UNIT 3

CNN & ITS CONCEPTS

CNN :

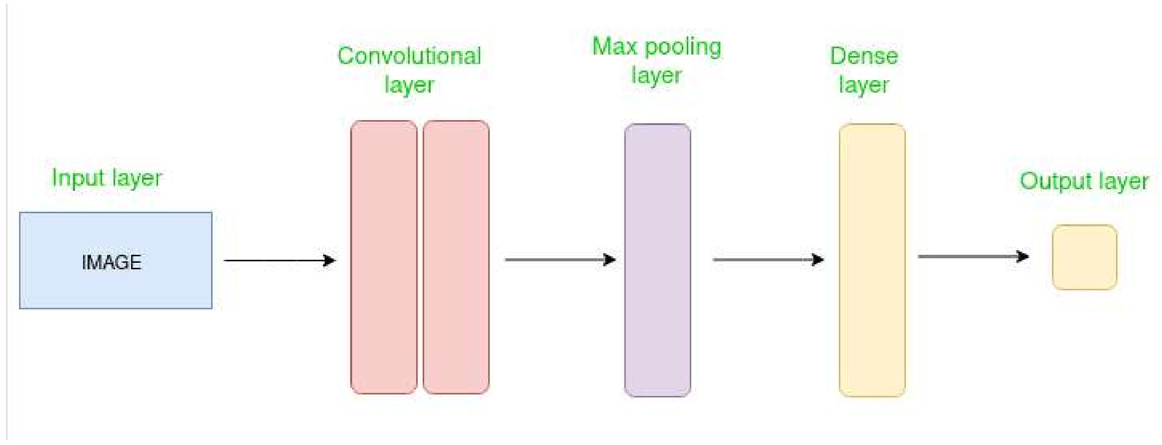
- CNN stands for Convolutional Neural Network.

- It is a type of deep learning neural network that is specifically designed to process and analyze data with a grid-like structure, such as images, videos, and time series data.

- The main idea behind CNNs is to exploit the spatial structure of the input data by using convolutional layers.

- These layers apply a set of learnable filters (also called kernels) to the input data, performing local operations such as feature detection. By using these filters, CNNs can automatically learn hierarchical representations of the input data, capturing both low-level features (e.g., edges, textures) and high-level features (e.g., shapes, objects).

- CNNs have achieved remarkable success in various computer vision tasks and have become a fundamental tool in the field of deep learning.



Feature Scaling :

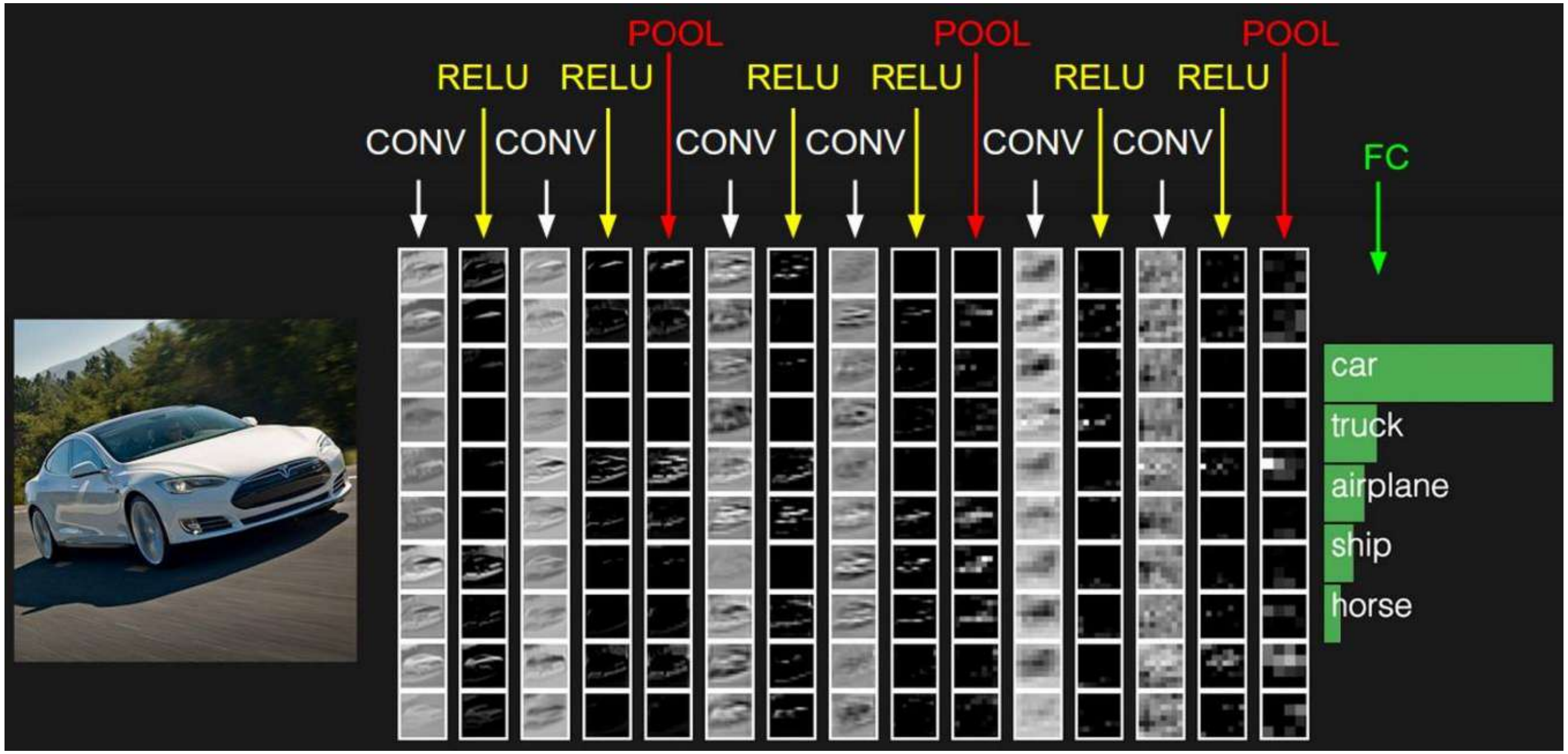
Feature scaling in CNNs typically involves normalizing the pixel values of the input images. The most common method is to perform min-max scaling, where the pixel values are rescaled to a specific range, often between 0 and 1. This normalization ensures that the pixel values have a consistent scale and prevents any single feature (e.g., pixel intensity) from dominating the learning process.

The min-max scaling process involves the following steps:

1. Convert the pixel values of the input images to a common range, such as between 0 and 255.
2. Normalize the pixel values by subtracting the minimum pixel value and dividing by the range (maximum value minus minimum value). This scales the pixel values to a range between 0 and 1.

It's important to note that the scaling should be applied consistently to both the training data and the test data.

Unbalanced influence of features:
Slow convergence:
Gradient explosion or vanishing
Model performance degradation:


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

<u>MAX POOLING :</u>

- The process of max pooling involves dividing the input feature map into non-overlapping rectangular regions (usually called pooling windows or kernels) and taking the maximum value within each region.

- The size and stride of the pooling window determine the amount of downsampling.

- Max pooling is a pooling operation commonly used in convolutional neural networks (CNNs) to downsample feature maps.

- It helps reduce the spatial dimensions of the input while retaining the most salient features. Max pooling is typically applied after convolutional layers in CNN architectures.

The process of max pooling involves dividing the input feature map into non-overlapping rectangular regions (usually called pooling windows or kernels) and taking

the maximum value within each region. The size and stride of the pooling window determine the amount of downsampling.

Here's a step-by-step explanation of how max pooling works:

1. Input Feature Map: Consider an input feature map from a previous convolutional layer. It is a 3-dimensional tensor with dimensions (height, width, channels). Each channel represents a specific feature or filter response.
2. Pooling Window: Define a rectangular pooling window with a specified size (e.g., 2x2). The window slides over the input feature map, moving by a certain stride value (e.g., 2), and extracts the maximum value within each window.
3. Maximum Value Extraction: Within each pooling window, the maximum value is selected. This value represents the most prominent feature within that region.
4. Output Feature Map: The output of max pooling is a downsampled feature map with reduced spatial dimensions. The size of the output feature map depends on the input size, pooling window size, and stride. The number of channels remains the same.

PADDING IN CNN :

Padding in convolutional neural networks (CNNs) is a technique used to preserve spatial dimensions during the convolutional operation. It involves adding extra border pixels (padding) around the input image or feature map before applying convolutional filters. Padding helps maintain the size of the output feature map, especially when using filters with a larger receptive field.

There are two common types of padding:

1. Valid Padding (No Padding): In this approach, no padding is added to the input image or feature map. The convolutional filters are applied only to the valid pixels, resulting in an output feature map with reduced spatial dimensions compared to the input. The reduction in size depends on the filter size and the stride used in the convolution operation. This type of padding is called "valid" because it produces valid output only for pixels where the entire filter fits within the input.

2. Same Padding: Same padding ensures that the output feature map has the same spatial dimensions as the input. It achieves this by adding an equal amount of padding pixels symmetrically around the input. The amount of padding is

determined by the filter size and the desired output size. For a filter of size F and a desired output size O, the amount of padding P can be calculated as:

P = (F - 1) / 2

The formula ensures that an equal number of padding pixels is added to both sides of the input. Same padding is called "same" because it preserves the spatial dimensions and allows for a more straightforward alignment of features between the input and output feature maps.

Padding serves several purposes in CNNs:

1. Preservation of Spatial Information
2. Prevention of Information Loss:

Strides in CNN :

In convolutional neural networks (CNNs), strides refer to the step size used when moving the convolutional filter/kernel over the input feature map during the convolution operation. The stride determines the amount of spatial displacement between consecutive applications of the filter.

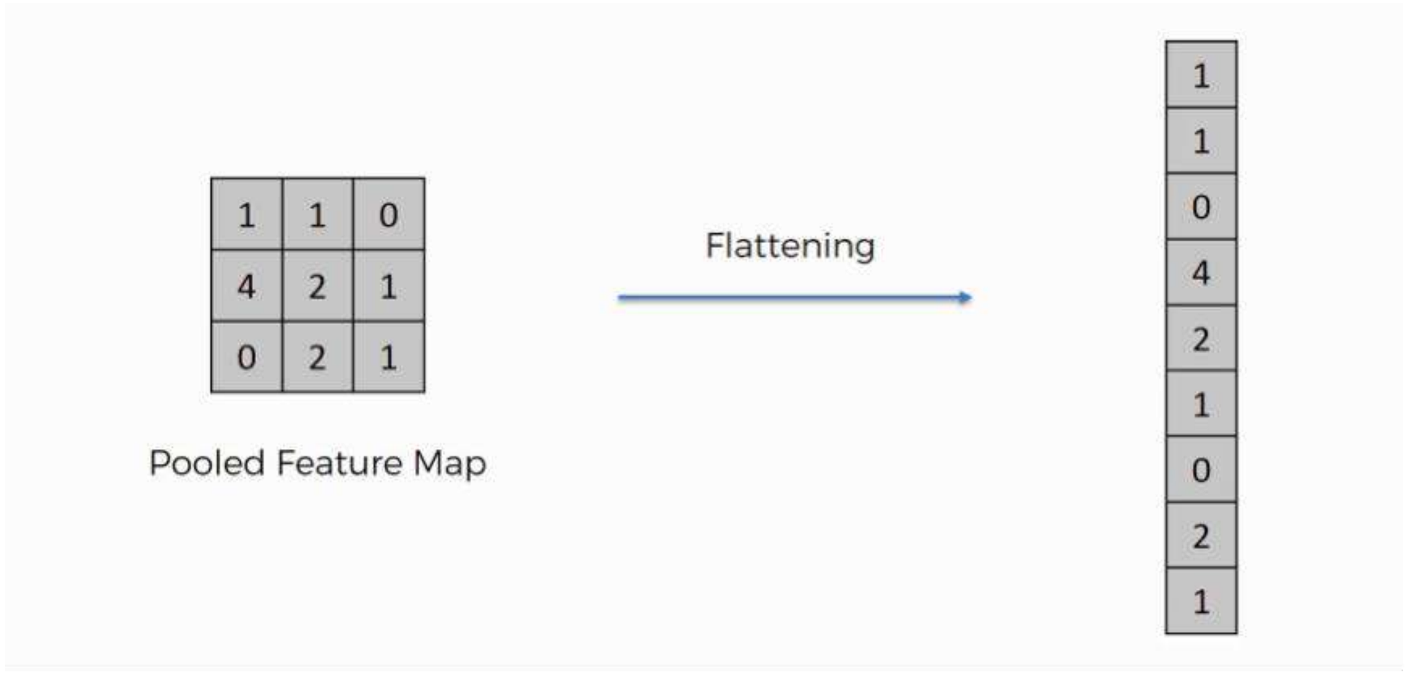
When applying convolution with strides, the filter skips a certain number of pixels in each step, resulting in a downsampled output feature map compared to the input.

Strided convolutions are commonly used to reduce the spatial dimensions of the feature maps, enabling the network to capture larger receptive fields and extract higher-level features.

05/09/2023 :

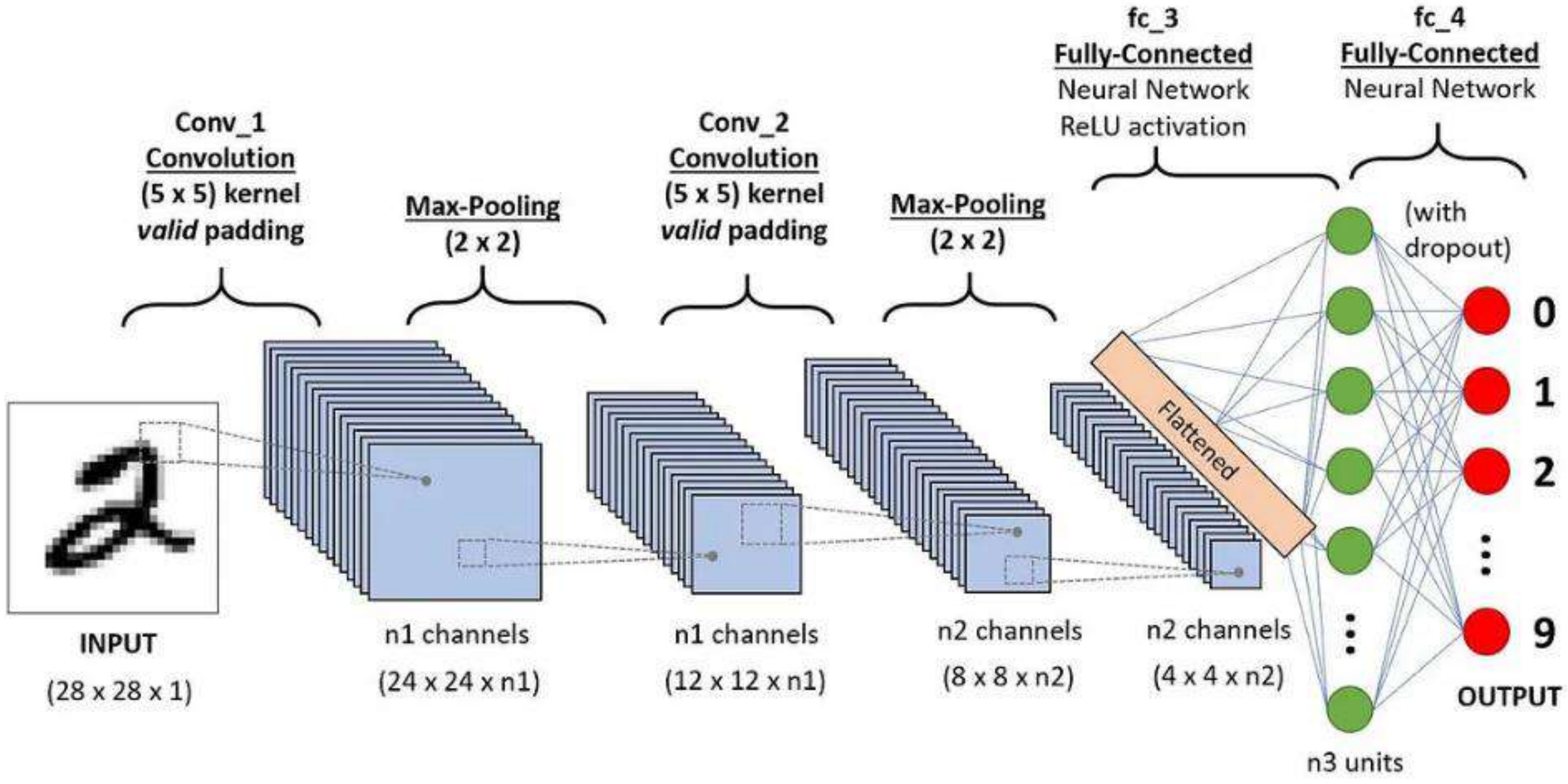
Flattening layer :

Flattening layer is to converts data into 1-dimentional array for feeding next layer. we flatted output of convolutional layer into single long feature vector. which is connected to final classification model, called fully connected layer. let’s suppose we’ve **[5,5,5]** pooled feature map are flattened into **1x125** single vector. So, flatten layers converts multidimensional array to single dimensional vector.



Connec ted Layer :

The model take input image of size 28x28 and applies first Conv layer with kernel 5x5 , stride 1 and padding zero output n1 channels of size 24x24 which is calculated by the output of a pooling layer is (Input Size — Pool Size + 2*Padding)/Stride + 1.. then poling layer same like conv but this time filter size 2x2 and stride 2, when we calculate using Conv layer formula outputs are 12x12 of same channel n1. i repeats similar way once again and at the end flatten layer converts two dimensional arrays to one dimensional vector.



Layers used to build ConvNets :

A complete Convolution Neural Networks architecture is also known as covnets. A covnets is a sequence of layers, and every layer transforms one volume to another through a differentiable function.

Types of layers:

Datasets

Let’s take an example by running a covnets on of image of dimension 32 x 32 x 3.

