**CNN/ConvNet**) is a class of deep neural networks, most commonly applied to analyze visual imagery
- It uses a special technique called **Convolution.**
- **Convolution** is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.

Edge Detection

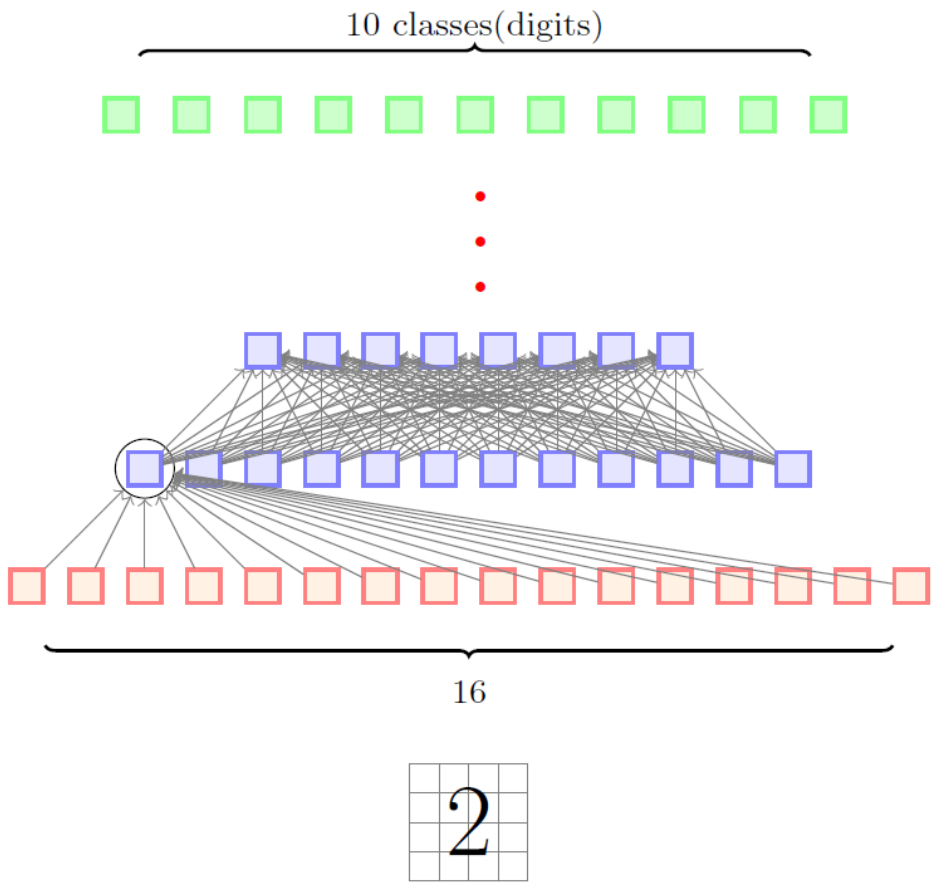


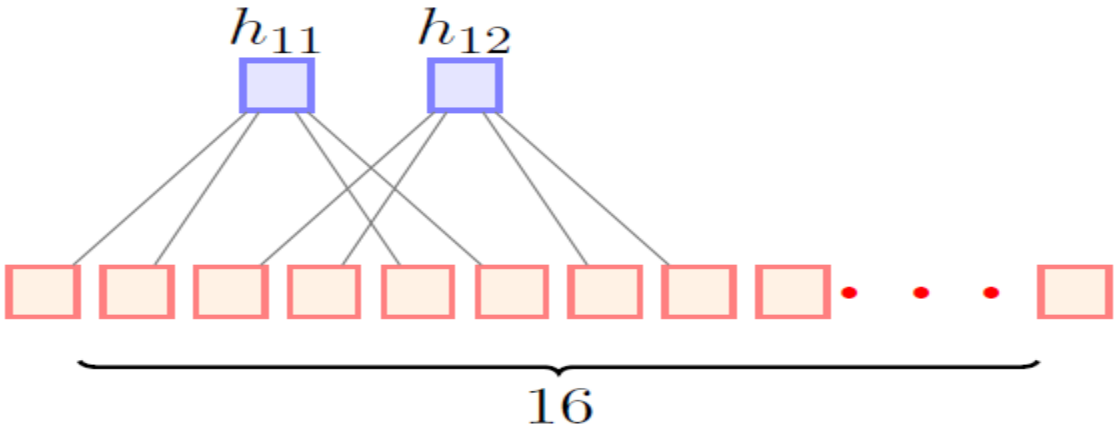
CNN

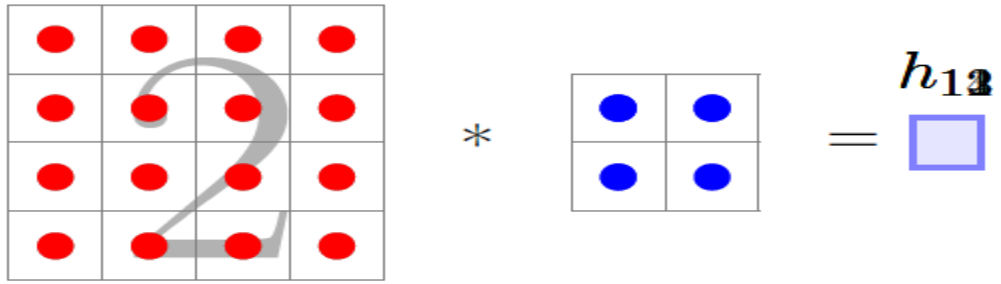


Instead of using handcrafted kernels (such as edge detectors)can we learn multiple meaningful kernels/lters in addition to learning the weights of the classfier