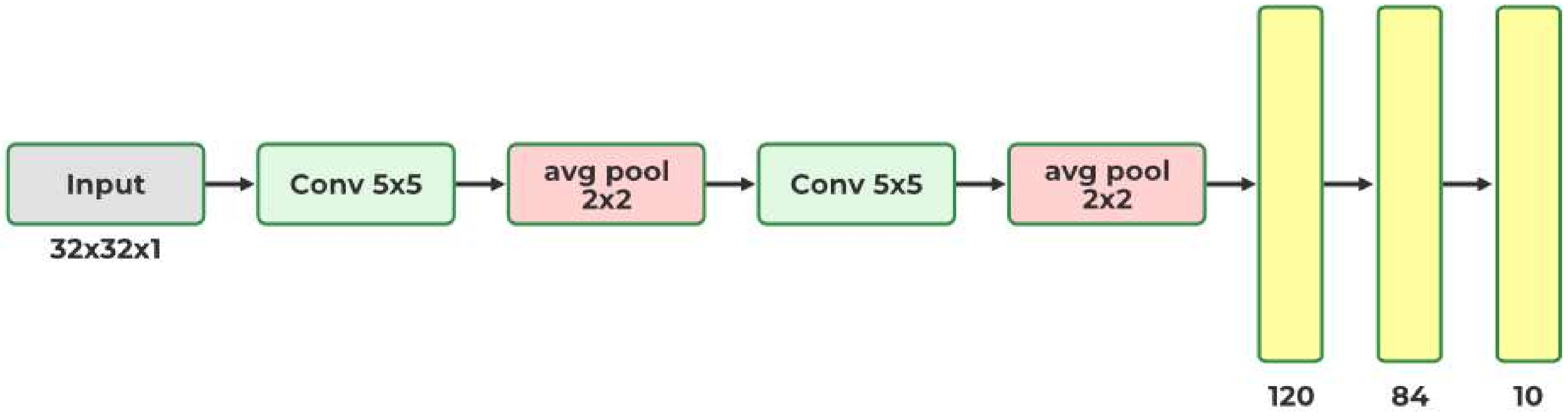
- **Input Layers:** It’s the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3.

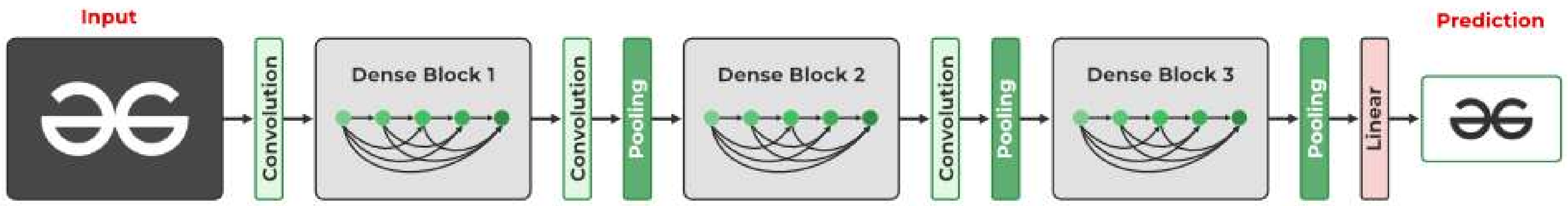
- **Convolutional Layers:** This is the layer, which is used to extract the feature from the input dataset. It applies a set of learnable filters known as the kernels to the input images. The filters/kernels are smaller matrices usually 2×2, 3×3, or 5×5 shape. It slides over the input image data and computes the dot product between kernel weight and the corresponding input image patch. The output of this layer is referred as feature maps. Suppose we use a total of 12 filters for this layer we'll get an output volume of dimension 32 x 32 x 12.

- **Activation Layer:** By adding an activation function to the output of the preceding layer, activation layers add nonlinearity to the network. it will apply an element-wise activation function to the output of the convolution layer. Some common activation functions are **RELU**: max(0, x), **Tanh**, **Leaky RELU**, etc. The volume remains unchanged hence output volume will have dimensions 32 x 32 x 12.

- **Pooling layer:** This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents overfitting. Two common types of pooling layers are **max pooling** and **average pooling**. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12.
- **Flattening:** The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.
- **Fully Connected Layers:** It takes the input from the previous layer and computes the final classification or regression task.
- **Output Layer:** The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

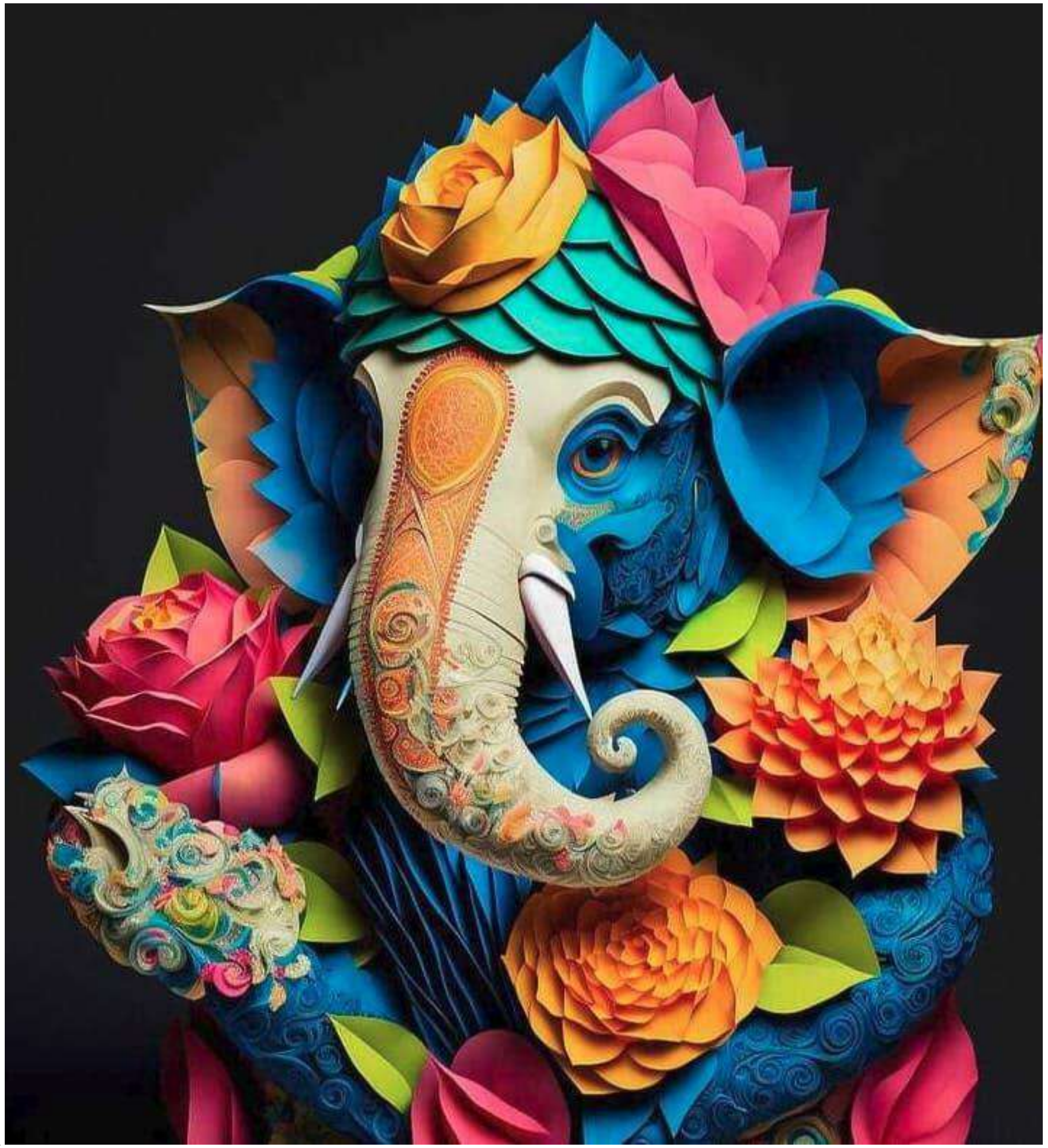
Real time examples :


Input
32x32x1
Conv 5x5
avg pool 2x2
Conv 5x5
avg pool 2x2
120
84
10


Input
Convolution
Dense Block 1
Convolution
Pooling
Dense Block 2
Convolution
Pooling
Dense Block 3
Pooling
Linear
Prediction

Example:

Let’s consider an image and apply the convolution layer, activation layer, and pooling layer operation to extract the inside feature.



Solution :

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from itertools import product

# set the param
plt.rc('figure', autolayout=True)
plt.rc('image', cmap='magma')

# define the kernel
kernel = tf.constant([[-1, -1, -1],
                     [-1,  8, -1],
                     [-1, -1, -1],
                    ])
```

```
# load the image
image = tf.io.read_file('Ganesh .jpg')
image = tf.io.decode_jpeg(image, channels=1)
image = tf.image.resize(image, size=[300, 300])
```

(`tf.image.resize_with_pad()`)

```
# plot the image
img = tf.squeeze(image).numpy()
plt.figure(figsize=(5, 5))
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.title('Original Gray Scale image')
plt.show();
```

```python
# Reformat
image = tf.image.convert_image_dtype(image, 
dtype=tf.float32)
image = tf.expand_dims(image, axis=0)
kernel = tf.reshape(kernel, [*kernel.shape, 1, 1])
kernel = tf.cast(kernel, dtype=tf.float32)

# convolution layer
conv_fn = tf.nn.conv2d

image_filter = conv_fn(
    input=image,
    filters=kernel,
    strides=1, # or (1, 1)
    padding='SAME',
)

plt.figure(figsize=(15, 5))

# Plot the convolved image
plt.subplot(1, 3, 1)

plt.imshow(
    tf.squeeze(image_filter)
)
plt.axis('off')
plt.title('Convolution')

# activation layer
relu_fn = tf.nn.relu
# Image detection
```

```python
image_detect = relu_fn(image_filter)

plt.subplot(1, 3, 2)
plt.imshow(
    # Reformat for plotting
    tf.squeeze(image_detect)
)

plt.axis('off')
plt.title('Activation')

# Pooling layer
pool = tf.nn.pool
image_condense = pool(input=image_detect,
                             window_shape=(2, 2),
                             pooling_type='MAX',
                              strides=(2, 2),
                              padding='SAME',
                             )

plt.subplot(1, 3, 3)
plt.imshow(tf.squeeze(image_condense))
plt.axis('off')
plt.title('Pooling')
plt.show()
```

Advantages of Convolutional Neural Networks (CNNs):

1. Good at detecting patterns and features in images, videos, and audio signals.
2. Robust to translation, rotation, and scaling invariance.
3. End-to-end training, no need for manual feature extraction.
4. Can handle large amounts of data and achieve high accuracy.

Disadvantages of Convolutional Neural Networks (CNNs):

1. Computationally expensive to train and require a lot of memory.
2. Can be prone to overfitting if not enough data or proper regularization is used.
3. Requires large amounts of labeled data.
4. Interpretability is limited, it's hard to understand what the network has learned

Why using Pooling layers , how to select which pooling layer for any particular application ?

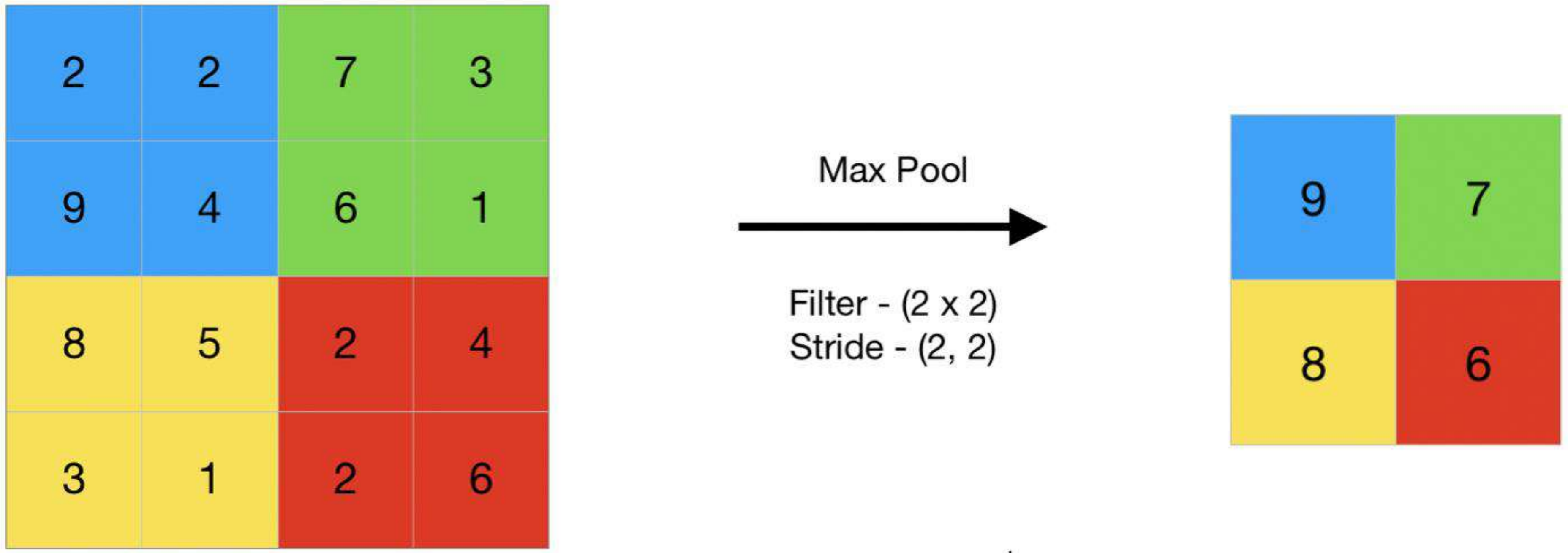
- Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.

- The pooling layer summarises the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarised features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.
-

Types of Pooling Layers:

Max Pooling

1. Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.


2
2
7
3
9
4
6
1
8
5
2
4
3
1
2
6
Max Pool
Filter - (2 x 2)
Stride - (2, 2)
9
7
8
6

```python
import numpy as np
from keras.models import Sequential
from keras.layers import MaxPooling2D

# define input image
image = np.array([[2, 2, 7, 3],
                  [9, 4, 6, 1],
                  [8, 5, 2, 4],
                  [3, 1, 2, 6]])
image = image.reshape(1, 4, 4, 1)

# define model containing just a single max pooling layer
model = Sequential(
    [MaxPooling2D(pool_size = 2, strides = 2)])

# generate pooled output
output = model.predict(image)

# print output image
output = np.squeeze(output)
print(output)
```

1. **Output:**

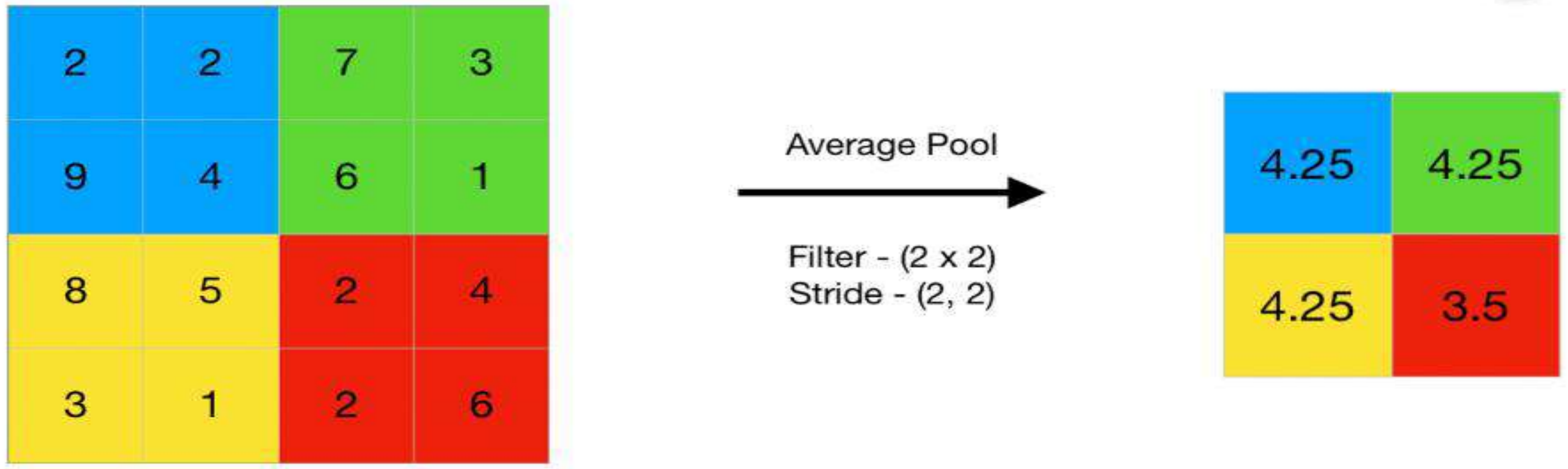
```
[[9. 7.]
[8. 6.]]2
```

1. The MaxPooling2D layer is typically used in CNN architectures to perform downsampling and reduce the spatial dimensions of the input feature maps.

2. In this code, the image array is reshaped using the reshape method. The resulting shape (1, 4, 4, 1) indicates that it is a single sample with a height of 4, a width of 4, and a single channel (grayscale image). This reshaping is often necessary when working with convolutional neural networks (CNNs) in frameworks like Keras.

Average Pooling

1. Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.



1. **Code #2 : Performing Average Pooling using keras**

Python3

```python
import numpy as np
from keras.models import Sequential
from keras.layers import AveragePooling2D

# define input image
image = np.array([[2, 2, 7, 3],
                  [9, 4, 6, 1],
                  [8, 5, 2, 4],
                  [3, 1, 2, 6]])
image = image.reshape(1, 4, 4, 1)

# define model containing just a single average pooling layer
model = Sequential(
    [AveragePooling2D(pool_size = 2, strides = 2)])

# generate pooled output
output = model.predict(image)

# print output image
output = np.squeeze(output)
print(output)
```

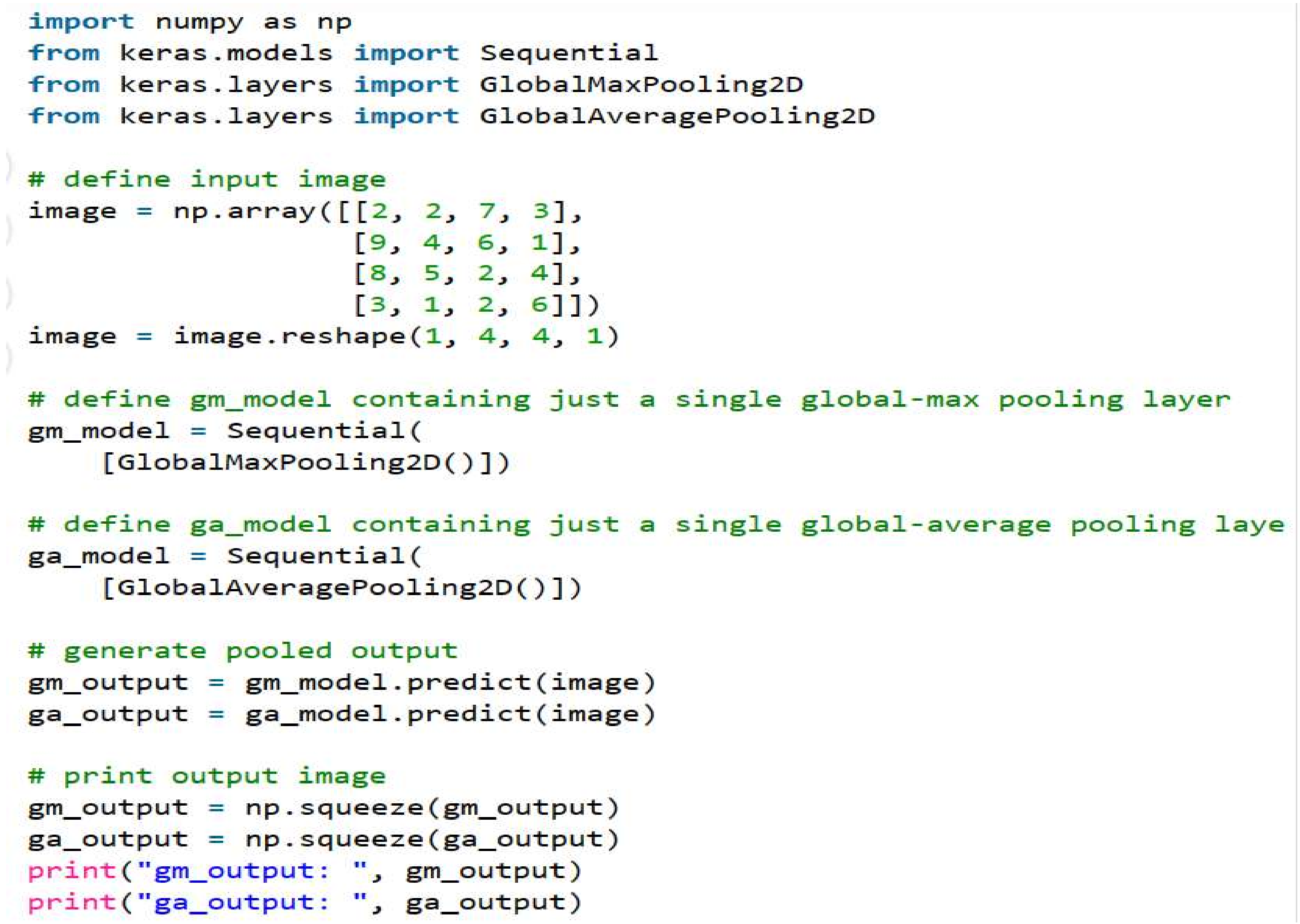
Output :

```
[[4.25 4.25]
[4.25 3.5 ]]
```

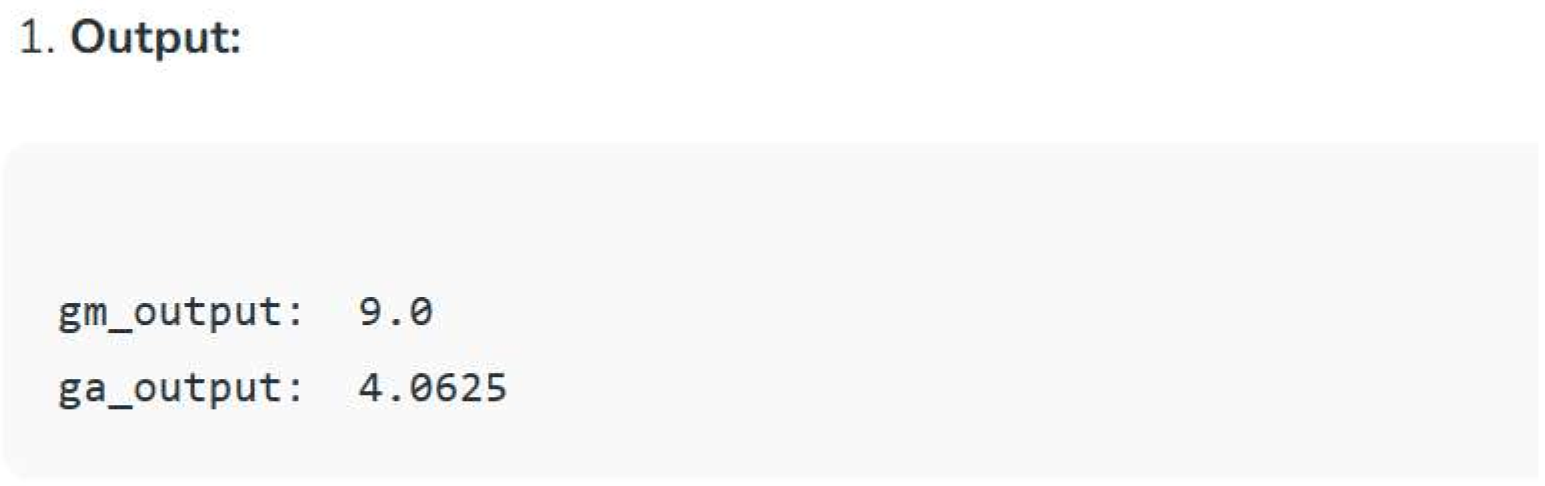
Global Pooling :

1. Global pooling reduces each channel in the feature map to a single value. Thus, an $\mathbf{n_h \times n_w \times n_c}$ feature map is reduced to $\mathbf{1 \times 1 \times n_c}$ feature map. This is equivalent to using a filter of dimensions $\mathbf{n_h \times n_w}$ i.e. the dimensions of the feature map. Further, it can be either global max pooling or global average pooling.

   Code #3 : Performing Global Pooling using keras



```
import numpy as np
from keras.models import Sequential
from keras.layers import GlobalMaxPooling2D
from keras.layers import GlobalAveragePooling2D

# define input image
image = np.array([[2, 2, 7, 3],
                  [9, 4, 6, 1],
                  [8, 5, 2, 4],
                  [3, 1, 2, 6]])
image = image.reshape(1, 4, 4, 1)

# define gm_model containing just a single global-max pooling layer
gm_model = Sequential(
    [GlobalMaxPooling2D()])

# define ga_model containing just a single global-average pooling laye
ga_model = Sequential(
    [GlobalAveragePooling2D()])