Regular Feed Forward and CNN



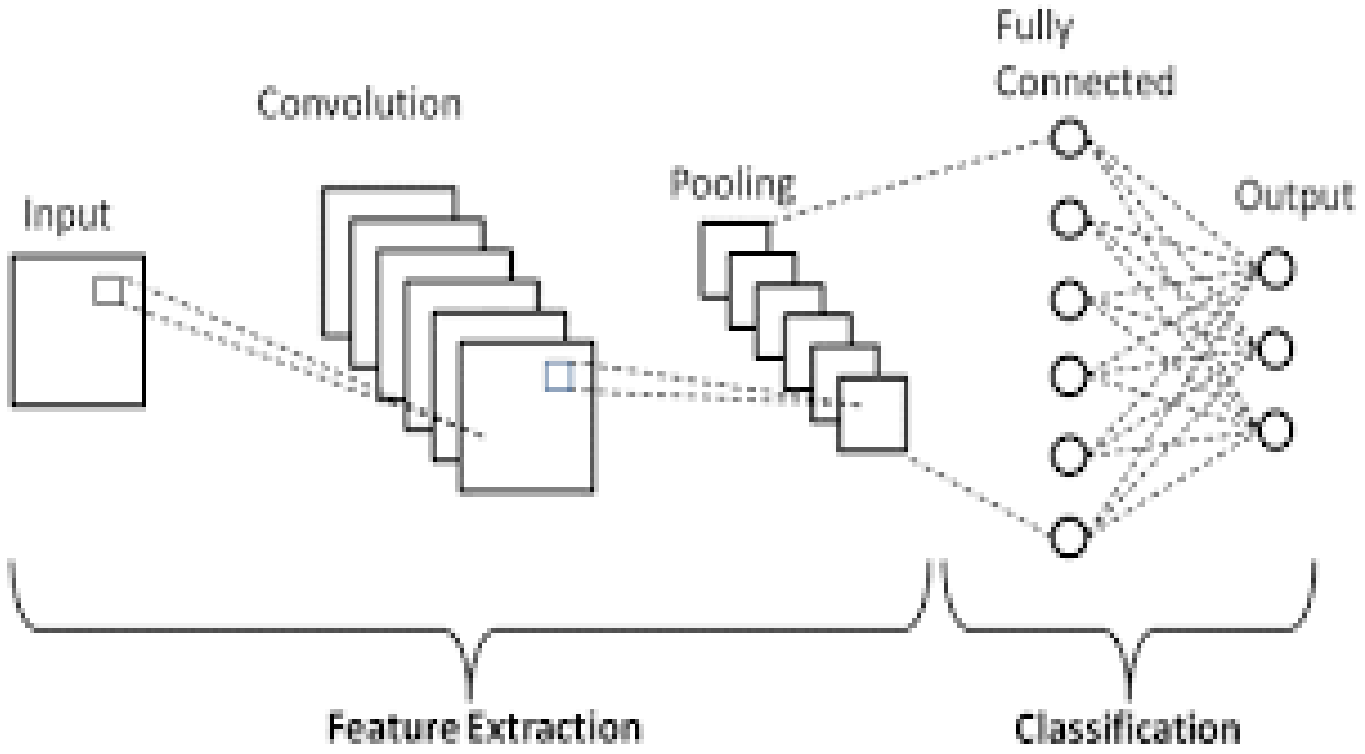




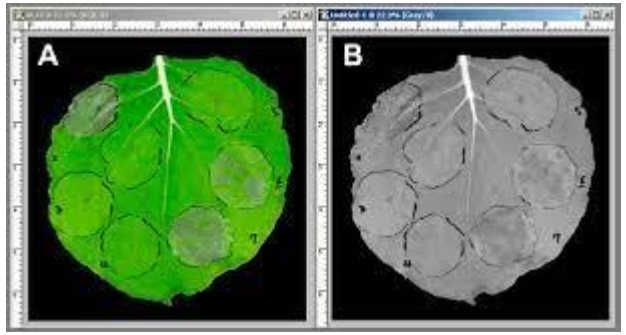
Basic Architecture of CNN

Convolutional Neural Network consists of multiple layers like the input layer

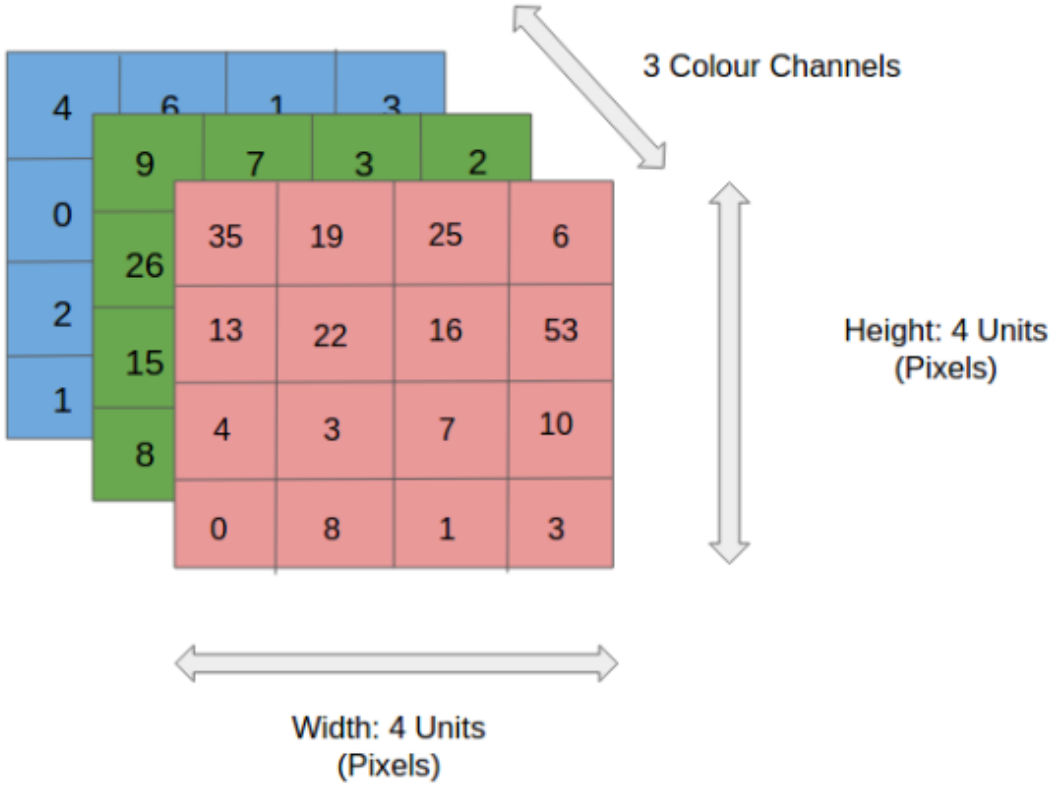
- Convolutional layer
- Pooling layer
- Fully connected layers



Basics -RGB and GrayScale Images



- An RGB image is nothing but a matrix of pixel values having three planes whereas a grayscale image is the same but it has a single plane.



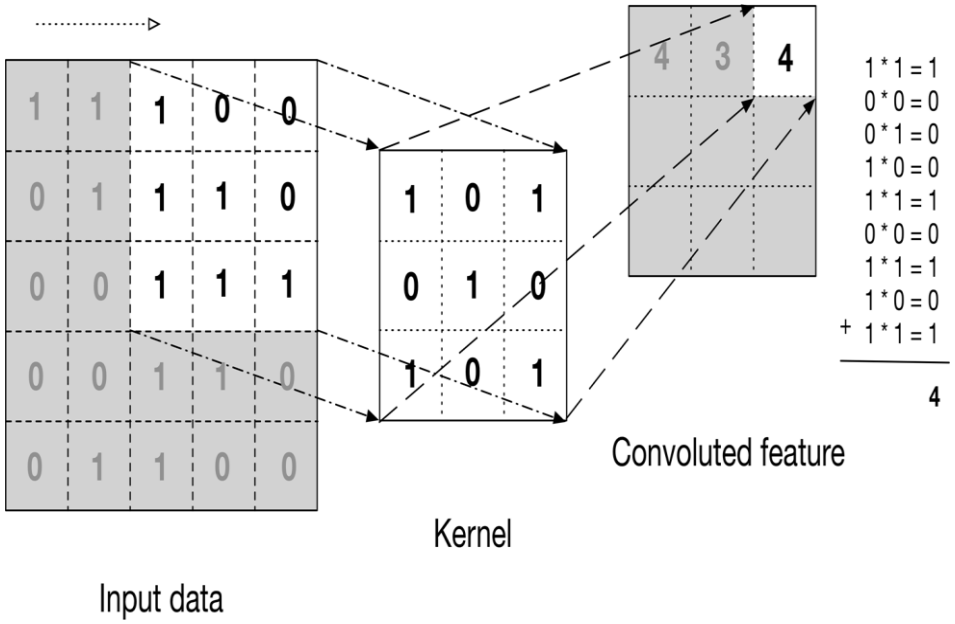
Working of CNN- Convolution Layer

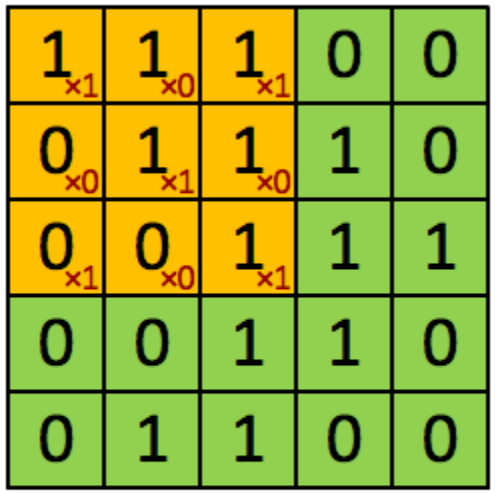
NEW HORIZON
COLLEGE OF ENGINEERING

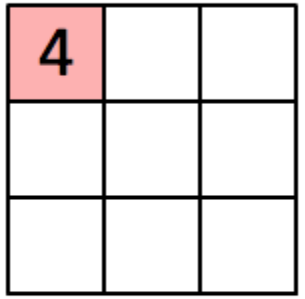
- The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are **input data, a filter, and a feature map**.
- The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image.
- The first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc.
- With added layers, the architecture adapts to the High-Level features also
- Dot product operation is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel.
- The filter is smaller than the input data

Working of CNN- Convolution Layer

- Let us consider the grayscale image with following values







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

Working of CNN- Convolution Layer

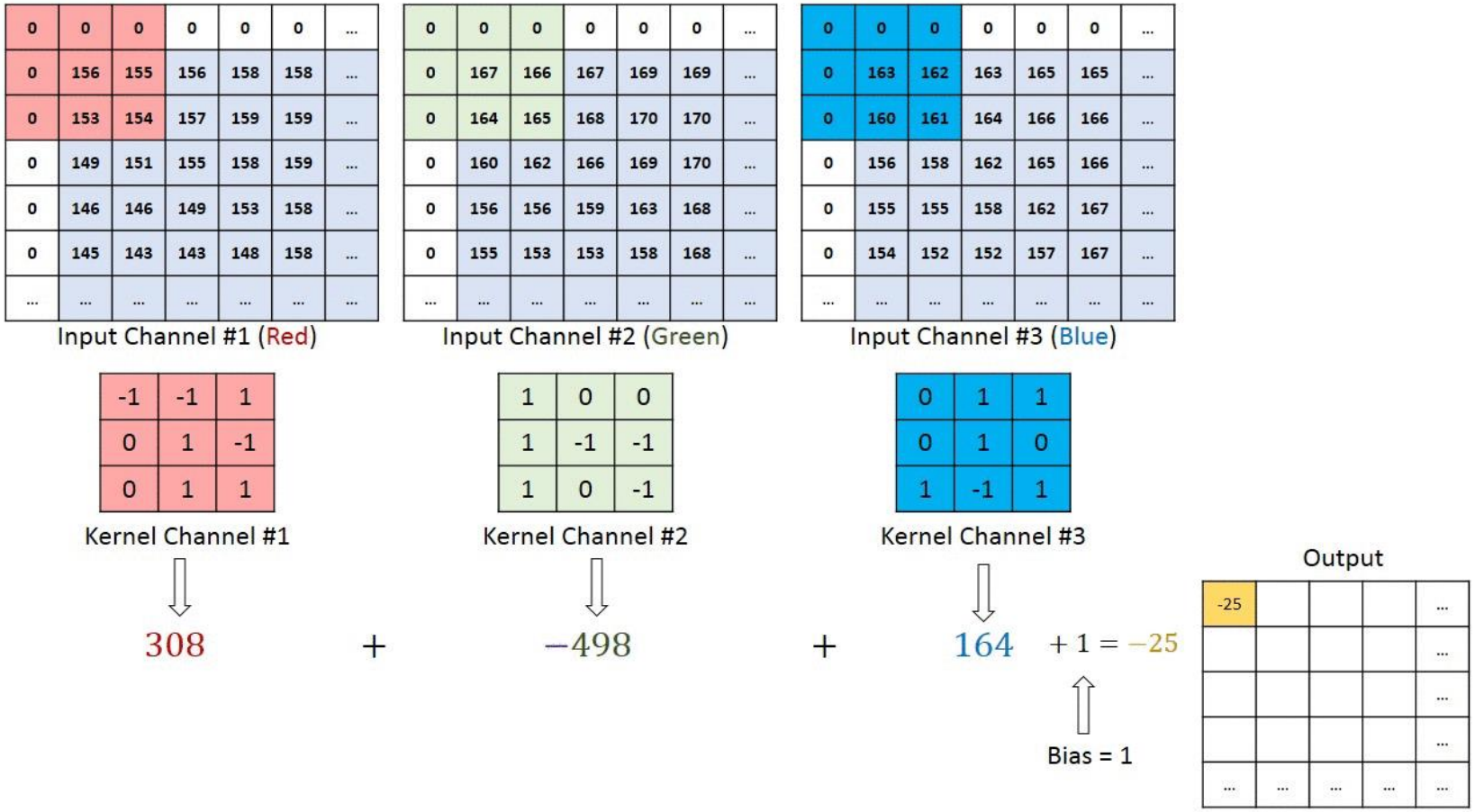
- In the above demonstration, the green section represents **5x5x1 input image, I**. The element involved in the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter, K**, represented in color yellow. We have selected **K as a 3x3x1 matrix.**
- The filter moves to the right with a certain Stride Value till it parses the complete width

```
Kernel/Filter, K =

1  0  1
0  1  0
1  0  1
```