# generate pooled output
gm_output = gm_model.predict(image)
ga_output = ga_model.predict(image)

# print output image
gm_output = np.squeeze(gm_output)
ga_output = np.squeeze(ga_output)
print("gm_output: ", gm_output)
print("ga_output: ", ga_output)
```

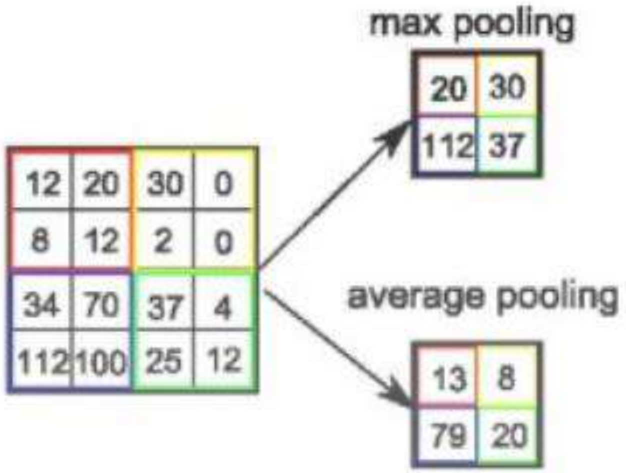


1. **Output:**

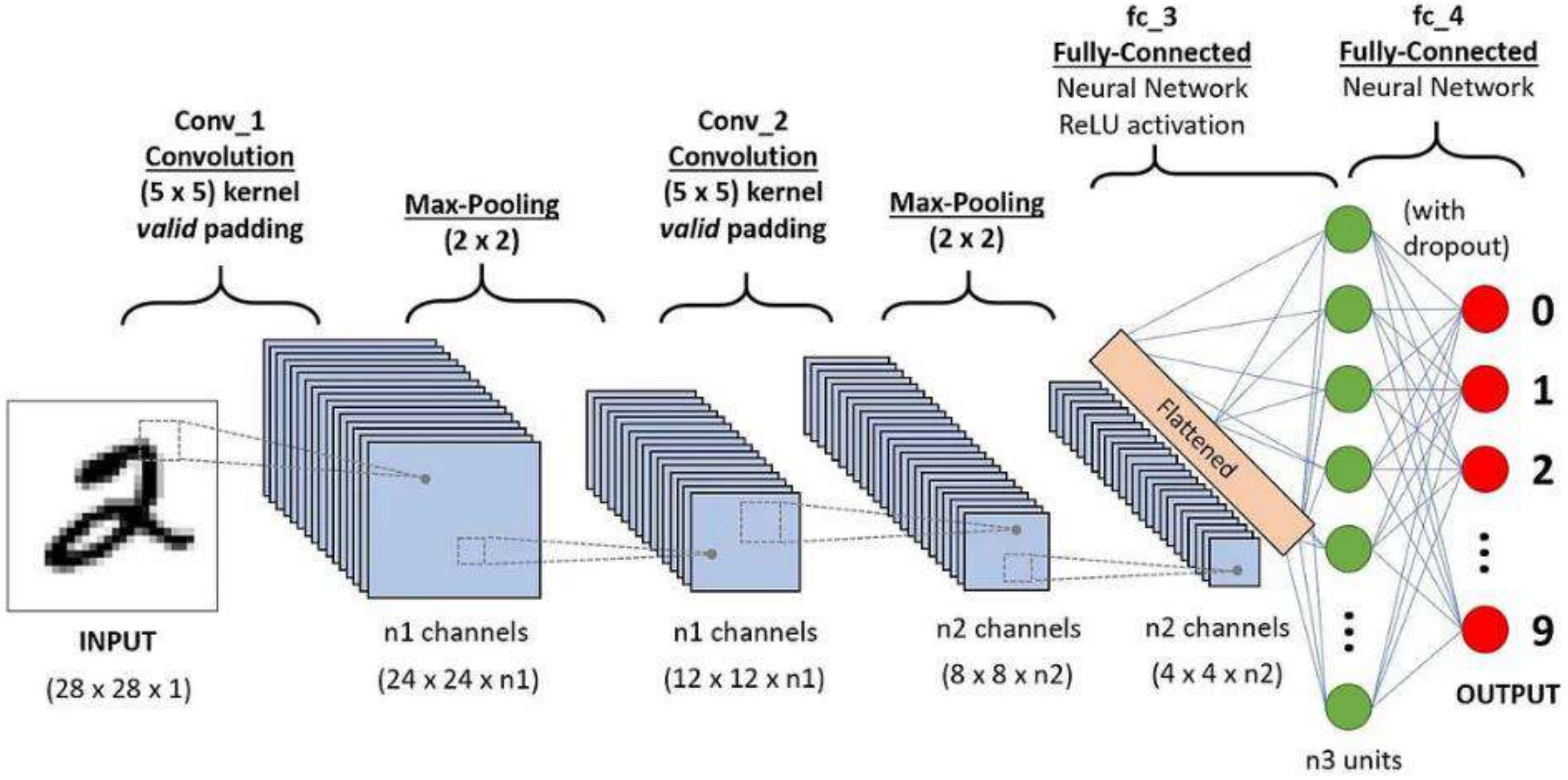
```
gm_output:  9.0
ga_output:  4.0625
```

<u>USES OF AVERGAE POOLING OVER MAX POOLING :</u>

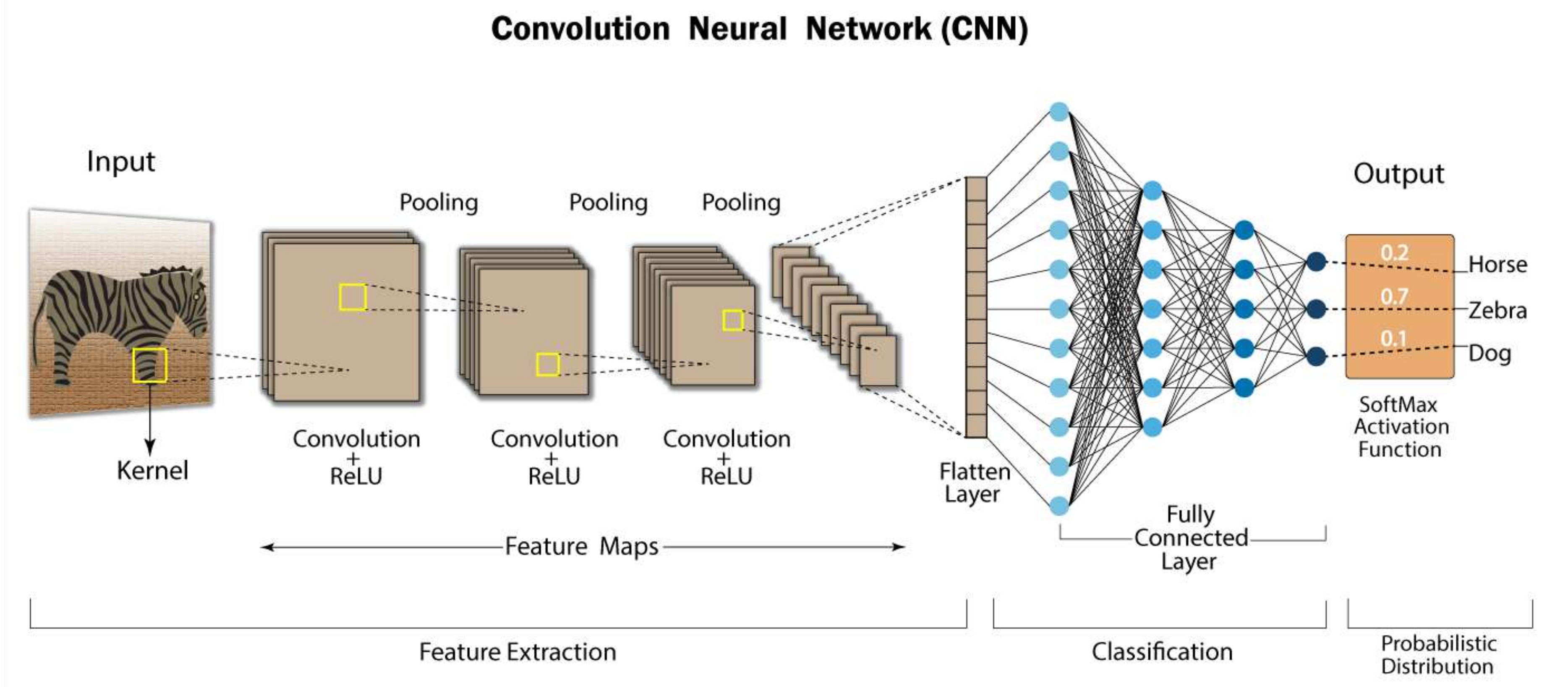
The main objective of average pooling is to just do the object detection and nothing more specific about the extreme features need to be identified.

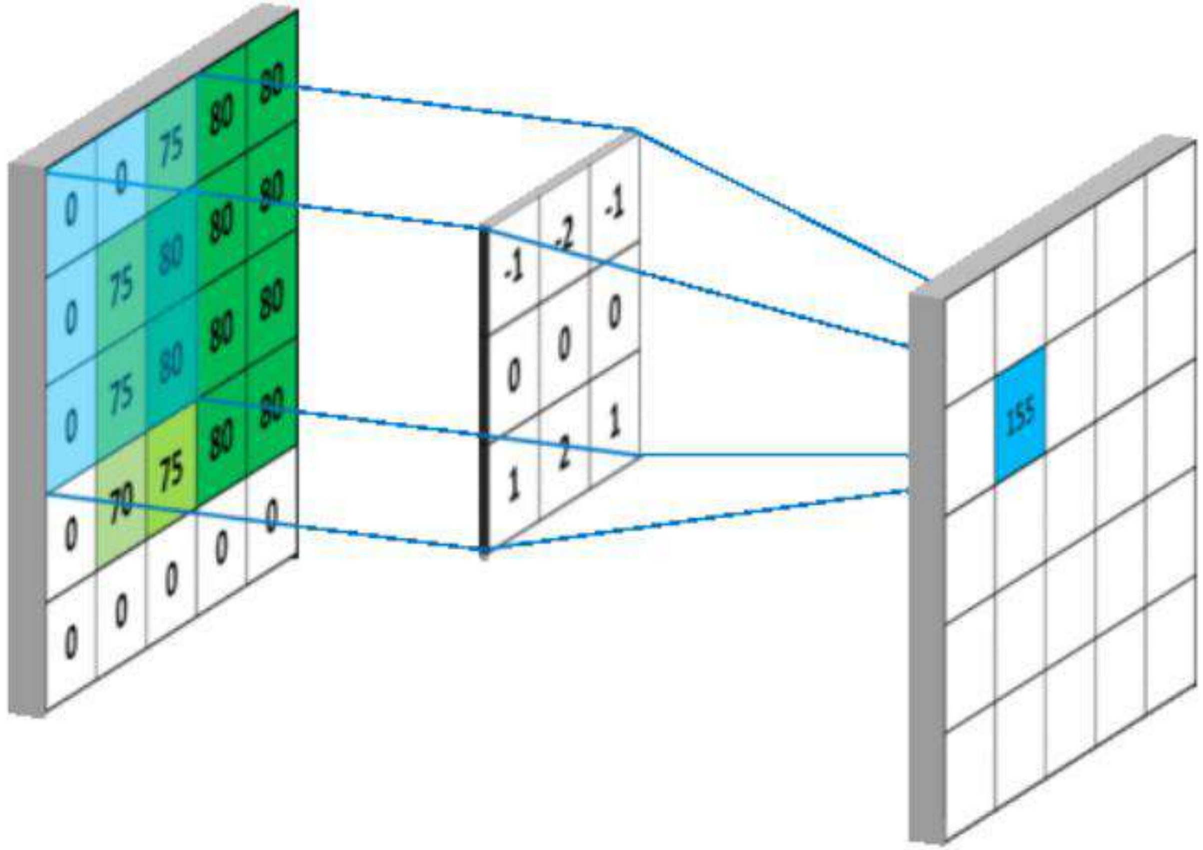


12
20
30
0
8
12
2
0
34
70
37
4
112
100
25
12
max pooling
20
30
112
37
average pooling
13
8
79
20



fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
(with dropout)
Flattened
0
1
2
9
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
OUTPUT


Convolution Neural Network (CNN)
Input
Kernel
Pooling
Pooling
Pooling
Convolution + ReLU
Convolution + ReLU
Convolution + ReLU
Feature Maps
Flatten Layer
Fully Connected Layer
Output
0.2
0.7
0.1
Horse
Zebra
Dog
SoftMax Activation Function
Feature Extraction
Classification
Probabilistic Distribution



0
0
75
80
80
0
75
80
80
80
0
75
80
80
80
0
75
80
80
80
0
70
75
80
0
0
0
0
0
0
-1
-2
-1
0
0
0
1
2
1
155

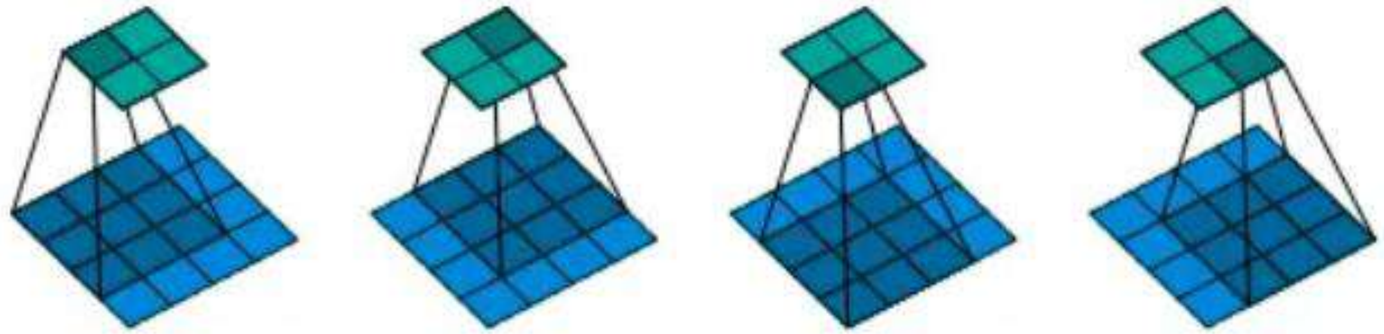


Figure 2.1: (No padding, unit strides) Convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

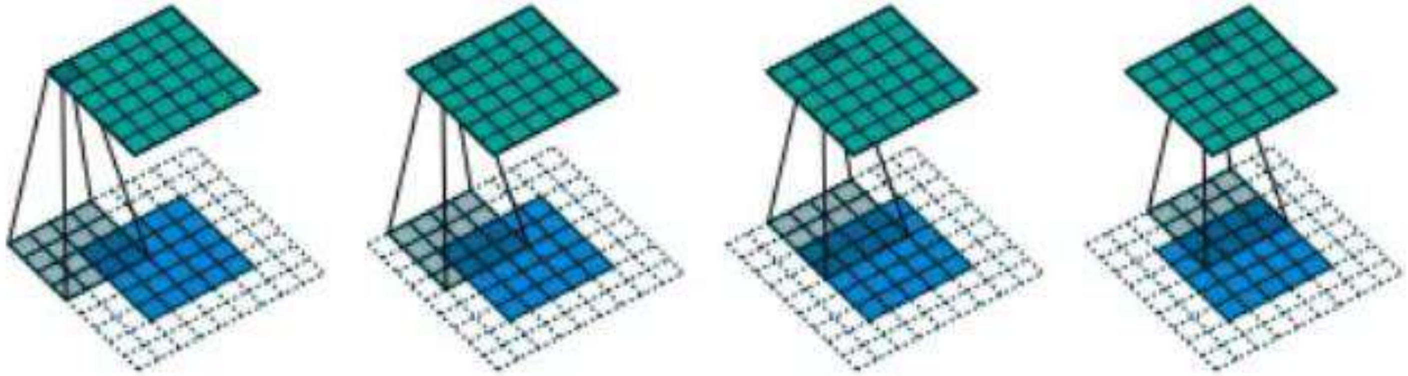


Figure 2.2: (Arbitrary padding, unit strides) Convolving a 4×4 kernel over a 5×5 input padded with a 2×2 border of zeros using unit strides (i.e., $i = 5$, $k = 4$, $s = 1$ and $p = 2$).

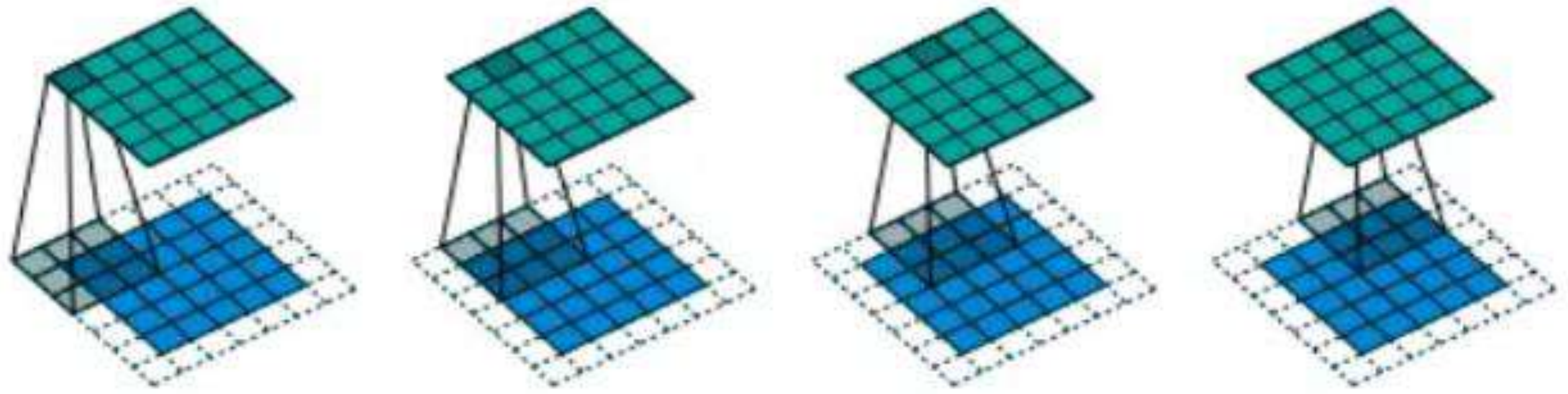


Figure 2.3: (Half padding, unit strides) Convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$).

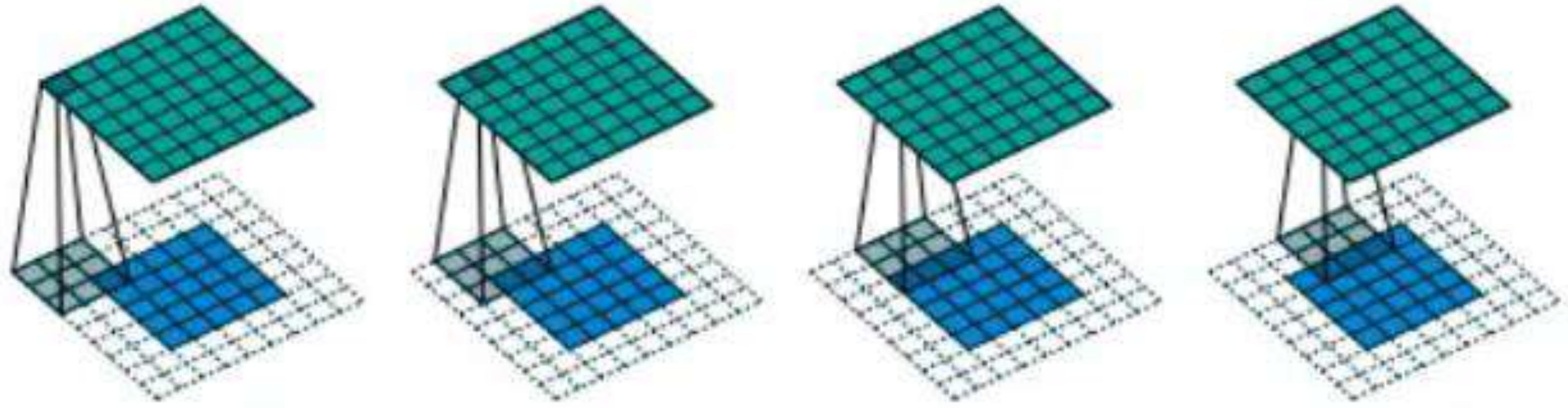


Figure 2.4: (Full padding, unit strides) Convolving a 3×3 kernel over a 5×5 input using full padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 2$).

Output size of image calculated using this formula

$$[(W-K+2P)/S]+1.$$

- W is the input volume
- K is the Kernel size
- P is the padding
- S is the stride

EQUATIONS to remember :

No padding : $(W - K) + 1$

After padding : $(W + 2P - K)+1$

After changed strides

$(W + 2P - K + 1 / S$

Problem 1 :

An input image has been converted into a matrix of size 12 X 12 along with a filter of size 3 X 3 with a Stride of 1. Determine the size of the convoluted matrix.

Problem 2:

What is Stride? What is the effect of high Stride on the feature map?
Stride refers to the number of pixels by which we slide over the filter matrix over the input matrix. For instance –

- If **Stride =1**, then move the filter one pixel at a time.
- If **Stride=2,** then move the filter two-pixel at a time.

Moreover, larger Strides will produce a smaller feature map.

Problem 3 :

List down the hyperparameters of a Pooling Layer.
The hyperparameters for a pooling layer are:

- **Filter size**
- **Stride**
- **Max or average pooling**

Question 1

For which purpose Convolutional Neural Network is used?

It is a multi purpose alghorithm that can be used for Supervised Learning.

Mainly to process and analyse financial models, predicting future trends.

It is a multi purpose alghorithm that can be used for Unsupervised Learning.

Mainly to process and analyse digital images, with some success cases involving processing voice and natural language.

Question 2

What is the biggest advantage utilizing CNN?

- Little dependence on pre processing, decreasing the needs of human effort developing its functionalities.
- It works well both for Supervised and Unsupervised Learning.
- It is easy to understand and fast to implement.
- It has the highest accuracy among all alghoritms that predicts images.

Question 3

Which answer explains better the Convolution?

It is a technique to standardize the dataset.

Understand the model features and selecting the best.

Detect key features in images, respecting their spatial boundaries.

It is the first step to use CNN.

Question 4

Which answer explains better the ReLU?

Helps in the detection of features, increasing the non-linearity of the image, converting positive pixels to zero. This behavior allows you to detect variations of attributes.

A technique that allows you to find outliers.

Helps in the detection of features, decreasing the non-linearity of the image, converting negative pixels to zero. This behavior allows you to detect variations of attributes.

It is used to find the best features considering their correlation.

Question 5

Which answer explains better the Pooling?

Creates a pool of data in order to improve the accuracy of the alghorithm predicting images.

It assists in the detection of distorted features, in order to find dominant attributes.

It assists in the detection of features, even if they are distorted, in addition to decreasing the attribute sizes, resulting in decreased computational need. It is also very useful for extracting dominant attributes.

Decrease the features size, in order to decrease the computional power that are needed.

Question 6

Which answer explains better the Flattening?

Transform images to vectors to make it easier to predict.

Delete unnecessary features to make our dataset cleaner.

It is the last step of CNN.

Once we have the pooled feature map, this component transforms the information into a vector. It's the input we need to get on with Artificial Neural Networks.

Question 7

Which answer explains better the Full Connection?

Full Connection acts by placing different weights in each synapse in order to minimize errors. This step can be repeated until an expected result is achieved.

It is a componente that connects diferents alghorithms in order to increase the accuracy.

Full Connection acts by placing different weights in each synapse in order to minimize errors. No iteration is needed, since we can get the best results in our first attempt.