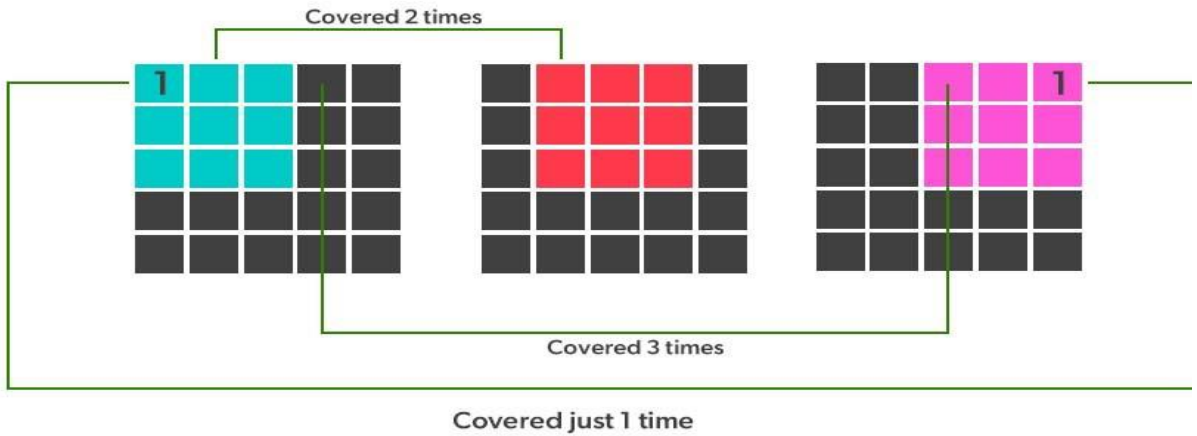
The Kernel shifts 9 times because of **Stride Length = 1,** every time performing an **elementwise multiplication operation**

Convolution Neural Network-Convolution Layer

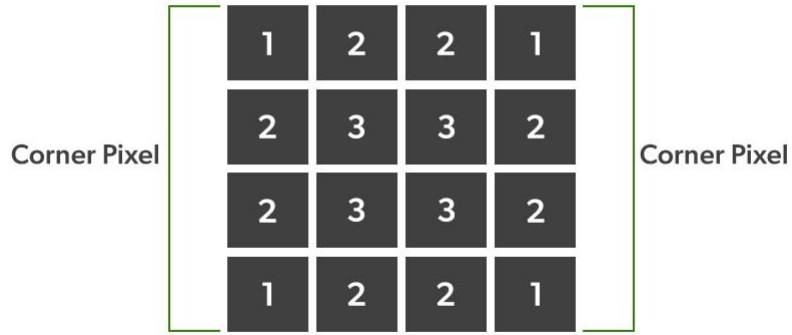


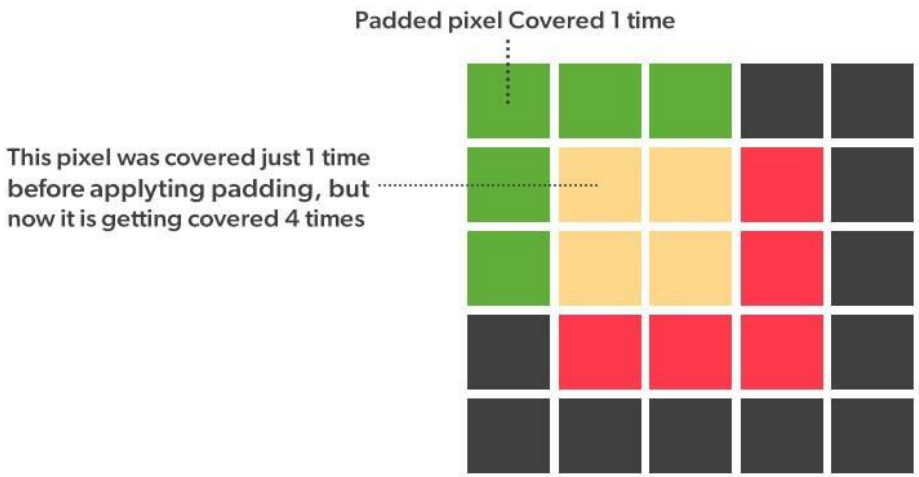
Convolution Neural Network-Convolution Layer-Padding

- The main aim of the network is to find the important feature in the image with the help of convolutional layers but it may happen that some features are at the corner of the image which the kernel (feature extractor) visit very less number of times due to which there could be a possibility to miss out some of the important information.
- So, padding is the solution where we add extra pixels around the 4 corners of the image which increases the image size by 2
- Padding should be as neutral as possible which means do not alter the original image features in any way to make it complicated for the network to learn further.



How Many time Each Pixel is Covered





Convolution Neural Network-Convolution Layer-Padding

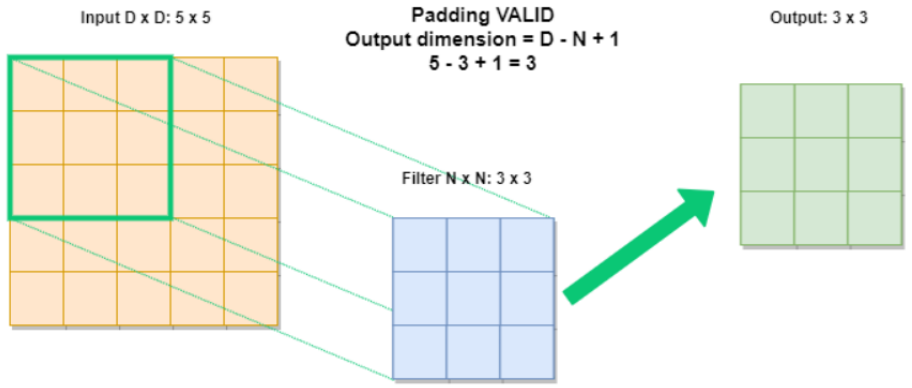
- **Valid Padding** where we are not applying padding at all & **P=0**

Eg:5x5x1 and then apply the 3x3x1 kernel over it, we find that the convolved matrix turns out to be of dimensions 3x3x1.

Ouput=n-f+1

- n= size of input image
- F=size of filter

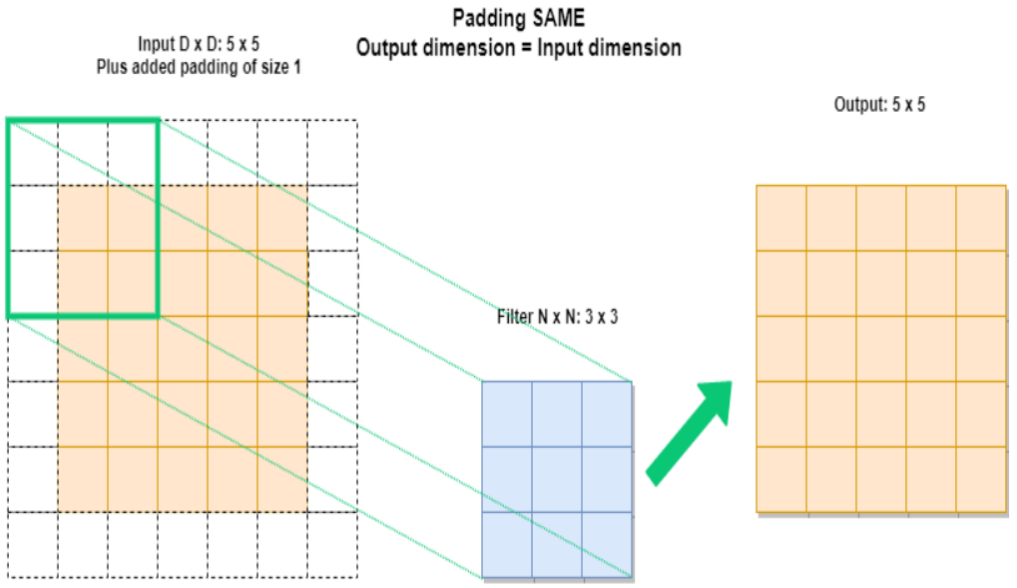
5-3+1



Padding in CNN

Same Padding-where output images size should be equal to the input image size due to which we have applied the same padding.

Same mode preserves the size of the image after process by convolution. So that input and output from the convolutional layer will be of the same size



The size of output image after padding is calculated as

n + 2p – f + 1

- p = number of layers of zeros added to the border
- of the image
- n= size of input image
- F=size of filter
- Eg:(8 x 8) image and using a (3 x 3) filter
- 8+2(1)-3+1 =8

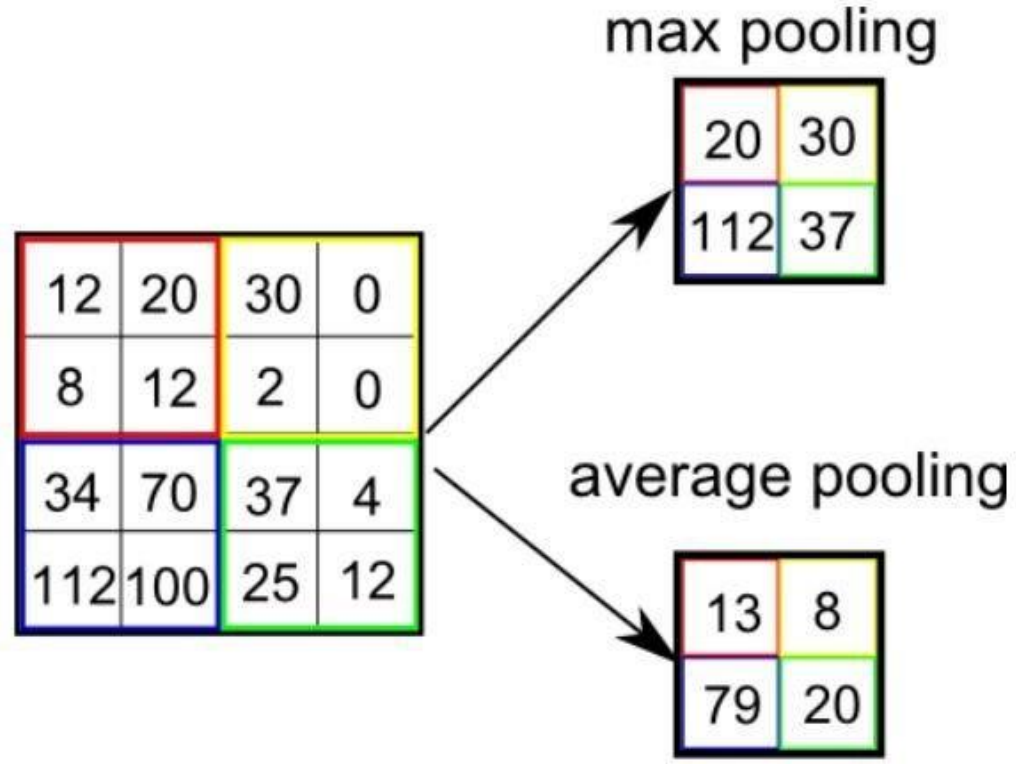
- When we have a stride value, the above formula can be rewritten as
- **Size of output image is** $\frac{n+2p-f}{s}$
- Example image size 125*49 , filter size 5*5, padding p=2, and stride s=2. The output dimensions are W= (125+4-5)/2 =62
- H= (49+4-5)/2 = 24
- So the output activation map dimension (62,24)

Convolution Neural Network-Pooling

- Pooling layer is responsible for reducing the spatial size of the Convolved Feature.
- Pooling layers, also known as downsampling, conducts dimensionality reduction, reducing the number of parameters in the input.
- pooling operation sweeps a filter across the entire input, but the difference is that this filter does not have any weights.
- Instead, the kernel applies an aggregation function to the values within the receptive field, populating the output array.