It is the last step of CNN, where we connect the results of the earlier componentes to create a output.

CNN is divided in two big steps. Feature Learning and Classification. What happens in each step?

Feature Learning has Convolution, ReLU and Pooling components, with inumerous iterations between them before move to Classification, which uses the Flattening and Full Connection components.

Feature Learning has Flattening and Full Connection components, with inumerous iterations between them before move to Classification, which uses the Convolution, ReLU and Pooling componentes.

option4

During Feature Learning, CNN uses appropriates alghorithms to it, while during classification its changes the alghorithm in order to achive the expected result.

What is the difference between CNN and ANN?

CNN has one or more layers of convolution units,

which receives its input from multiple units.

CNN uses a more simpler alghorithm than ANN.

CNN is a easiest way to use Neural Networks.

They complete eachother, so in order to use ANN,

you need to start with CNN.

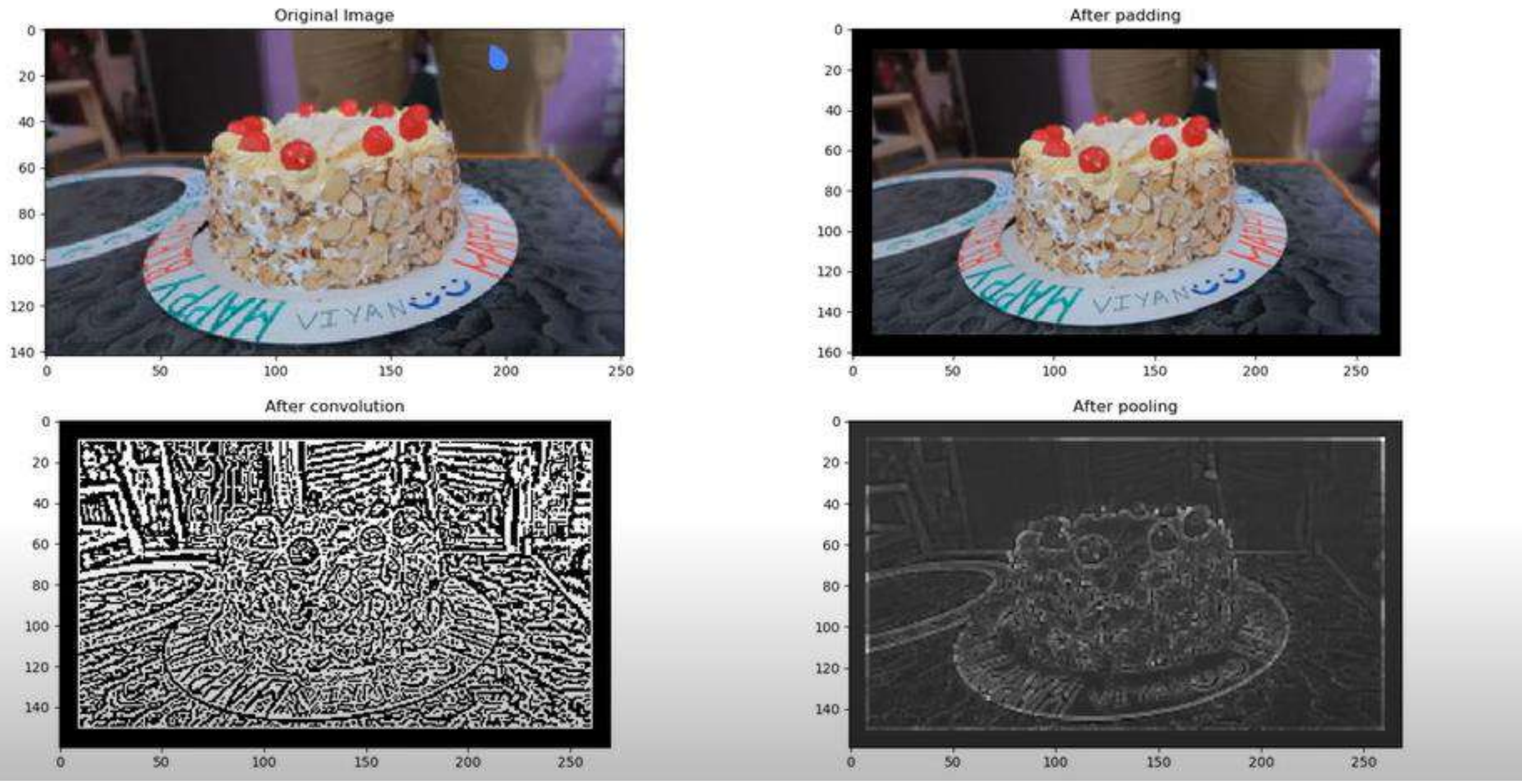
What is the benefit to use CNN instead ANN?

Reduce the number of units in the network, which means fewer parameters to learn and reduced chance of overfitting. Also they consider the context information in the small neighborhoos. This feature is very important to achieve a better prediction in data like images.

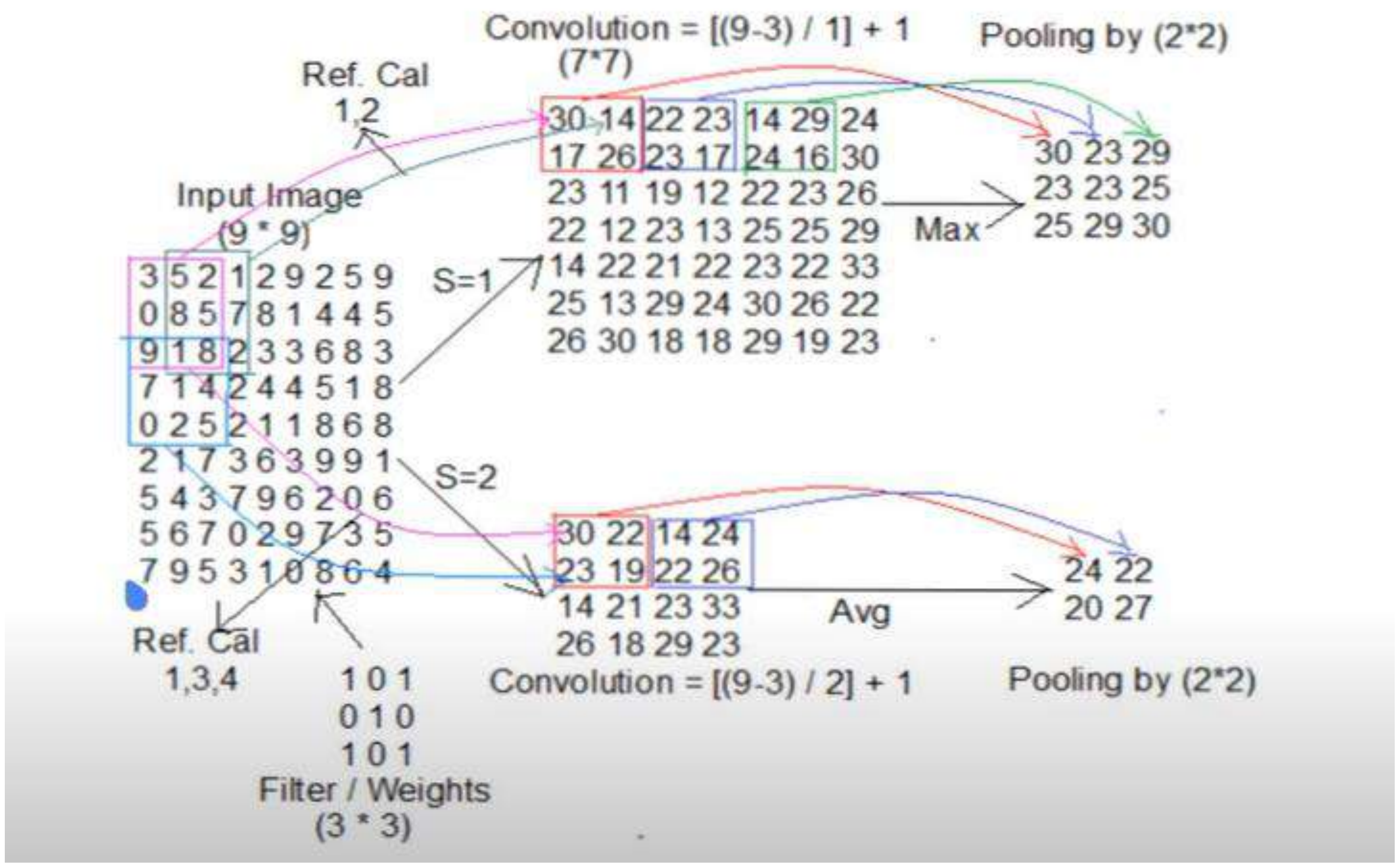
Increase the number of units in the network, which means more parameters to learn and increase chance of overfitting. Also they consider the context information in the small neighborhoos. This feature is very important to achieve a better prediction.

CNN has better results since you have more computional power.

There is no benefit, ANN is always better.



Original Image
After padding
After convolution
After pooling
HAPPY BIRTHDAY
VIYAN


Convolution = [(9-3) / 1] + 1
(7*7)
Pooling by (2*2)
Ref. Cal
1,2
Input Image
(9 * 9)
3 5 2 1 2 9 2 5 9
0 8 5 7 8 1 4 4 5
9 1 8 2 3 3 6 8 3
7 1 4 2 4 4 5 1 8
0 2 5 2 1 1 8 6 8
2 1 7 3 6 3 9 9 1
5 4 3 7 9 6 2 0 6
5 6 7 0 2 9 7 3 5
7 9 5 3 1 0 8 6 4
S=1
30 14 22 23 14 29 24
17 26 23 17 24 16 30
23 11 19 12 22 23 26
22 12 23 13 25 25 29
14 22 21 22 23 22 33
25 13 29 24 30 26 22
26 30 18 18 29 19 23
Max
30 23 29
23 23 25
25 29 30
S=2
30 22 14 24
23 19 22 26
14 21 23 33
26 18 29 23
Avg
24 22
20 27
Ref. Cal
1,3,4
1 0 1
0 1 0
1 0 1
Filter / Weights
(3 * 3)
Convolution = [(9-3) / 2] + 1
Pooling by (2*2)

11/9/2023 :

Building powerful image classification models using very little data (keras.io) - lab exercise

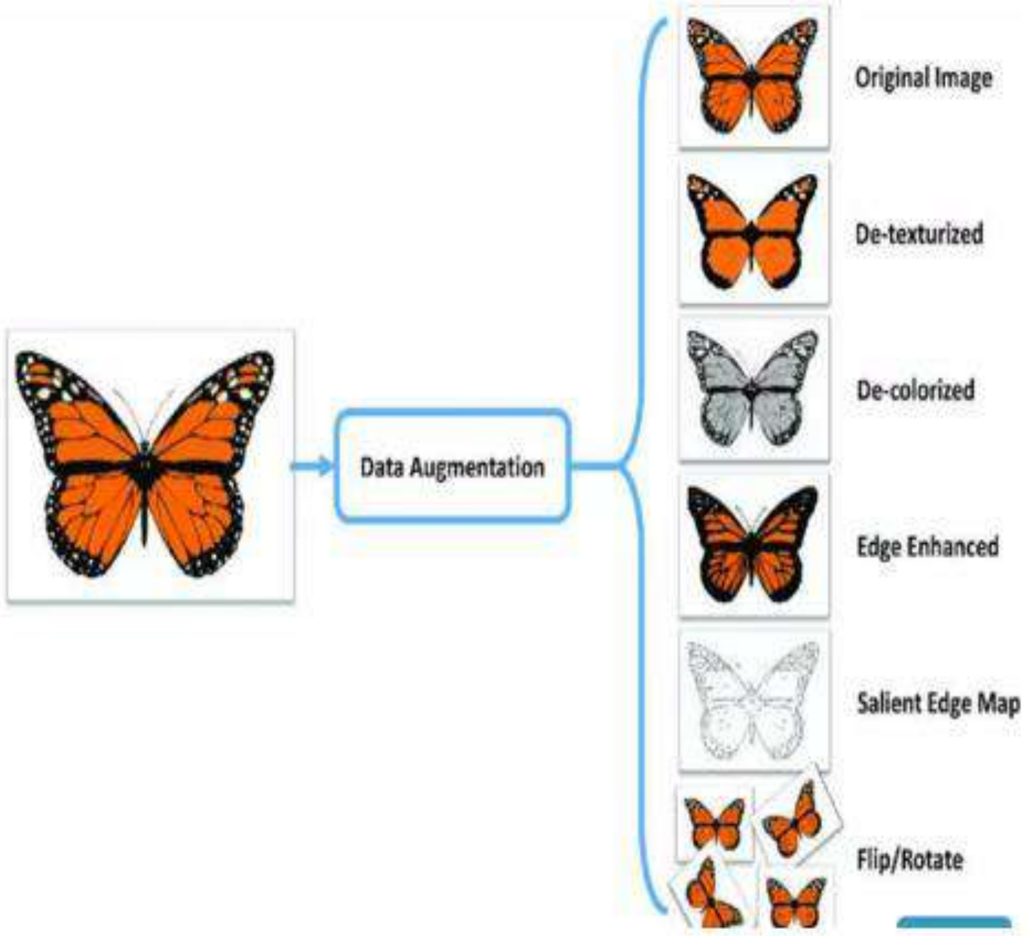
Train Neural Network by loading your images |TensorFlow, CNN, Keras tutorial - YouTube – lab exercise

CS 230 - Convolutional Neural Networks Cheatsheet (stanford.edu) – tutorials for classes

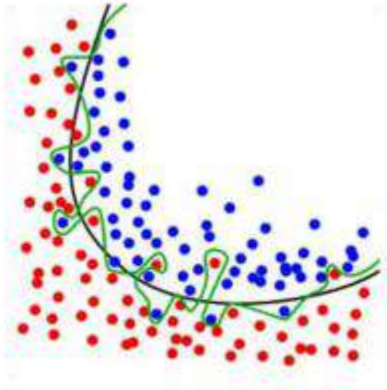
Convolutional Neural Networks | CNN | Kernel | Stride | Padding | Pooling | Flatten | Formula - YouTube – Assignment 4

What is Data Augmentation?

- Expand the size of a training set by creating modified data from the existing one.



Purpose of Data Augmentation



Enlarge
dataset

Prevent
overfitting

Improve
performance model

What to Augment!

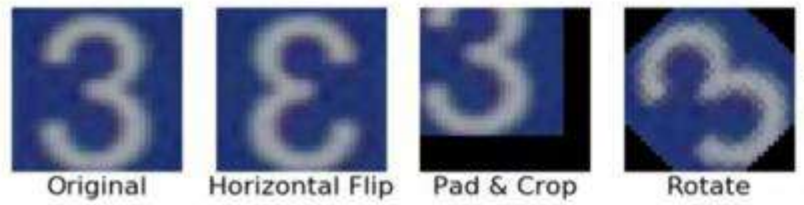
Audio

Texts

Images

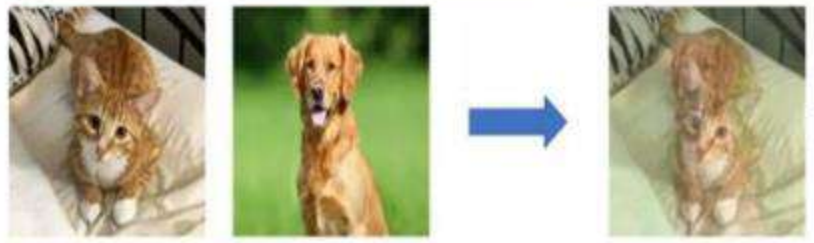
Any other types

Data Augmentation techniques : For Images



Geometric transformations

Color space transformations



Mixing images

Data Augmentation techniques : For Text

In my presentation focus topic is Data Augmentation.

In my presentation **theme** topic is Data Augmentation – Synonym Replacement

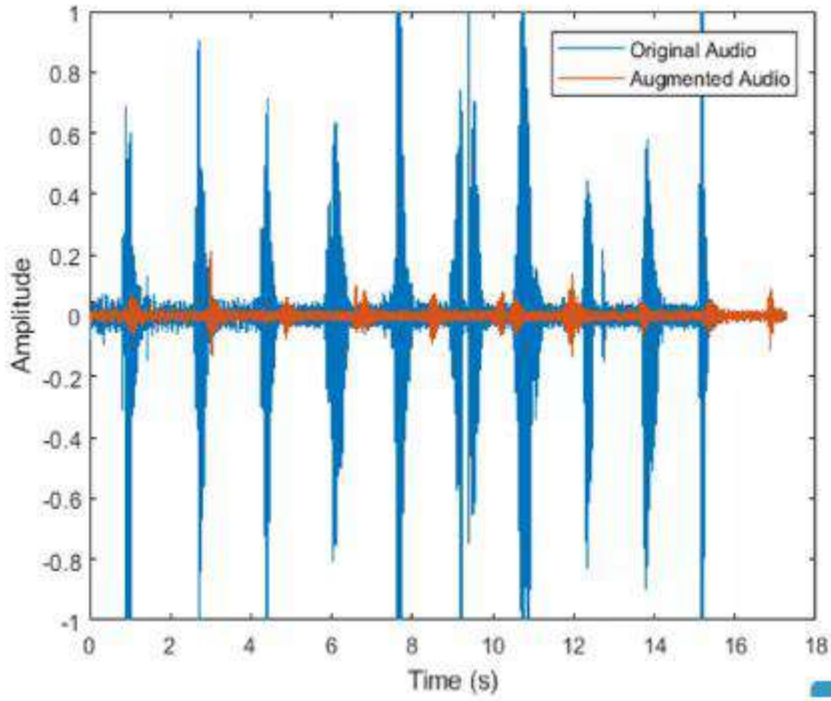
In my presentation focus topic is Data Augmentation **techniques** -- Random Insertion

In my presentation **topic focus** is Data Augmentation -- Random Swap

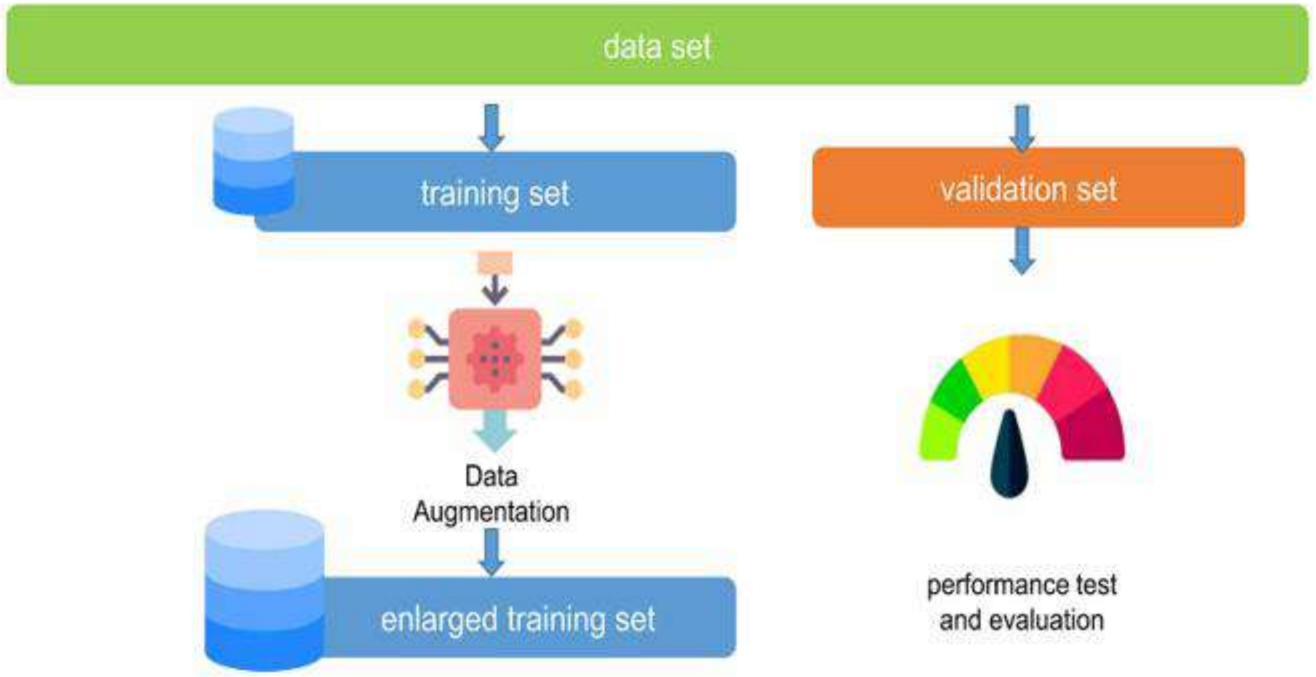
In my presentation focus topic Data Augmentation --Random Deletion

Data Augmentation techniques : For Audio

- Noise Injection
- Shifting
- Changing the speed of the Tape



Workflow of Data Augmentation



When do I use data augmentation?

small data set

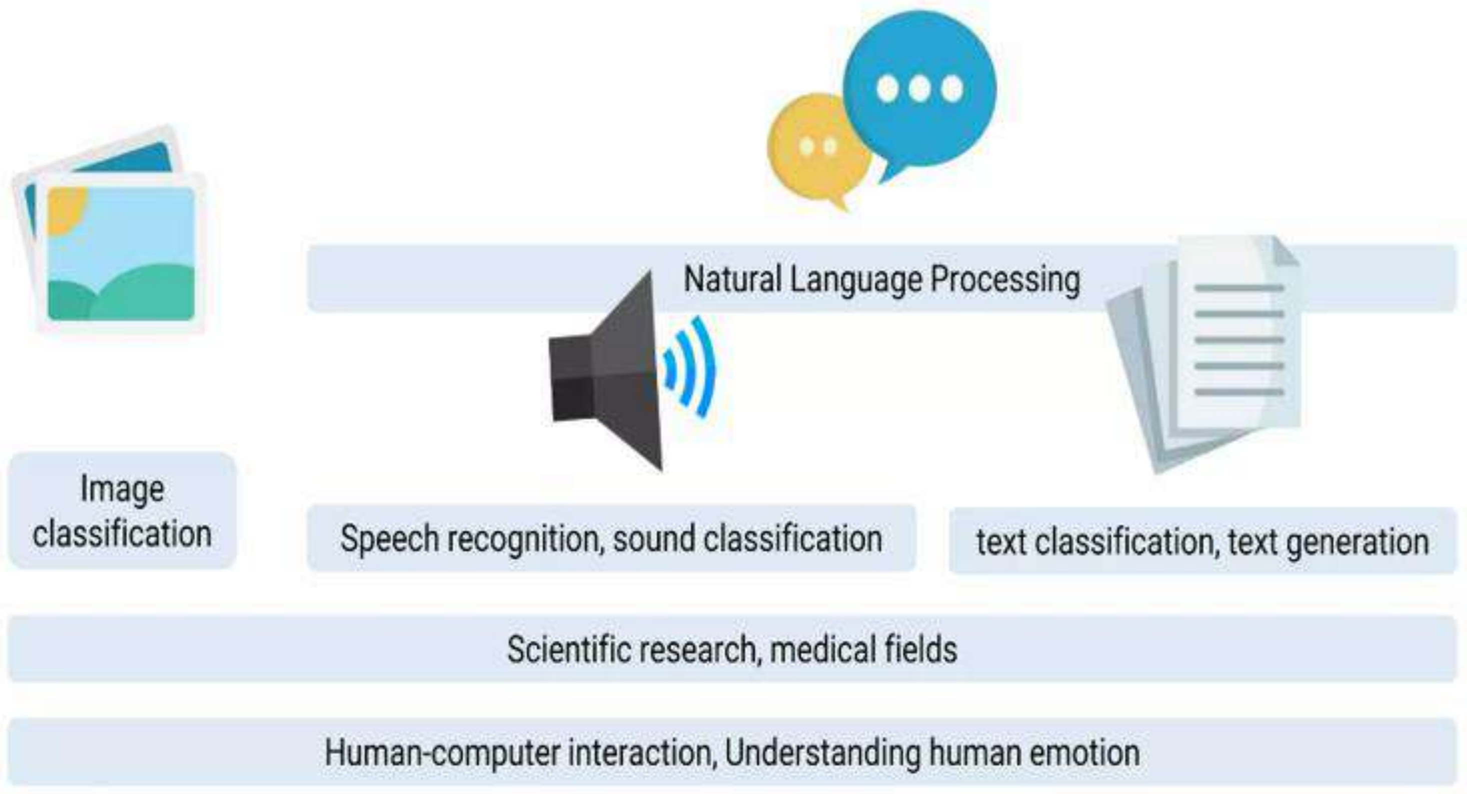
complex problem

transformation of data would be effective and easy

You know how to find the correct method for your data

You have a strategy for dealing with overfitting

Data Augmentation Applications


Natural Language Processing
Image classification
Speech recognition, sound classification
text classification, text generation
Scientific research, medical fields
Human-computer interaction, Understanding human emotion

Natural Language Processing: some tasks

Translation tasks

Speech emotion recognition

Speech recognition

Text-to-speech

demo

Text classification

syntax and semtantic analysis

Text generation, dialogue management

NLP and Data augmentation in speech emotion recognition

C. Etienne and B. Schmauch, **"Speech Emotion Recognition with Data Augmentation and Layer-wise Learning Rate Adjustment"**

Problem:

class imbalance and small dataset

Solution:

Data augmentation by rescaling of spectograms

Results:

Improvement in accuracy of predictions

	Baseline			Best model
Augmentation during training	-	-	+	+
Oversampling (×2) of happiness and anger	-	+	+	+
Frequency range (kHz)	4	4	4	8
Weighted accuracy	66.4	63.5	64.2	64.5
Unweighted accuracy	57.7	59.8	60.9	61.7

NLP and Data augmentation in translation tasks

T. Potapczyk, P. Przybysz, M. Chochowski, A. Szumaczuk: “**Samsung’s System for the IWSLT 2019 End-to-End Speech Translation Task**“

Problem:

Translate English to German TED lectures

Solution:

Data augmentation by applying audio effects

Results:

Improvement in BLEU score

Model	tst2010
ASR Transformer for SLT baseline	12.76
+ Model averaging	12.99
+ Dense FF in Decoder	13.20
+ Spectogram masking	13.74
+ Data augmentation (speed x1)	14.85
+ 8 head attention	15.31
+ Data augmentation (speed + echo x1)	15.56