Convolution Neural Network-Pooling

- There are two main types of pooling:
- **Max pooling:** As the filter moves across the input, it selects the pixel with the maximum value to send to the output array.This approach tends to be used more often compared to average pooling.
- **Average pooling:** As the filter moves across the input, it calculates the average value within the receptive field to send to the output array.



Pooling

1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

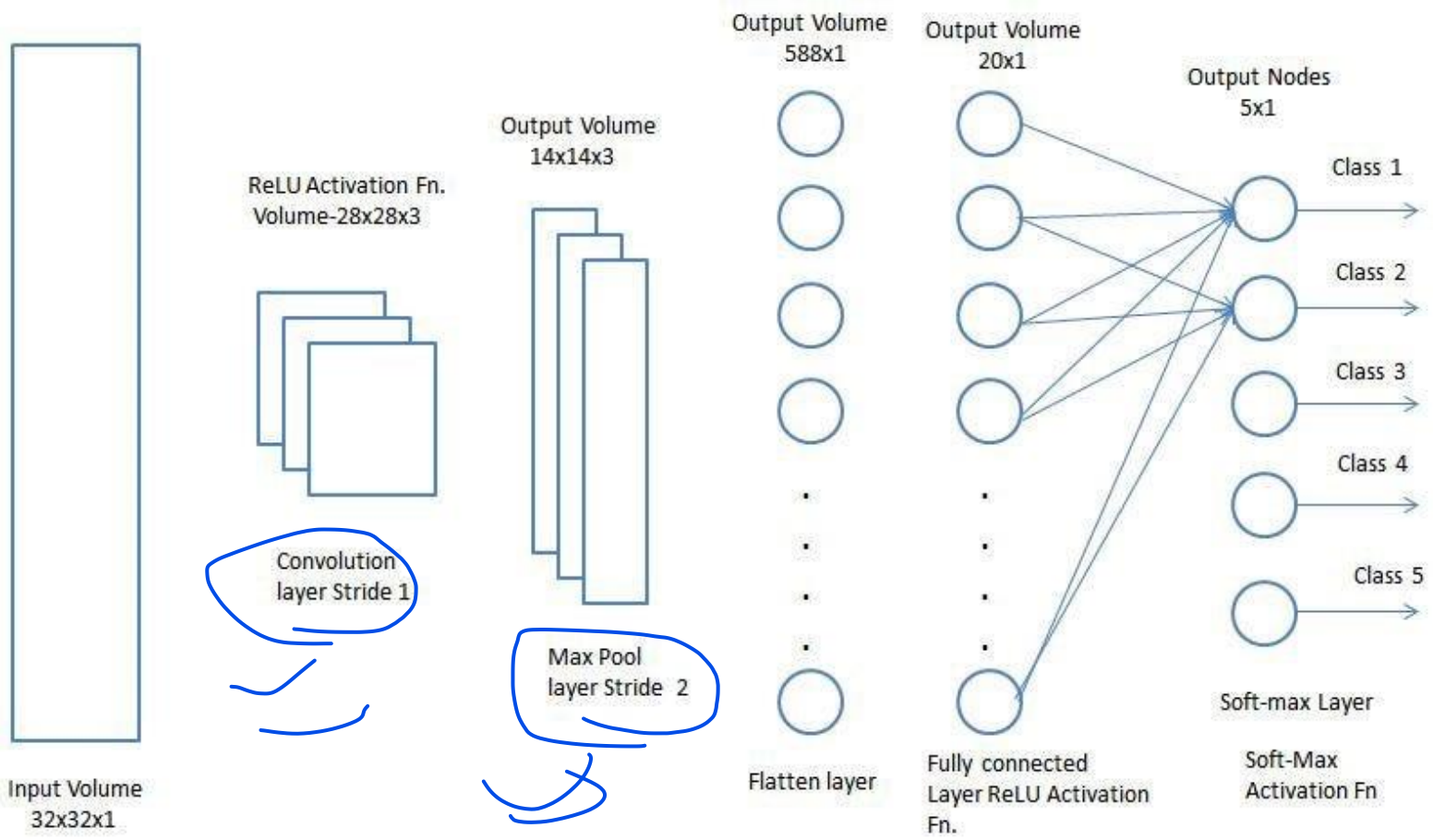
8 4
7 5

Perform max pooling with a stride of two and filter size 2*2

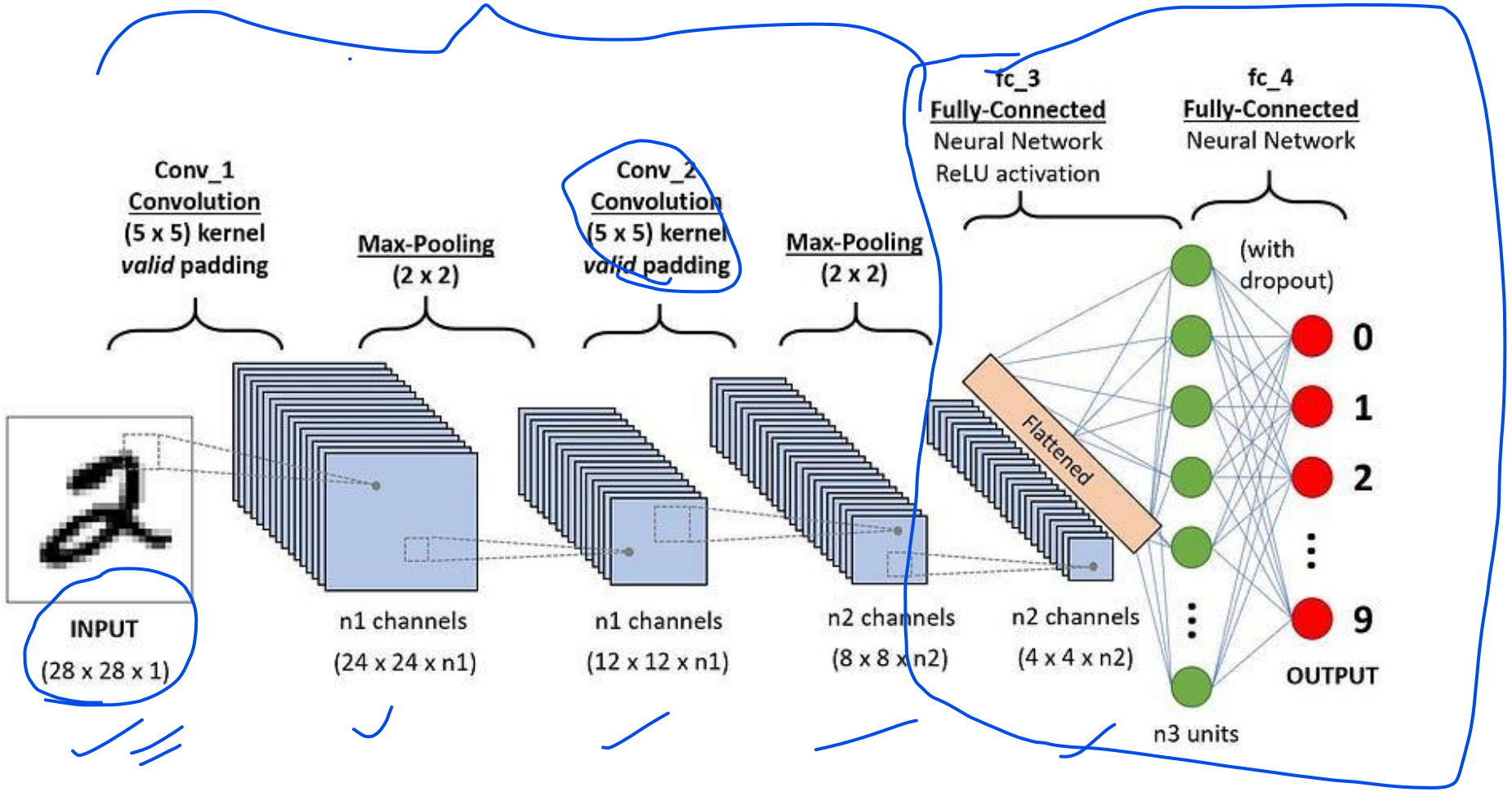
Perform average pooling with a stride of one and filter size 2*2

4.5 4.25 2.5
6.5 5.25 4
4.25 3.5 3

Classification-Fully Connected Layer



A CNN sequence to classify handwritten digits



CNN Architectures

- There are various architectures of CNNs available

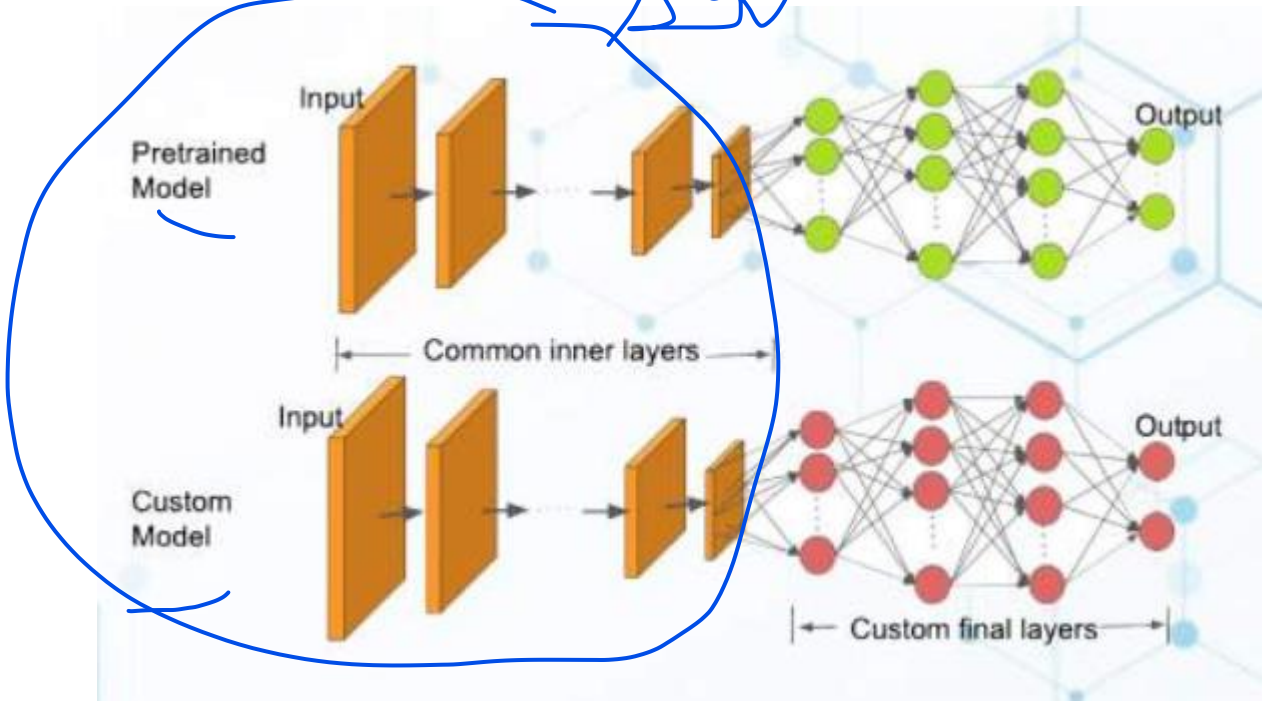
1. LeNet
2. AlexNet
3. VGGNet
4. GoogLeNet
5. ResNet
6. ZFNet

Transfer Learning

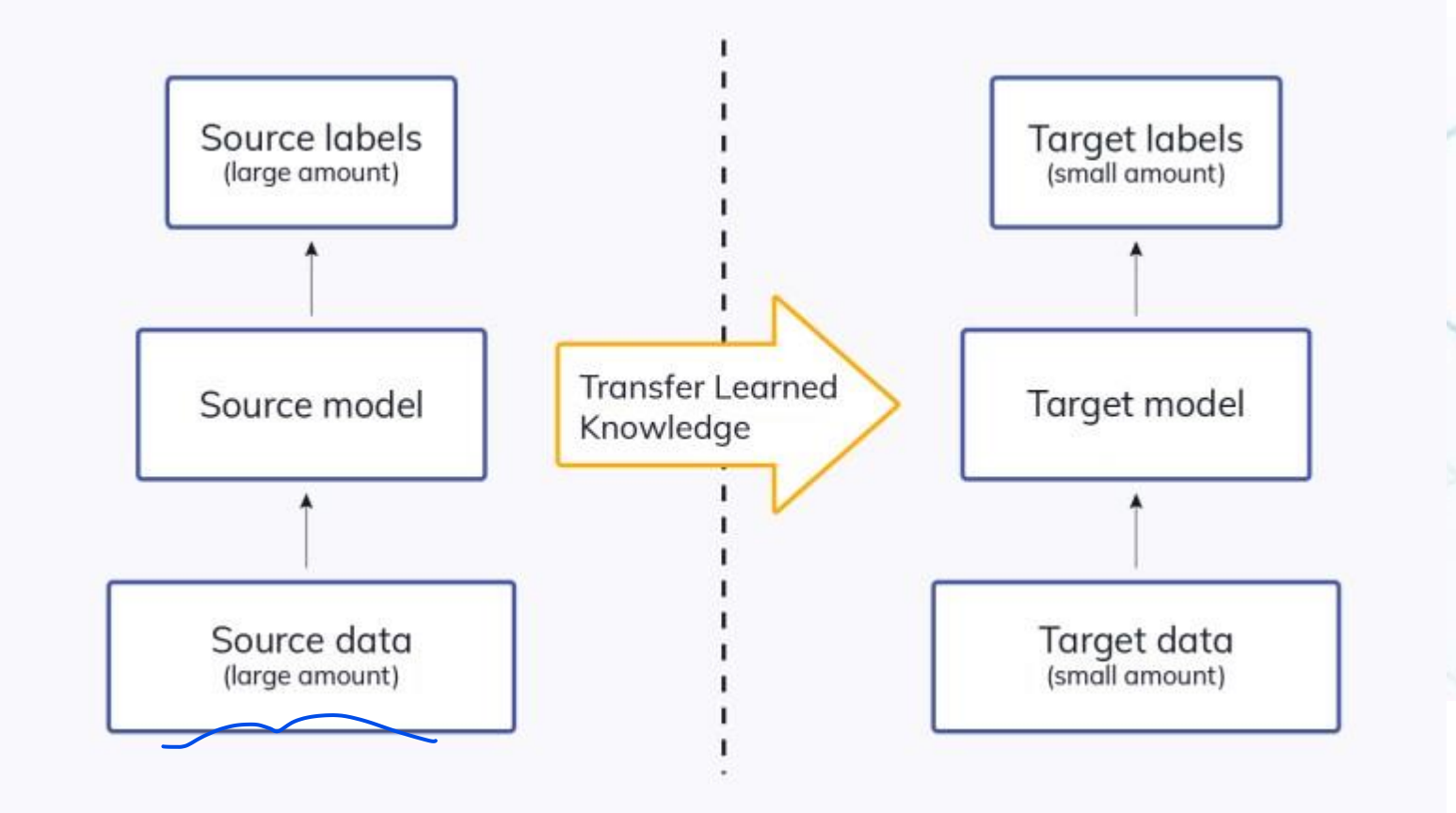
- Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task.
- The knowledge of an already trained machine learning model is transferred to a different but closely linked problem throughout transfer learning
- It is the most common approach used in Computer Vision and Natural Language Processing where models are used as the starting point for other problems so as to save time.

For example, consider the ResNet model of Computer Vision that is trained on the "ImageNet" dataset having 14 million images that can be used for Image Classification tasks directly with changes in fully-connected layers only for deciding the out

Transfer Learning



Transfer Learning



ImageNet

- The ImageNet project is a large visual database designed for use in visual object recognition software research.
- ImageNet contains more than 20,000 categories

Eg: balloon, strawberry

AI researcher Fei-Fei Li began working on the idea for ImageNet in 2006

Li wanted to expand and improve the data available to train AI algorithms.

Approaches of Transfer Learning

Two common approaches are as follows:

1. Develop Model Approach
2. Pre-trained Model Approach

- **Develop Model Approach**

1. Select Source Task. Select a related predictive modeling problem with abundance of data where there is some relationship in the input data, output data, and/or concepts learned during the mapping from input to output data.
2. Develop Source Model. Develop a skillful model for this first task. The model must be better than a naive model to ensure that some feature learning has been performed.
3. Reuse Model. The model fit on the source task can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modeling technique used.
4. Tune Model. Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.

Approaches in Transfer Learning

- Pre-trained Model Approach

1. Select Source Model. A pre-trained source model is chosen from available models. Many research institutions release models on large and challenging datasets that may be included in the pool of candidate models from which to choose from.
2. Reuse Model. The model pre-trained model can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modeling technique used.
3. Tune Model. Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.

- This second type of transfer learning is common in the field of deep learning.
-

- Examples of the areas of machine learning that utilise transfer learning include:
- Natural language processing
- Computer vision
- Neural networks
- The advantage of pre-trained models is that they are generic enough for use in other real-world applications.
- For example:

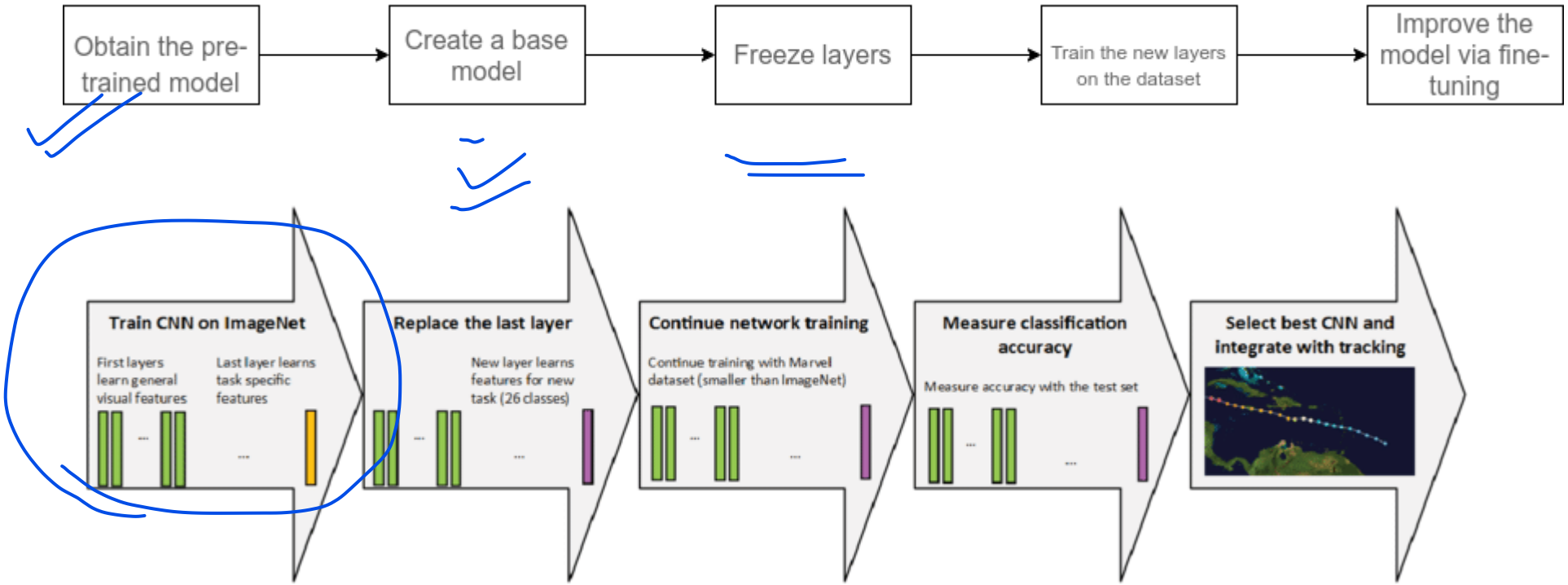
Models trained on the ImageNet can be used in real-world image classification problems. This is because the dataset contains over 1000 classes. An insect researcher can use these models and fine-tune them to classify insects.

Which When and How to Transfer

NEW HORIZON
COLLEGE OF ENGINEERING

- Before deciding on the strategy of transfer learning, it is crucial to have an answer of the following questions:
- ***Which*** part of the knowledge can be transferred from the source to the target to improve the performance of the target task?
- ***When*** to transfer and when not to, so that one improves the target task performance/results and does not degrade them?
- ***How*** to transfer the knowledge gained from the source model based on our current domain/task?