BLEU scores for models trained on LibriSpeech data. Each subsequent model includes all the previous techniques in the table.

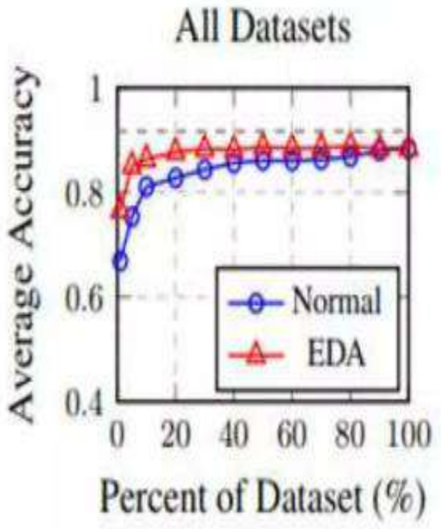
NLP and Data augmentation in classification tasks

J. Wei and K. Zou, **"EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks"**

Problem: Performing text classification depends on quality and quantity of data

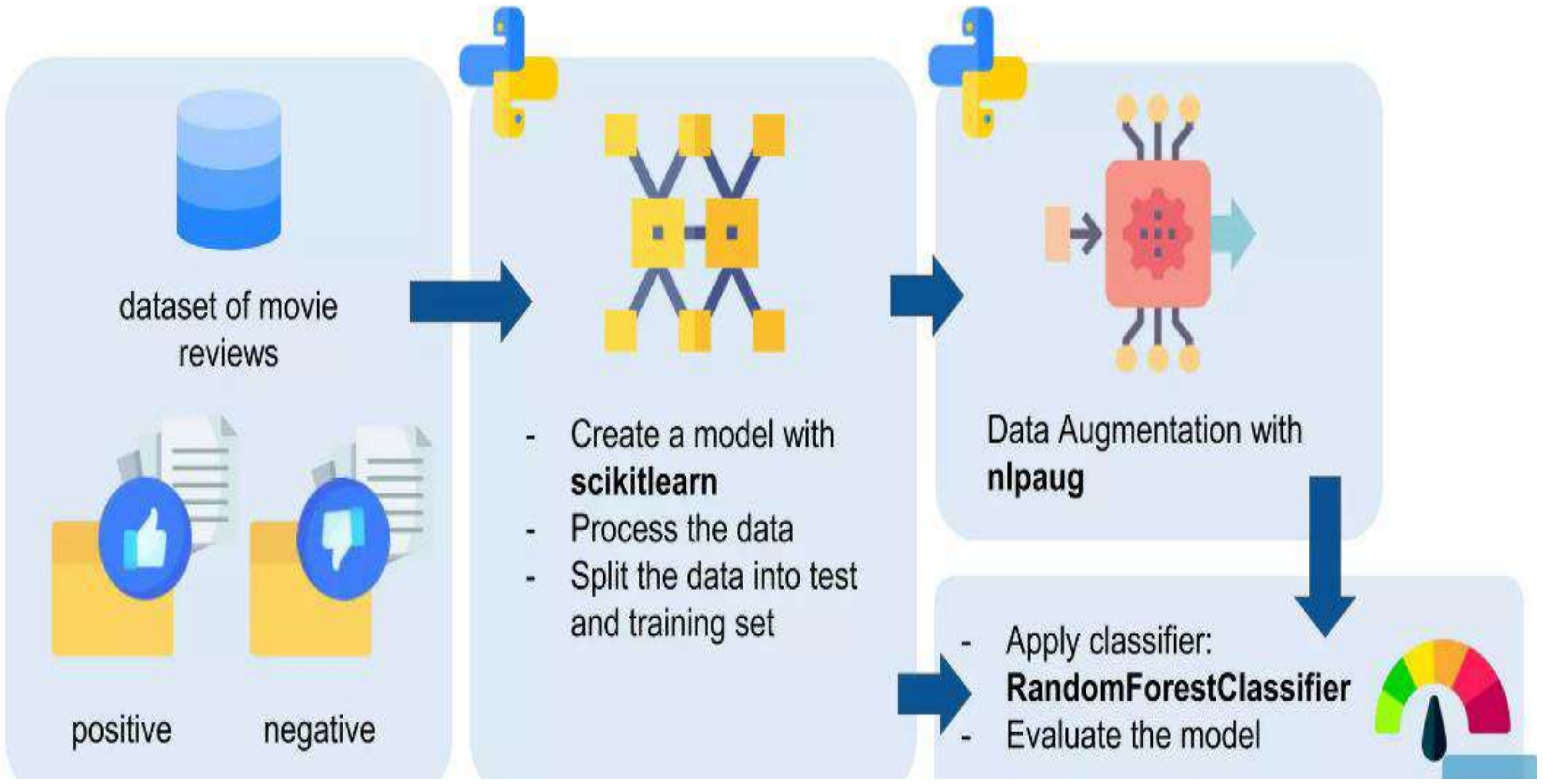
Solution: Data augmentation by application of multiple transformations on text

Results: Performance gain of model if right amount of data augmentation chosen



Performance on benchmark text classification tasks with and without EDA, for various dataset sizes used for training. [1]

https://github.com/mimmimkr/nlp_dataaug nd Data Augmentation

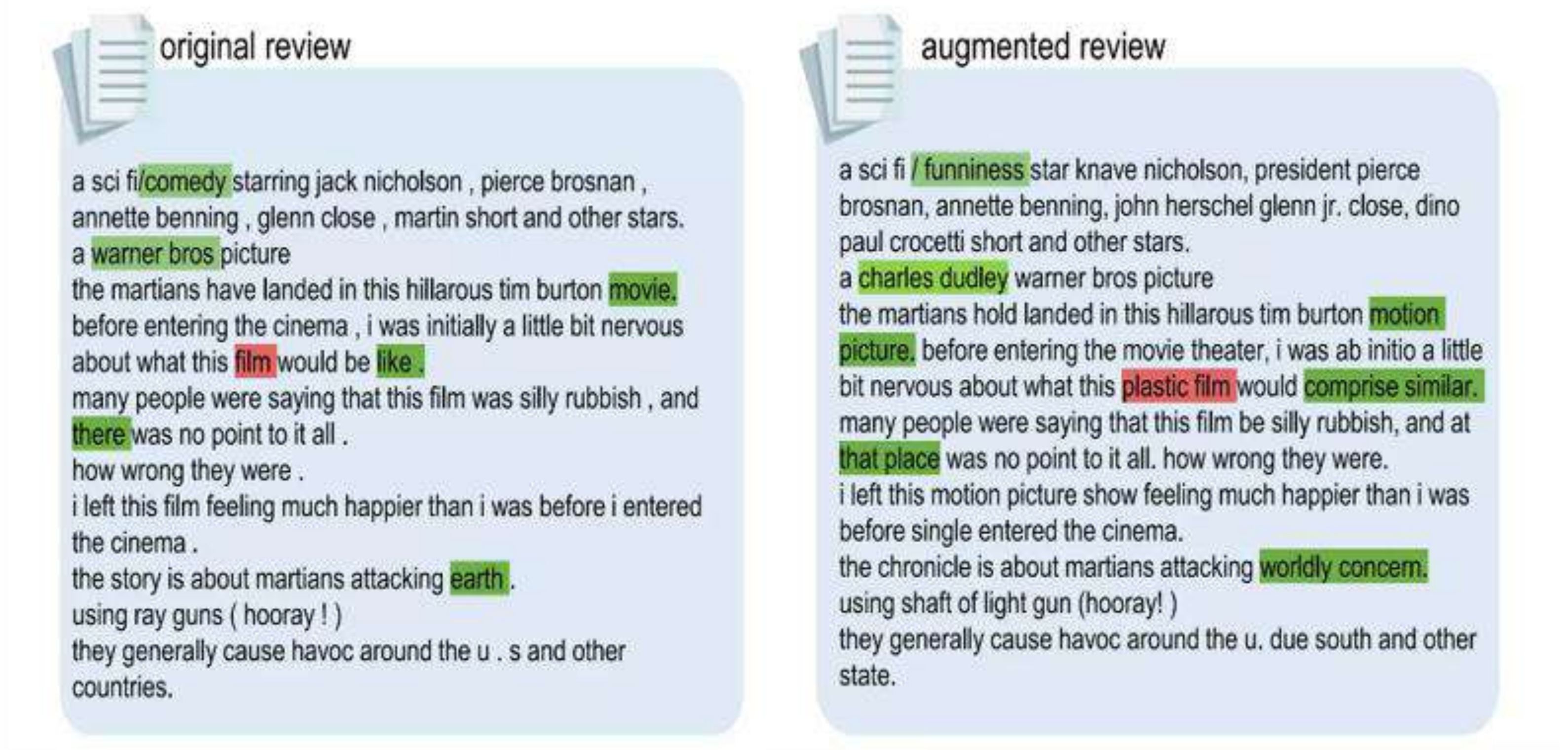


- Create a model with **scikitlearn**
- Process the data
- Split the data into test and training set

Data Augmentation with **nlpaug**

- Apply classifier: **RandomForestClassifier**
- Evaluate the model

Demo: Result of Augmentation with synonym replacement


original review

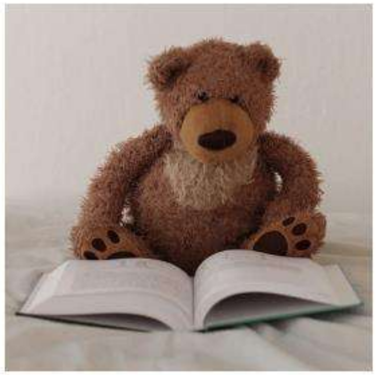
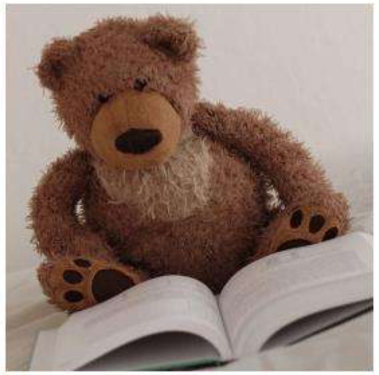
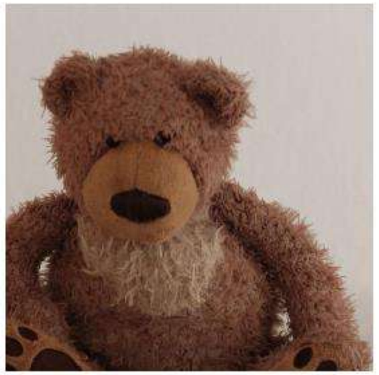
a sci fi/comedy starring jack nicholson , pierce brosnan , annette benning , glenn close , martin short and other stars.
a warner bros picture
the martians have landed in this hillarous tim burton movie.
before entering the cinema , i was initially a little bit nervous about what this film would be like .
many people were saying that this film was silly rubbish , and there was no point to it all .
how wrong they were .
i left this film feeling much happier than i was before i entered the cinema .
the story is about martians attacking earth .
using ray guns (hooray !)
they generally cause havoc around the u . s and other countries.

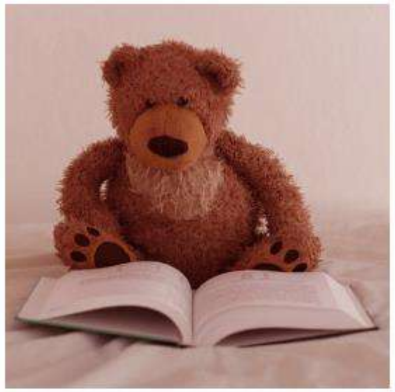
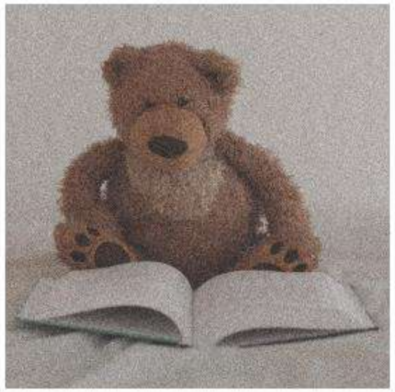
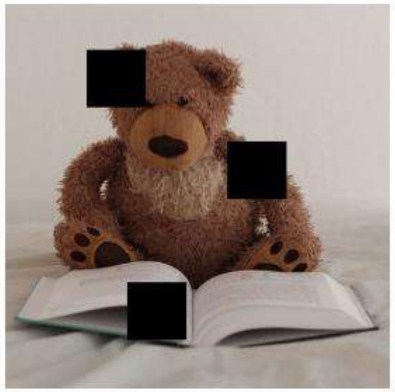
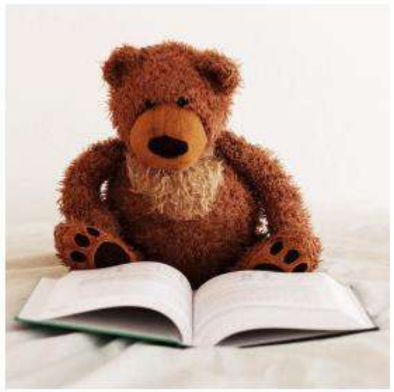
augmented review

a sci fi / funniness star knave nicholson, president pierce brosnan, annette benning, john herschel glenn jr. close, dino paul crocetti short and other stars.
a charles dudley warner bros picture
the martians hold landed in this hillarous tim burton motion picture. before entering the movie theater, i was ab initio a little bit nervous about what this plastic film would comprise similar.
many people were saying that this film be silly rubbish, and at that place was no point to it all. how wrong they were.
i left this motion picture show feeling much happier than i was before single entered the cinema.
the chronicle is about martians attacking worldly concern.
using shaft of light gun (hooray!)
they generally cause havoc around the u. due south and other state.

Data processing

❒ **Data augmentation** — Deep learning models usually need a lot of data to be properly trained. It is often useful to get more data from the existing ones using data augmentation techniques. The main ones are summed up in the table below. More precisely, given the following input image, here are the techniques that we can apply:

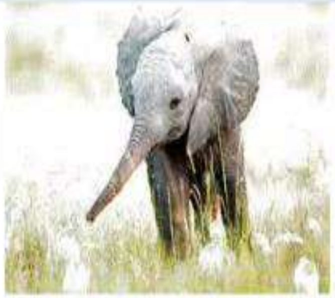
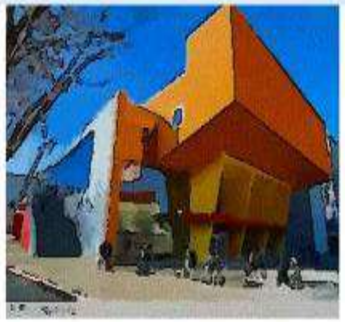
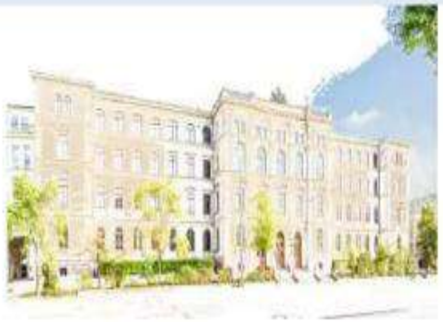
Original	Flip	Rotation	Random crop
			
• Image without any modification	• Flipped with respect to an axis for which the meaning of the image is preserved	• Rotation with a slight angle • Simulates incorrect horizon calibration	• Random focus on one part of the image • Several random crops can be done in a row

Color shift	Noise addition	Information loss	Contrast change
			
• Nuances of RGB is slightly changed • Captures noise that can occur with light exposure	• Addition of noise • More tolerance to quality variation of inputs	• Parts of image ignored • Mimics potential loss of parts of image	• Luminosity changes • Controls difference in exposition due to time of day

Remark: data is usually augmented on the fly during training.

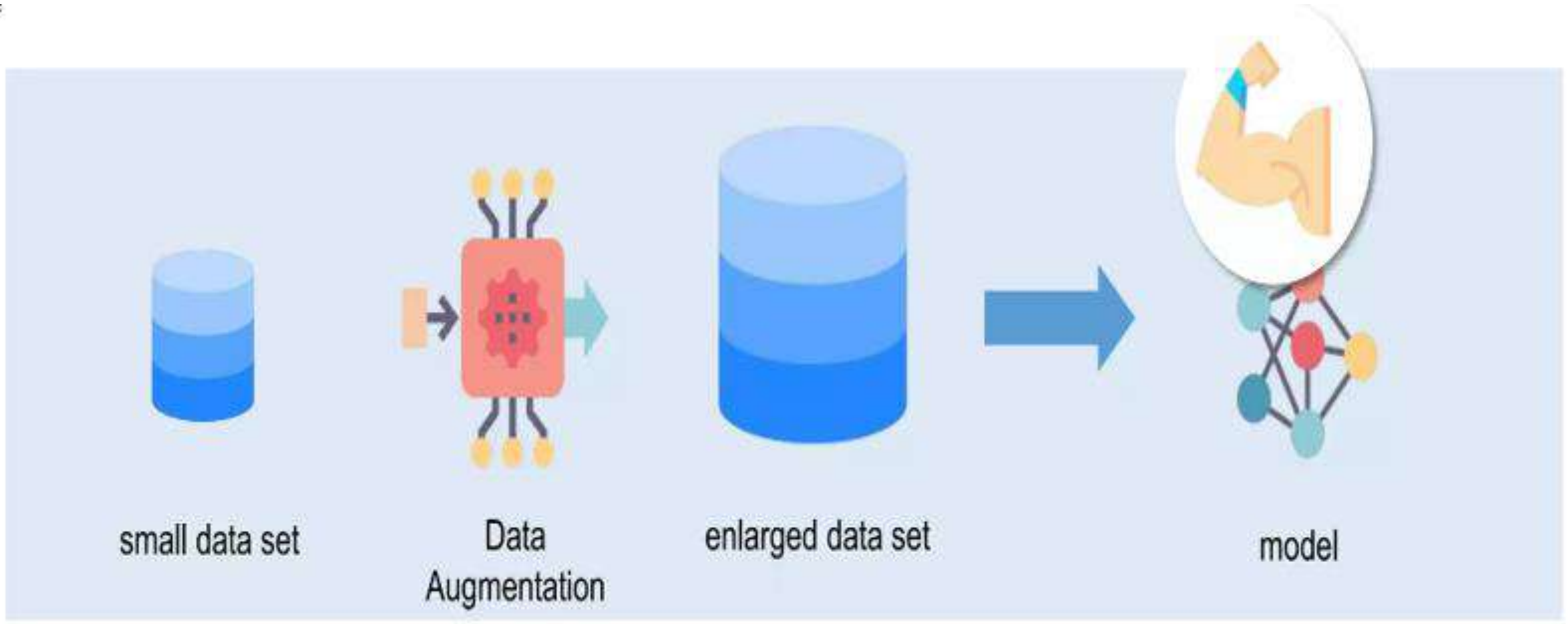
Demo: BONUS - image augmentation with imgaug

Augmentation by: cropping, scaling, artistic filters, weather, blur, rotation, flip...

)ata augmentation: pros and cons

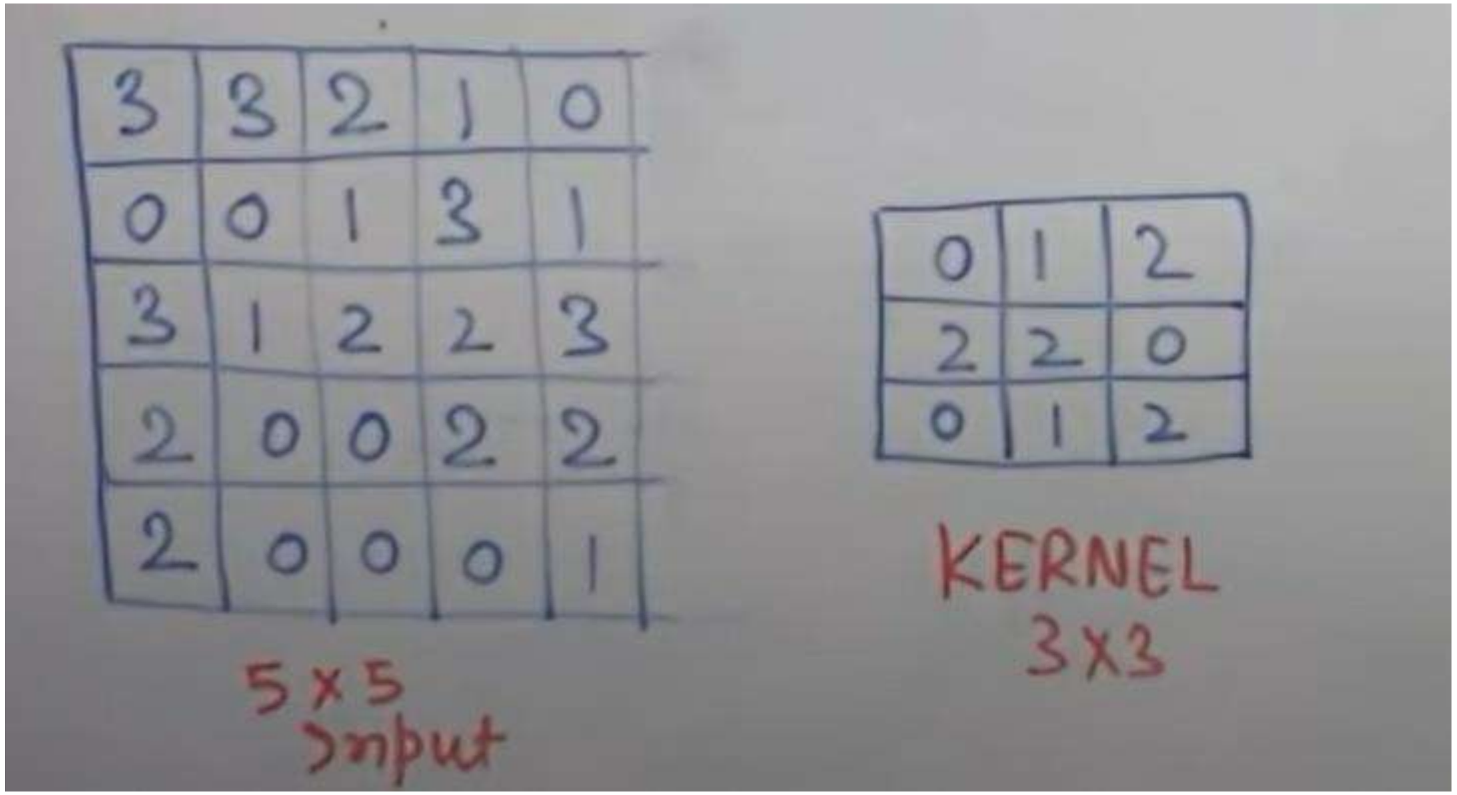
+	-
easy implementation, low effort	time consuming (sometimes)
tailored approach, no model changes	trial and error
performance boost: easy to measure	relational gaps between data and augmented data
can help with overfitting	can lead to overfitting if not done right


small data set
Data Augmentation
enlarged data set
model

1. Consider the below 6x6 matrix and perform the operations of CNN followed by taking 3x3 scalar filter with stride 1. If necessary perform padding with value 1 and also followed by max pooling of 2x2 matrix

$$\begin{bmatrix} 3 & 1 & 6 & 4 & 2 & 0 \\ 7 & 6 & 3 & 1 & 2 & 4 \\ 1 & 5 & 4 & 3 & 1 & 6 \\ 7 & 5 & 3 & 6 & 1 & 0 \\ 0 & 4 & 1 & 3 & 4 & 9 \\ 2 & 4 & 6 & 8 & 1 & -13 \end{bmatrix}_{6\times 6}.$$

2.



3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

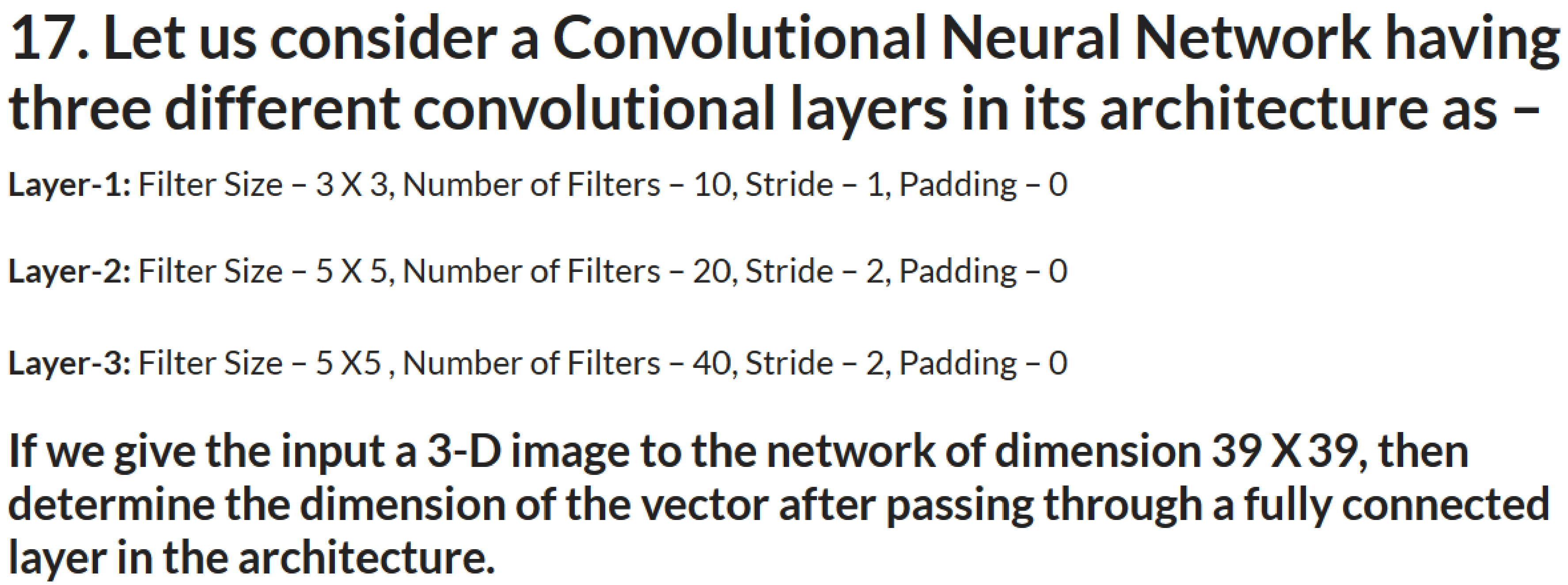
5x5
Input

0	1	2
2	2	0
0	1	2

KERNEL
3x3

Perform the operation of CNN on the above 5*5 matrix of the input image. If necessary perform padding with the value of 1 with unit strides and find out the average pooling layer of 2*2.matrix.

3. -



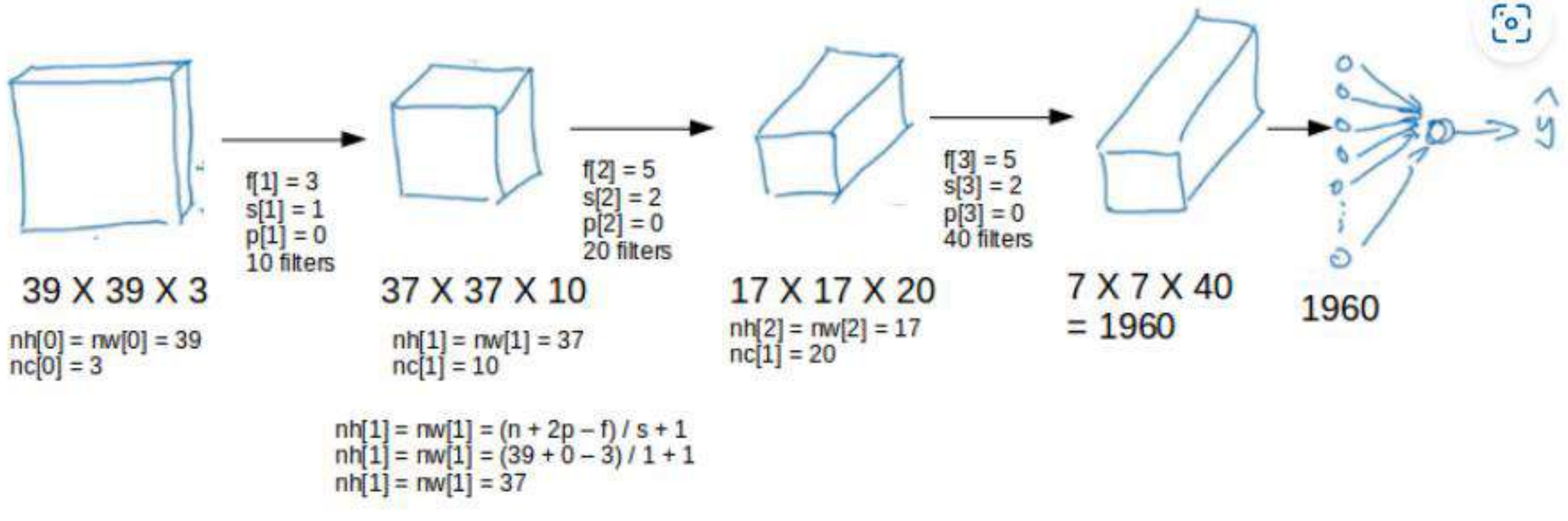
17. Let us consider a Convolutional Neural Network having three different convolutional layers in its architecture as –

Layer-1: Filter Size – 3 X 3, Number of Filters – 10, Stride – 1, Padding – 0

Layer-2: Filter Size – 5 X 5, Number of Filters – 20, Stride – 2, Padding – 0

Layer-3: Filter Size – 5 X5 , Number of Filters – 40, Stride – 2, Padding – 0

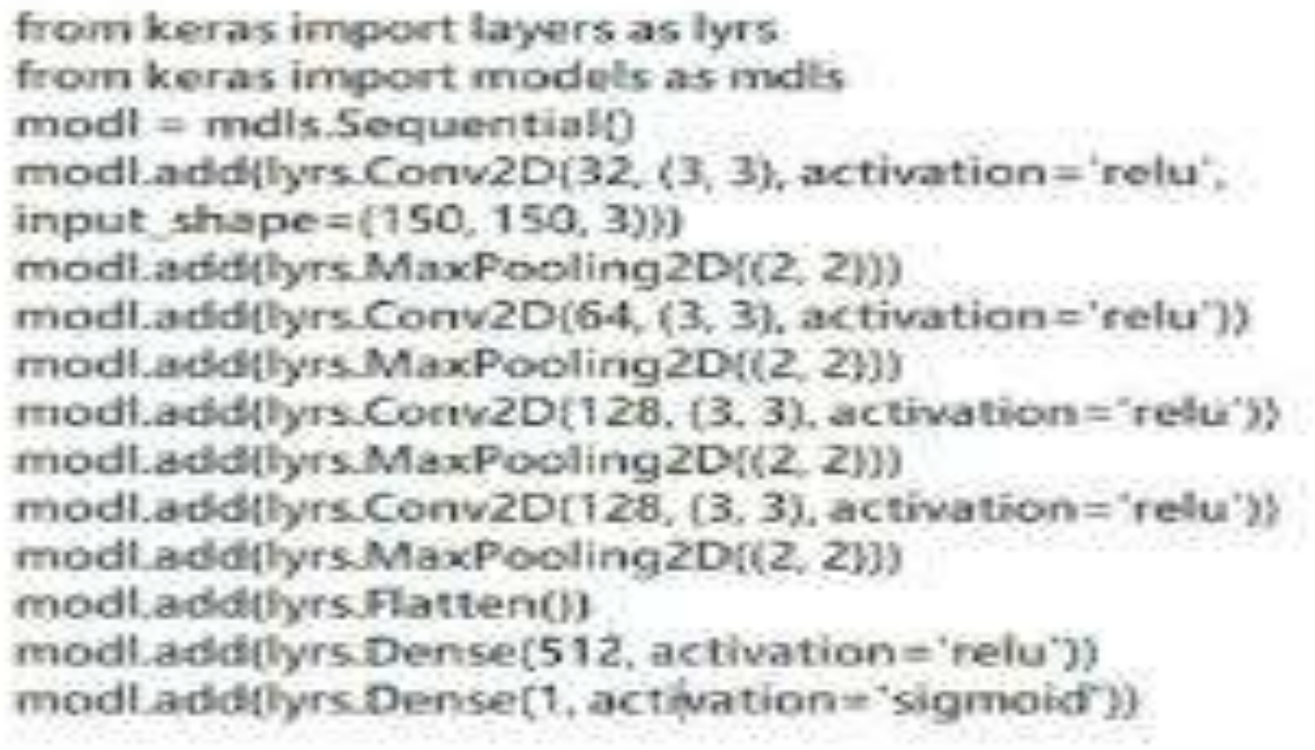
If we give the input a 3-D image to the network of dimension 39 X 39, then determine the dimension of the vector after passing through a fully connected layer in the architecture.



<u>12/09/2023 :</u>

<u>BUILDING CNN ARCHITECUTURE LAYER BY LAYER :</u>

<u>PROGRAM 1 : (13/09/2023 – lab)</u>