1. implement transfer learning in these six general steps.



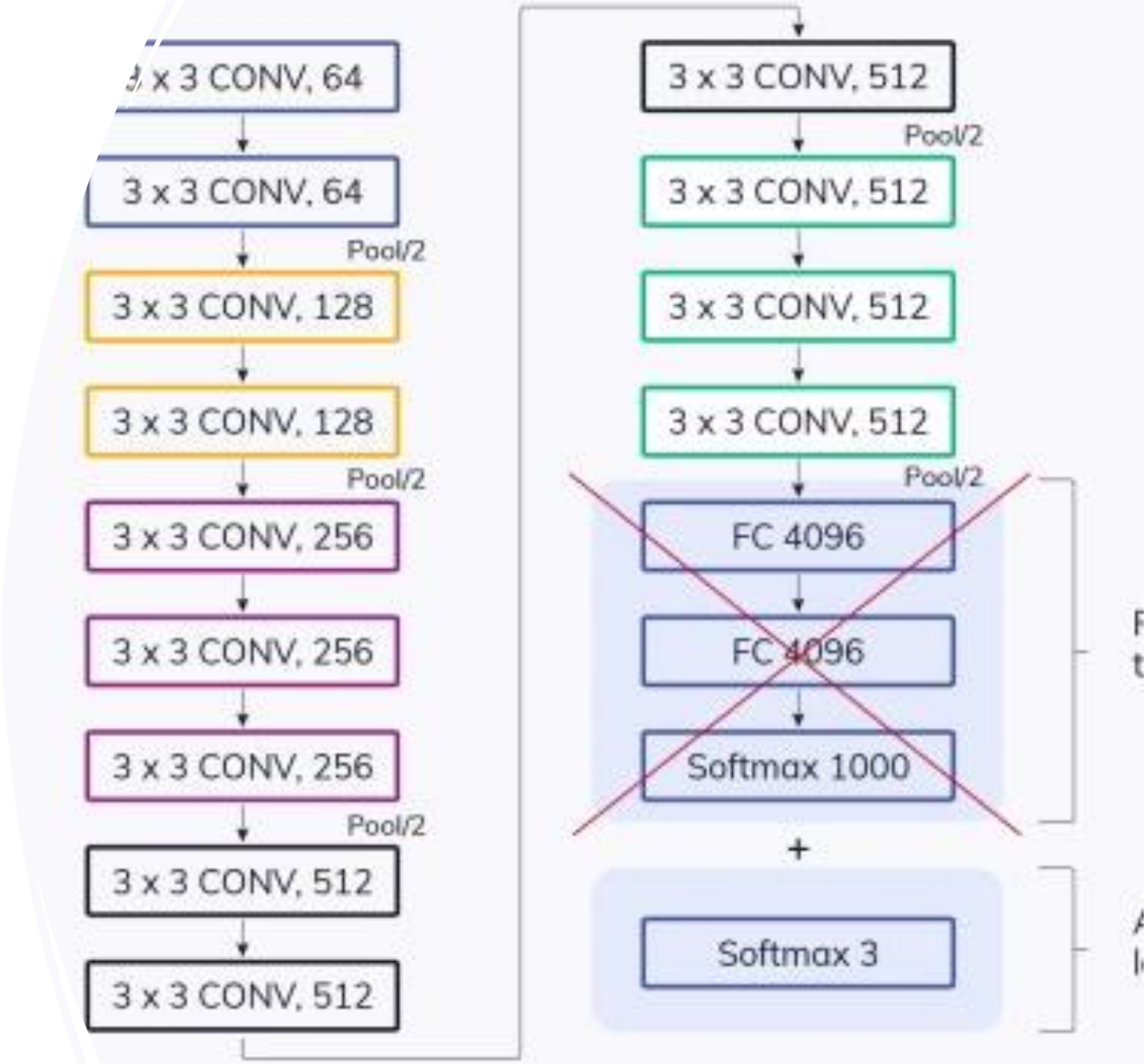
Transfer Learning

- **Obtain the pre-trained model**

The first step is to get the pre-trained model that we would like to use for our problem. The various sources of pre-trained models are covered in a separate section.

- **Create a base model**

The first step is to instantiate the **base model** using one of the architectures such as ResNet or Xception

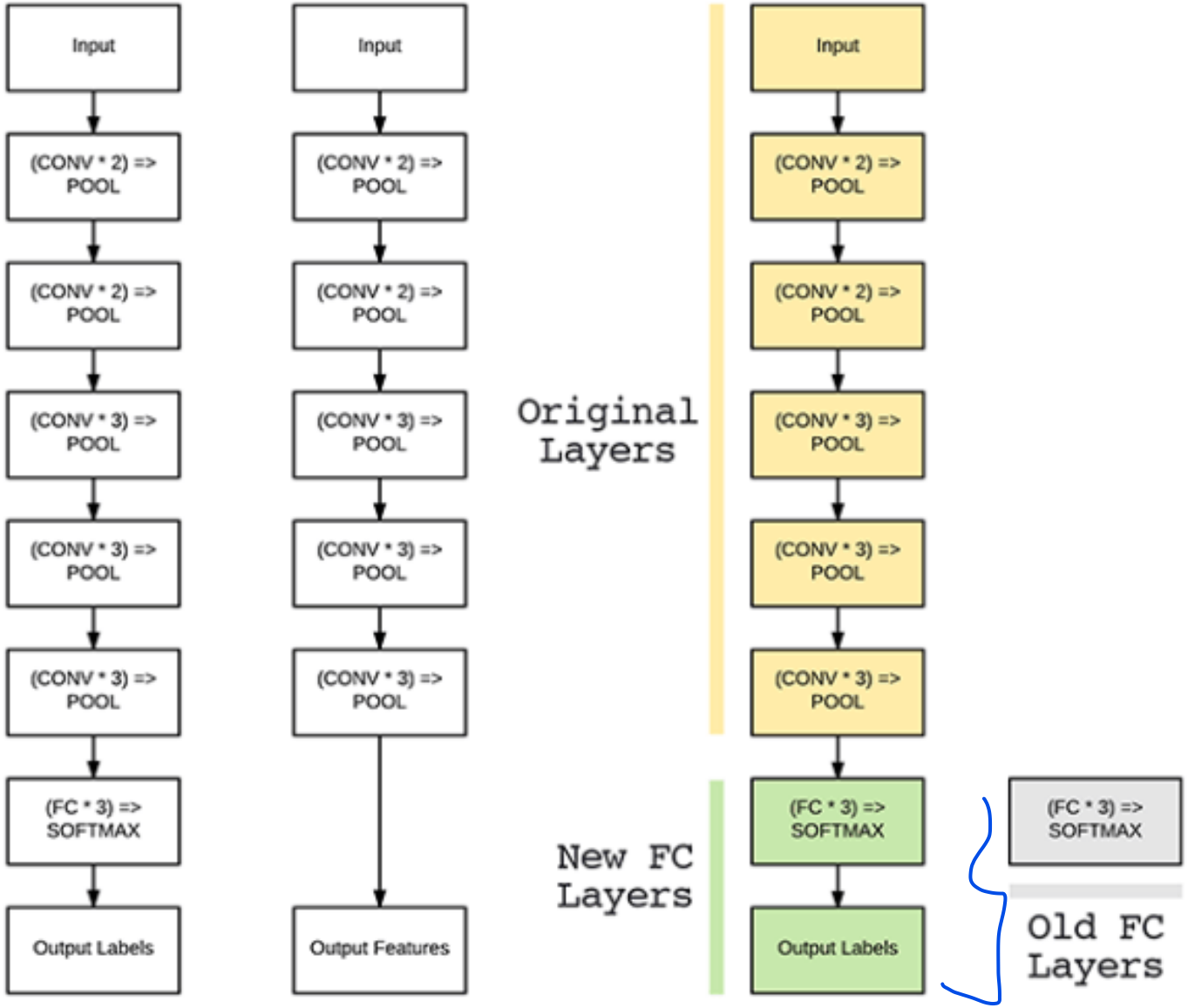


Transfer Learning

- **Add new trainable layers**

The next step is to **add new trainable layers** that will turn old features into predictions on the new dataset.

This is important because the pre-trained model is loaded without the final output layer.



Transfer Learning

NEW HORIZON
COLLEGE OF ENGINEERING

- **Train the new layers on the dataset**
- The pre-trained model's final output will most likely be different from the output that you want for your model.
- For example, pre-trained models trained on the ImageNet dataset will output 1000 classes. Our model might just have two classes. In this case, we have to train the model with a new output layer in place.
- Therefore, we will add some new dense layers.A final dense layer with units corresponding to the number of outputs expected by your model.

Transfer Learning

- **Improve the model via fine-tuning**
- We have a model that can make predictions on our dataset.
- Improve its **performance through fine-tuning**. Fine-tuning is done by unfreezing the base model or part of it and training the entire model again on the whole dataset at a very low learning rate.
- The low learning rate will increase the performance of the model on the new dataset while preventing overfitting.

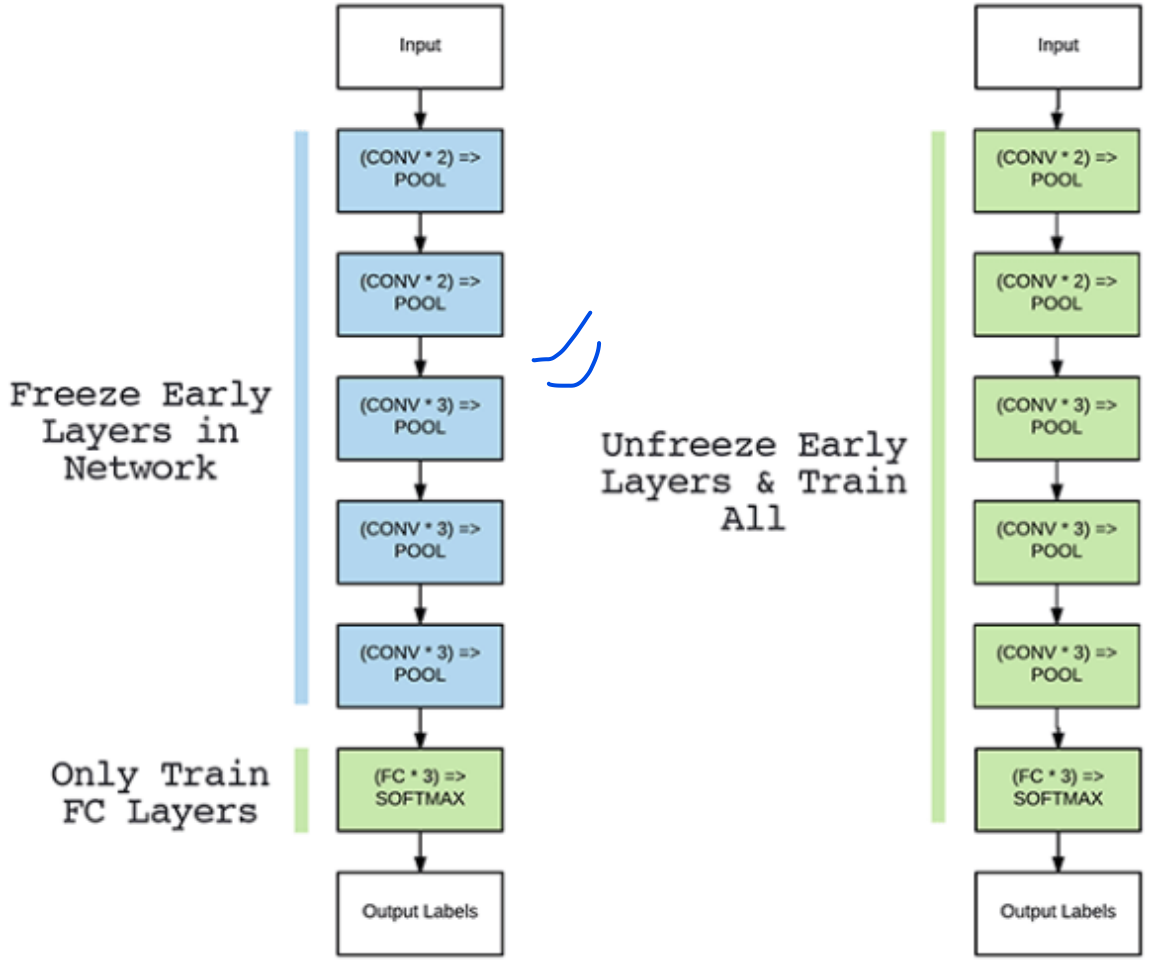
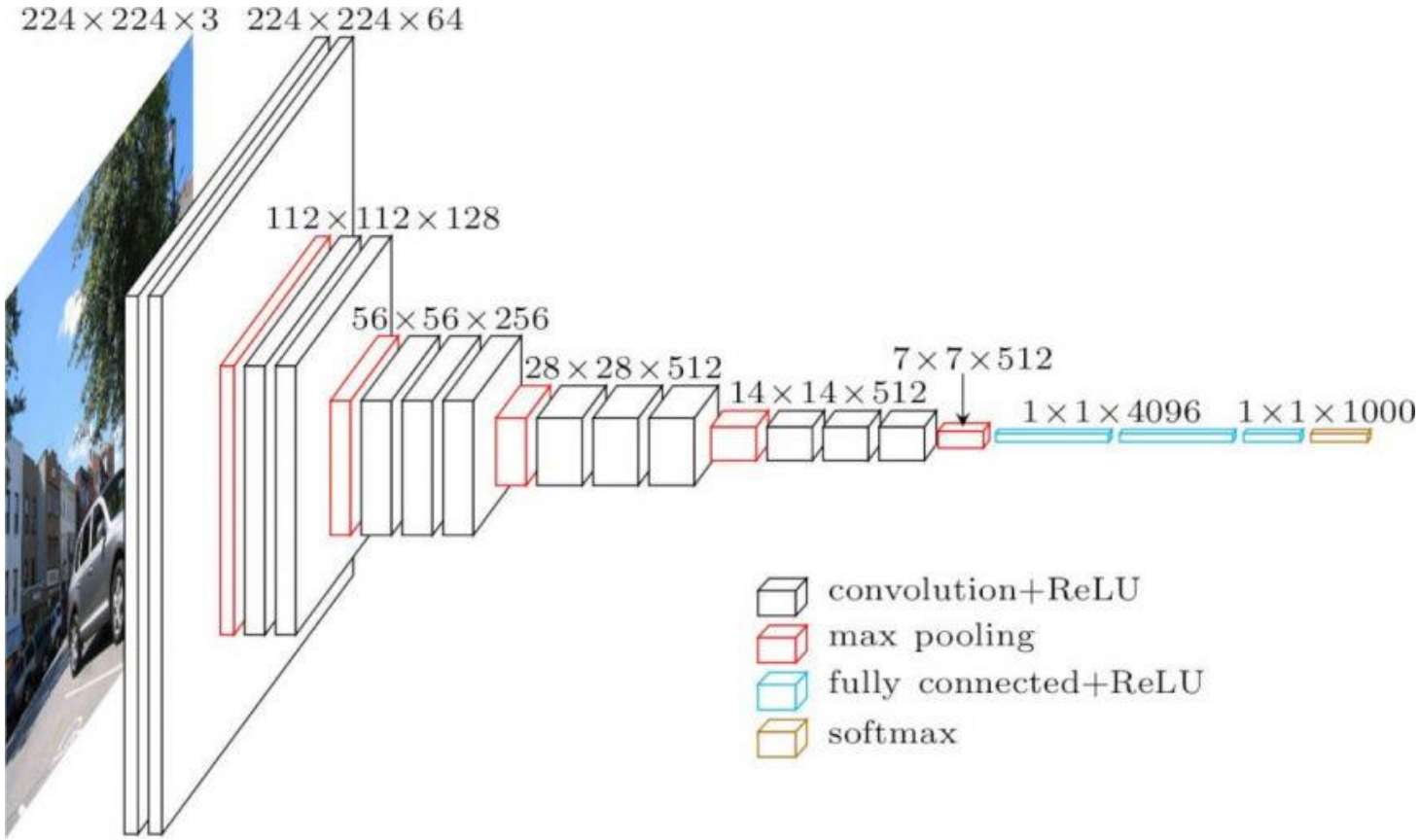


Image Classification using Transfer Learning

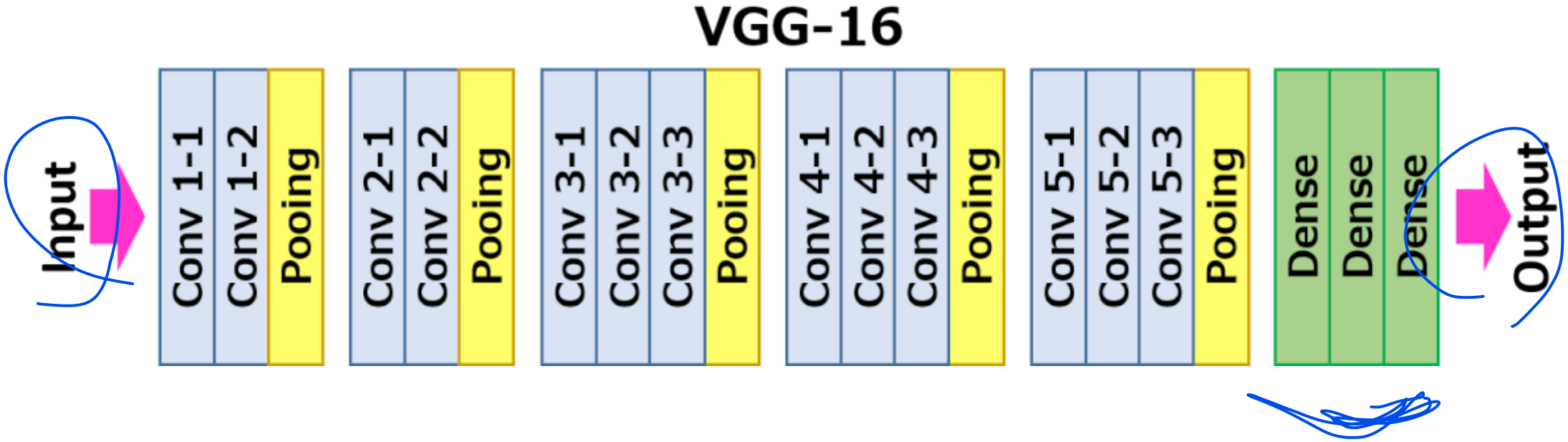
- Pre-Trained Models for Image Classification
  - VGG-16
  - ResNet50
  - Inceptionv3
  - EfficientNet

Pre-Trained Models for Image Classification

- 1. Very Deep Convolutional Networks for Large-Scale Image Recognition(VGG-16)
- The VGG-16 is one of the most popular pre-trained models for image classification.



Pre-Trained Models for Image Classification



The following are the layers of the model:

- Convolutional Layers = 13
- Pooling Layers = 5
- Dense Layers = 3

Pre-Trained Models for Image Classification

1. **Input**: Image of dimensions (224, 224, 3).
2. **Convolution Layer Conv1:**
    1. Conv1-1: 64 filters
    2. Conv1-2: 64 filters and Max Pooling
    3. Image dimensions: (224, 224)
3. **Convolution layer Conv2:** Now, we increase the filters to 128
    1. Input Image dimensions: (112,112)
    2. Conv2-1: 128 filters
    3. Conv2-2: 128 filters and Max Pooling
4. **Convolution Layer Conv3:** Again, double the filters to 256, and now add another convolution layer
    1. Input Image dimensions: (56,56)
    2. Conv3-1: 256 filters
    3. Conv3-2: 256 filters
    4. Conv3-3: 256 filters and Max Pooling

5. **Convolution Layer Conv4:** Similar to Conv3, but now with 512 filters
    1. Input Image dimensions: (28, 28)
    2. Conv4-1: 512 filters
    3. Conv4-2: 512 filters
    4. Conv4-3: 512 filters and Max Pooling
6. **Convolution Layer Conv5:** Same as Conv4
    1. Input Image dimensions: (14, 14)
    2. Conv5-1: 512 filters
    3. Conv5-2: 512 filters
    4. Conv5-3: 512 filters and Max Pooling
    5. The output dimensions here are (7, 7). At this point, we flatten the output of this layer to generate a feature vector

1. **Fully Connected/Dense FC1**: 4096 nodes, generating a feature vector of size(1, 4096)
2. **Fully ConnectedDense FC2**: 4096 nodes generating a feature vector of size(1, 4096)
3. **Fully Connected /Dense FC3**: 4096 nodes, generating 1000 channels for 1000 classes. This is then passed on to a Softmax activation function
4. **Output layer**

Pre-Trained Models for Image Classification

NEW HORIZON COLLEGE OF ENGINEERING

- The model is sequential in nature and uses lots of filters. At each stage, small 3 * 3 filters are used to reduce the number of parameters all the hidden layers use the ReLU activation function. Even then, the number of parameters is **138 Billion** – which makes it a slower and much larger model to train than others.
- Additionally, there are variations of the VGG16 model, which are basically, improvements to it, like VGG19 (19 layers). You can find a detailed explanation

Training a VGG-16 model on the dataset-

Step 1: Image Augmentation

Step 2: Training and Validation Sets

Step 3: Loading the Base Model

Step 4: Compile and Fit

Approaches in Transfer Learning

- Pre-trained Model Approach

1. Select Source Model. A pre-trained source model is chosen from available models. Many research institutions release models on large and challenging datasets that may be included in the pool of candidate models from which to choose from.
2. Reuse Model. The model pre-trained model can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modeling technique used.
3. Tune Model. Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.