```
from keras import layers as lyrs
from keras import models as mdls
modl = mdls.Sequential()
modl.add(lyrs.Conv2D(32, (3, 3), activation='relu',
input_shape=(150, 150, 3)))
modl.add(lyrs.MaxPooling2D((2, 2)))
modl.add(lyrs.Conv2D(64, (3, 3), activation='relu'))
modl.add(lyrs.MaxPooling2D((2, 2)))
modl.add(lyrs.Conv2D(128, (3, 3), activation='relu'))
modl.add(lyrs.MaxPooling2D((2, 2)))
modl.add(lyrs.Conv2D(128, (3, 3), activation='relu'))
modl.add(lyrs.MaxPooling2D((2, 2)))
modl.add(lyrs.Flatten())
modl.add(lyrs.Dense(512, activation='relu'))
modl.add(lyrs.Dense(1, activation='sigmoid'))
```

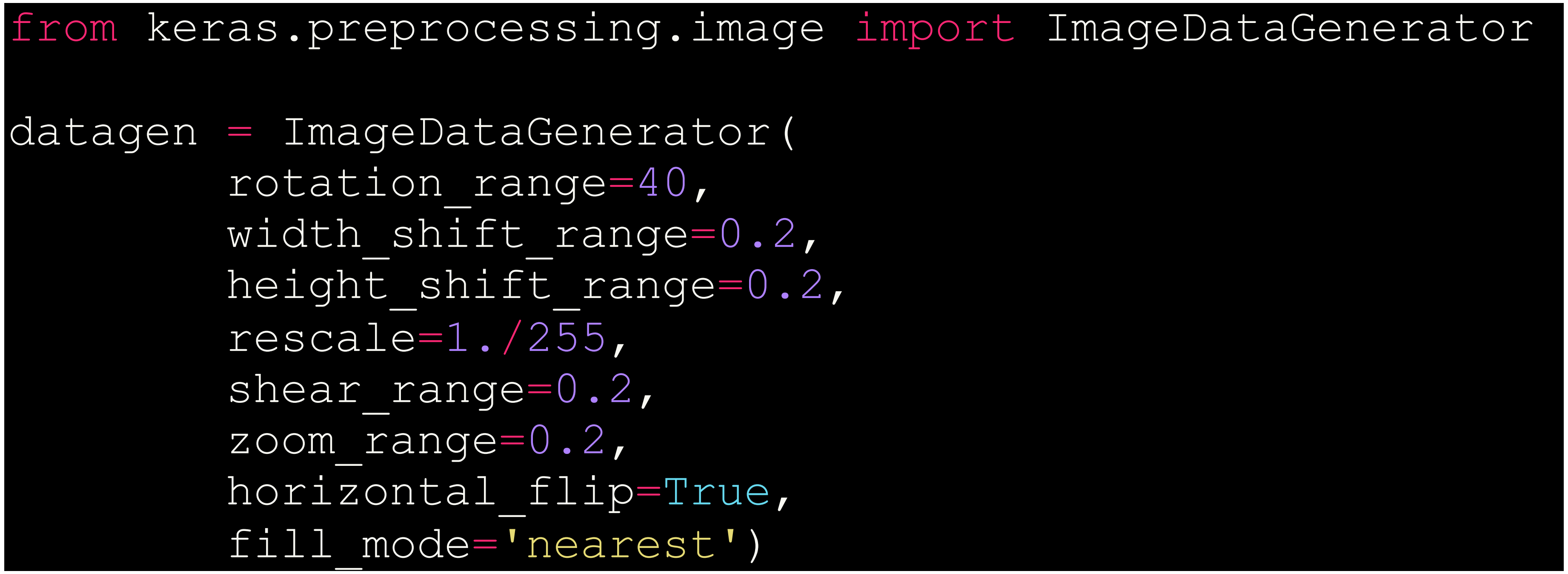
OUTPUT :

?

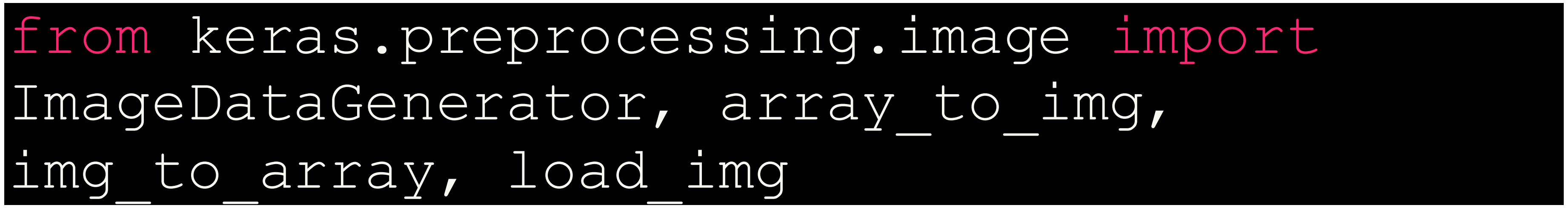
Program 2: (13/09/2023 – lab)

Building image classification when we having very little data using Data augmentation :

Lines of code :