- This second type of transfer learning is common in the field of deep learning.
-

Models for Transfer Learning

NEW HORIZON COLLEGE OF ENGINEERING

- There are models for image recognition that can be downloaded and used as the basis for image recognition and related computer vision tasks.
- VGG (e.g. VGG16 or VGG19).
- GoogLeNet (e.g. InceptionV3).
- Residual Network (e.g. ResNet50).
- Performance of these models were good. So they are widely used transfer learning models.
- Keras provides access to a number of top-performing pre-trained models that were developed for image recognition tasks.

Including VGG model

- Eg:
- Load model without output layer
- *model = VGG16(include_top=False)*
- When the “include_top” argument is False, the “input_tensor” argument must be specified, allowing the expected fixed-sized input of the model to be changed.

For example:

- # load model and specify a new input shape for images
- *new_input = Input(shape=(640, 480, 3))*
- model = VGG16(include_top=False, input_tensor=new_input)

- A model without a top will output activations from the last convolutional or pooling layer directly
- One approach to summarizing these activations for their use in a classifier by adding a global pooling layer, such as a max global pooling or average global pooling.
- The result is a vector that can be used as a feature descriptor for an input. Keras provides this capability directly via the ‘pooling‘ argument that can be set to ‘avg‘ or ‘max‘. For example:
- # load model and specify a new input shape for images and avg pooling output
- new_input = Input(shape=(640, 480, 3))
- model = VGG16(include_top=False, input_tensor=new_input, pooling='avg')

VGG16

- Images can be prepared for a given model using the preprocess_input() function

- For example:

- # prepare an image
- from keras.applications.vgg16 import preprocess_input
- images = ...
- prepared_images = preprocess_input(images)

- Finally, you may wish to use a model architecture on your dataset, but not use the pre-trained weights, and instead initialize the model with random weights and train the model from scratch.
- This can be achieved by setting the ‘weights‘ argument to None instead of the default ‘imagenet‘.
- ‘classes‘ argument can be set to define the number of classes in your dataset, which will then be configured for you in the output layer of the model. For example:
- ...
- # define a new model with random weights and 10 classes
- new_input = Input(shape=(640, 480, 3))
- model = VGG16(weights=None, input_tensor=new_input, classes=10)

Pre-Trained Model as Classifier

NEW HORIZON
COLLEGE OF ENGINEERING

- We will use the VGG16 model to classify new images.
- First, the photograph needs to loaded and reshaped to a 224×224 square, expected by the model, and the pixel values scaled in the way expected by the model. The model operates on an array of samples, therefore the dimensions of a loaded image need to be expanded by 1, for one image with 224×224 pixels and three channels.

- # example of using a pre-trained model as a classifier
- from keras.preprocessing.image import load_img
- from keras.preprocessing.image import img_to_array
- from keras.applications.vgg16 import preprocess_input
- from keras.applications.vgg16 import decode_predictions
- from keras.applications.vgg16 import VGG16
- # load an image from file
- image = load_img('dog.jpg', target_size=(224, 224))
- # convert the image pixels to a numpy array
- image = img_to_array(image)
- # reshape data for the model
- image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
- # prepare the image for the VGG model
- image = preprocess_input(image)
- # load the model
- model = VGG16()

- # predict the probability across all output classes
- yhat = model.predict(image)
- # convert the probabilities to class labels
- label = decode_predictions(yhat)
- # retrieve the most likely result, e.g. highest probability
- label = label[0][0]
- # print the classification
- print('%s (%.2f%%)' % (label[1], label[2]*100))

NEW HORIZON
COLLEGE OF ENGINEERING

- Examples of the areas of machine learning that utilise transfer learning include:
- Natural language processing
- Computer vision
- Neural networks
- The advantage of pre-trained models is that they are generic enough for use in other real-world applications.
- For example:

Models trained on the ImageNet can be used in real-world image classification problems. This is because the dataset contains over 1000 classes. An insect researcher can use these models and fine-tune them to classify insects.

Transfer Learning Approaches

- There are two common approaches to transfer learning: **Feature Extraction** and **Fine-Tuning**.
- 1. **Feature Extraction**
- In feature extraction, the pre-trained model is used to extract features from new data. Only the final classification layer is retrained on the new dataset. The earlier layers, which contain the learned features, are frozen and not updated during training.
- Steps:

1. Load the Pre-trained Model:
   - Load the model without the top classification layers (set include_top=False).
2. Add Custom Classification Layers:
   - Add new layers to the model that are specific to your new task.
3. Freeze the Pre-trained Layers:
   - Set the layers of the pre-trained model to non-trainable.
4. Compile and Train the Model:
   - Compile the model and train only the new classification layers.

- from tensorflow.keras.applications import VGG16
- from tensorflow.keras.models import Model
- from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
- from tensorflow.keras.optimizers import Adam

- # Load the pre-trained VGG16 model without the top layers
- base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

- # Add custom classification layers
- x = base_model.output
- x = GlobalAveragePooling2D()(x)
- x = Dense(1024, activation='relu')(x)
- predictions = Dense(10, activation='softmax')(x) # Assume 10 classes for the new task

- # Create the full model
- model = Model(inputs=base_model.input, outputs=predictions)

- # Freeze the pre-trained layers
- for layer in base_model.layers:
- layer.trainable = False

- # Compile the model
- model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

- # Train the model
- model.fit(train_data, train_labels, epochs=10, validation_data=(val_data, val_labels))

- 2. Fine-Tuning
- In fine-tuning, the pre-trained model is adapted more extensively by unfreezing some of the top layers and retraining them along with the new classification layers. This allows the model to better adjust to the new task.
- Steps:

1. Load the Pre-trained Model:
   - Load the model without the top classification layers (set include_top=False).
2. Add Custom Classification Layers:
   - Add new layers to the model that are specific to your new task.
3. Freeze Initial Layers:
   - Initially, freeze most of the pre-trained layers, keeping a few top layers trainable.
4. Compile and Train the Model:
   - Compile the model and train the top layers and the new classification layers.
5. Unfreeze Some Layers and Retrain:
   - Optionally, unfreeze more layers and continue training with a lower learning rate.

- from tensorflow.keras.applications import VGG16
- from tensorflow.keras.models import Model
- from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
- from tensorflow.keras.optimizers import Adam

- # Load the pre-trained VGG16 model without the top layers
- base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

- # Add custom classification layers
- x = base_model.output
- x = GlobalAveragePooling2D()(x)
- x = Dense(1024, activation='relu')(x)
- predictions = Dense(10, activation='softmax')(x) # Assume 10 classes for the new task

- # Create the full model
- model = Model(inputs=base_model.input, outputs=predictions)
- # Freeze the initial layers
- for layer in base_model.layers:
-     layer.trainable = False

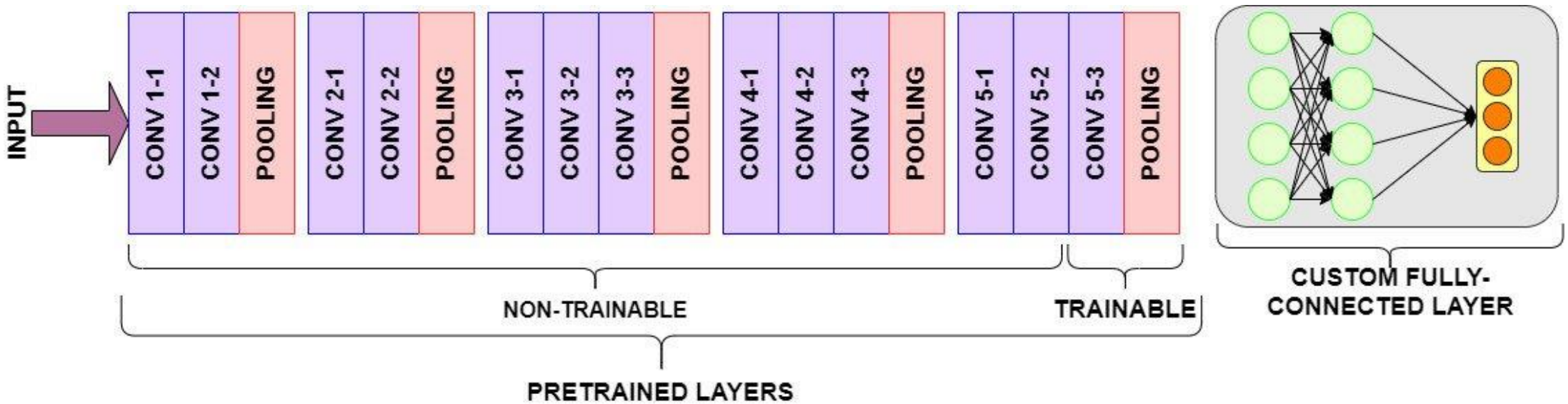
- # Compile the model
- model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

- # Train the model
- model.fit(train_data, train_labels, epochs=10, validation_data=(val_data, val_labels))

- # Unfreeze some of the top layers and retrain
- for layer in base_model.layers[-4:]:
- layer.trainable = True

- # Recompile the model with a lower learning rate
- model.compile(optimizer=Adam(1e-5), loss='categorical_crossentropy', metrics=['accuracy'])

- # Continue training
- model.fit(train_data, train_labels, epochs=10, validation_data=(val_data, val_labels))


INPUT
CONV 1-1
CONV 1-2
POOLING
CONV 2-1
CONV 2-2
POOLING
CONV 3-1
CONV 3-2
CONV 3-3
POOLING
CONV 4-1
CONV 4-2
CONV 4-3
POOLING
CONV 5-1
CONV 5-2
CONV 5-3
POOLING
NON-TRAINABLE
TRAINABLE
PRETRAINED LAYERS
CUSTOM FULLY-CONNECTED LAYER