```python
from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```



```python
from keras.preprocessing.image import
ImageDataGenerator, array_to_img,
img_to_array, load_img
```

```
datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')

img = load_img('data/train/cats/cat.0.jpg')
# this is a PIL image
x = img_to_array(img)  # this is a Numpy
array with shape (3, 150, 150)
```

```
x = x.reshape((1,) + x.shape)  # this is a
Numpy array with shape (1, 3, 150, 150)

# the .flow() command below generates
batches of randomly transformed images
# and saves the results to the `preview/`
directory
i = 0
for batch in datagen.flow(x, batch_size=1,

save_to_dir='preview', save_prefix='cat',
save_format='jpeg'):
    i += 1
    if i > 20:
            break  # otherwise the generator
would loop indefinitely
```

OUTPUT :

?

PROGRAM 3 : (13/09/2023 – lab)

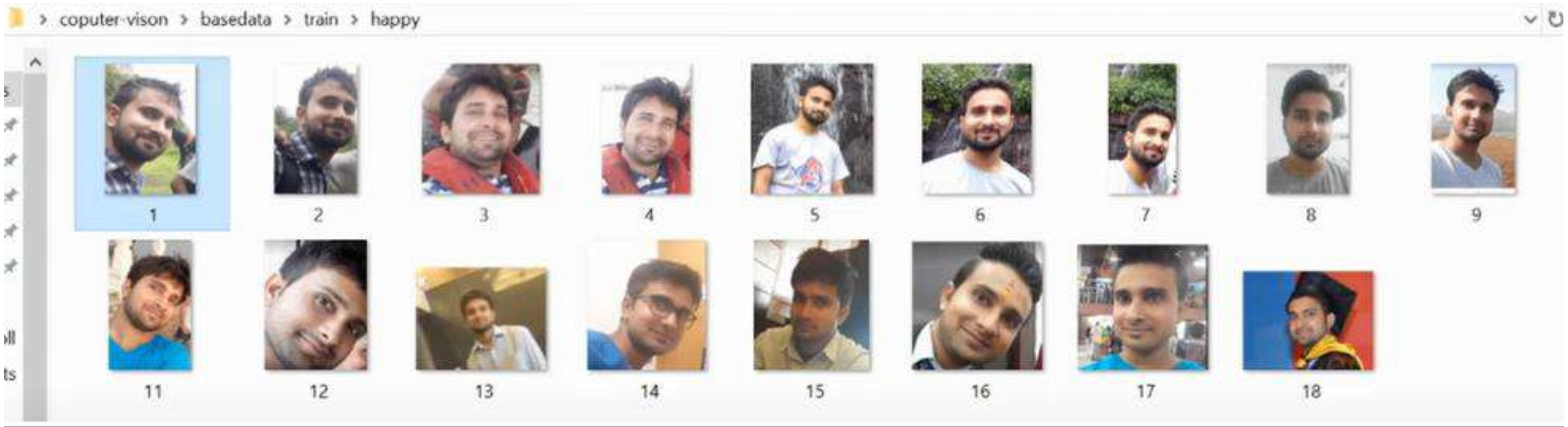
CREATE OWN DATASET AND IMPLEMENTATION USING CNN :

1) TRAINING A CNN BY LOADING YOUR OWN IMAGES :

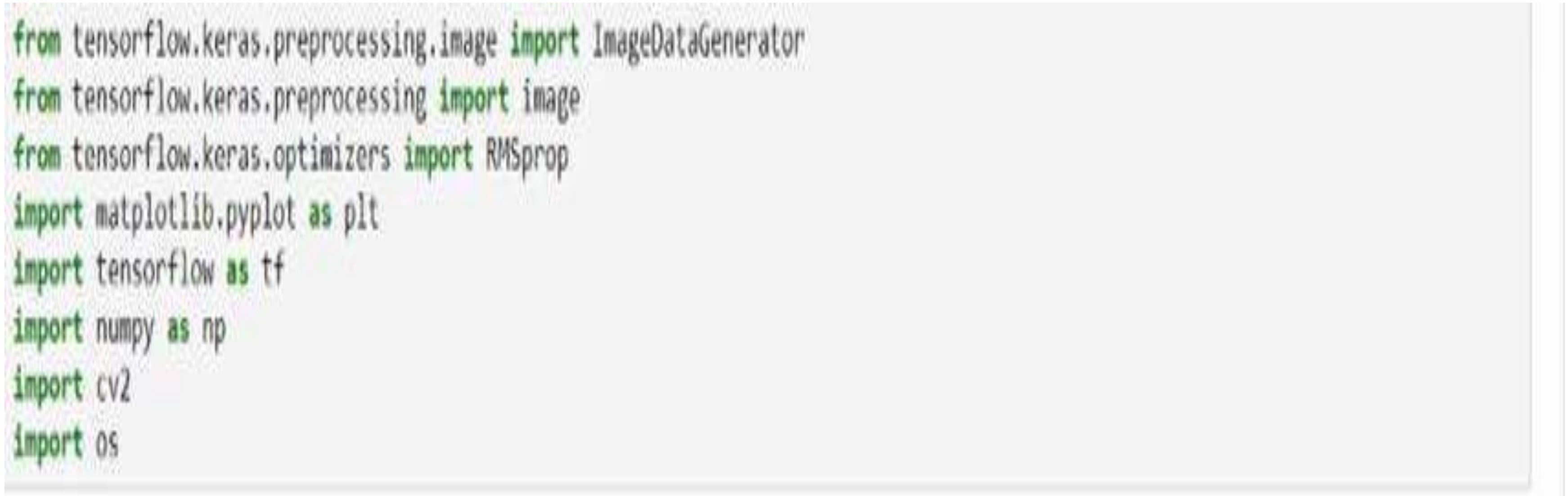
D: -> COMPUTER VISION → BASEDATA ->TRAINING, VALIDATION, TESTING

TRAINING
VALIDATION
TESTING … CREATE HAPPY , NOT HAPPY FOLDERS

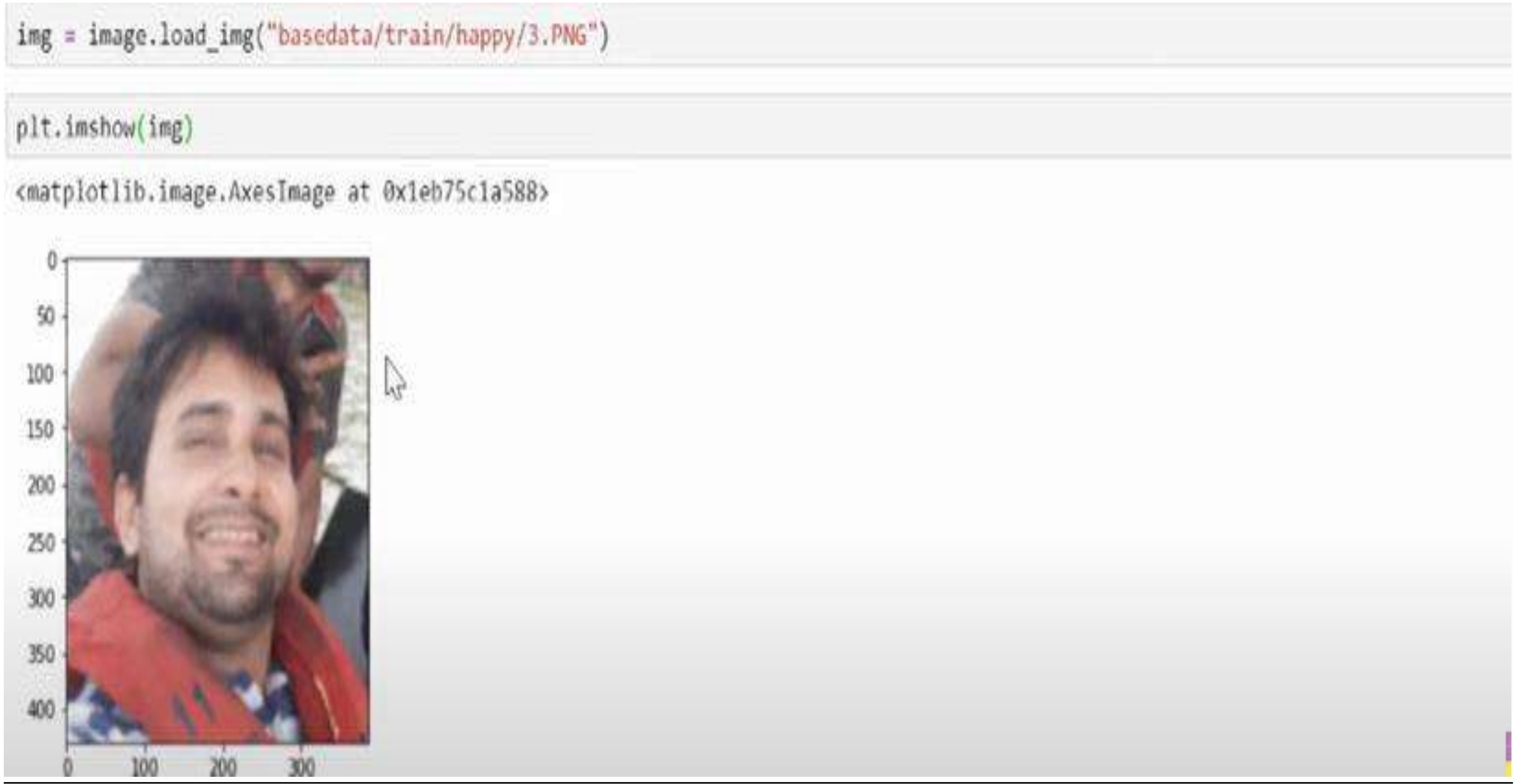
EG :



LINES OF CODE :



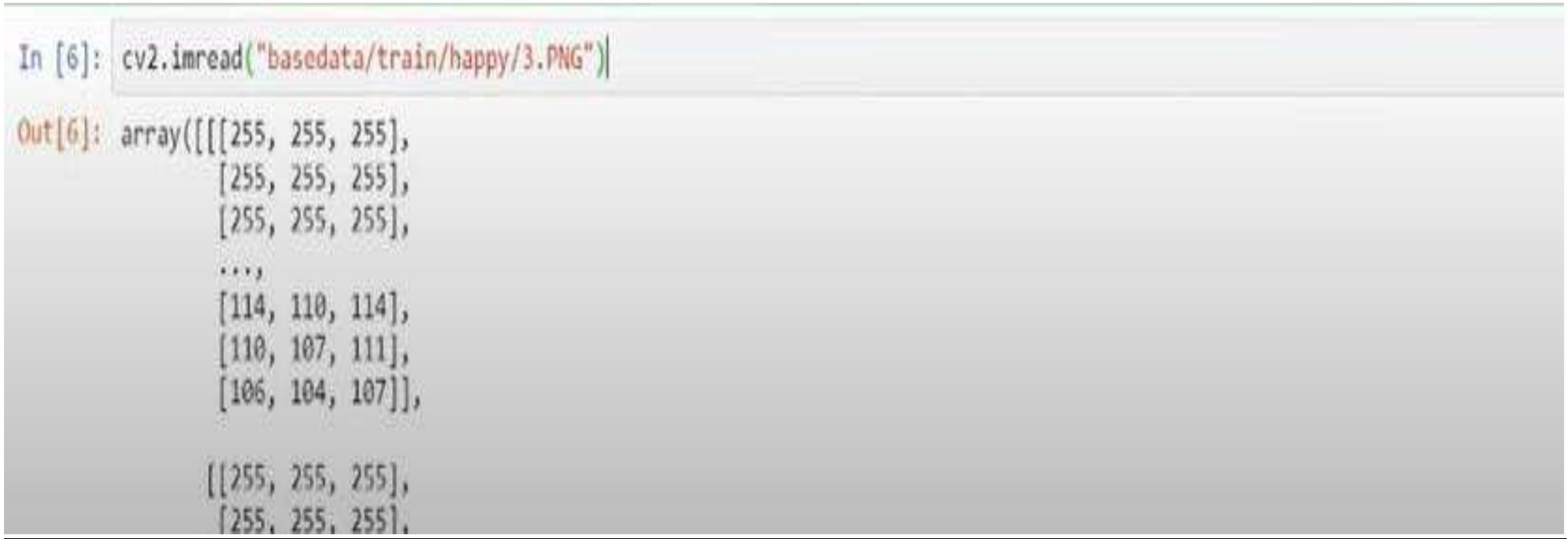
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.optimizers import RMSprop
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import cv2
import os
```



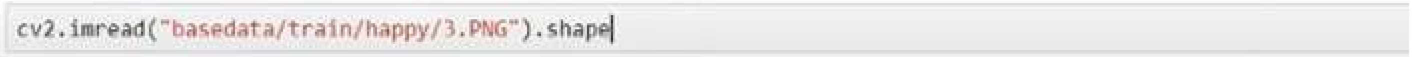
```
img = image.load_img("basedata/train/happy/3.PNG")
```

```
plt.imshow(img)
```

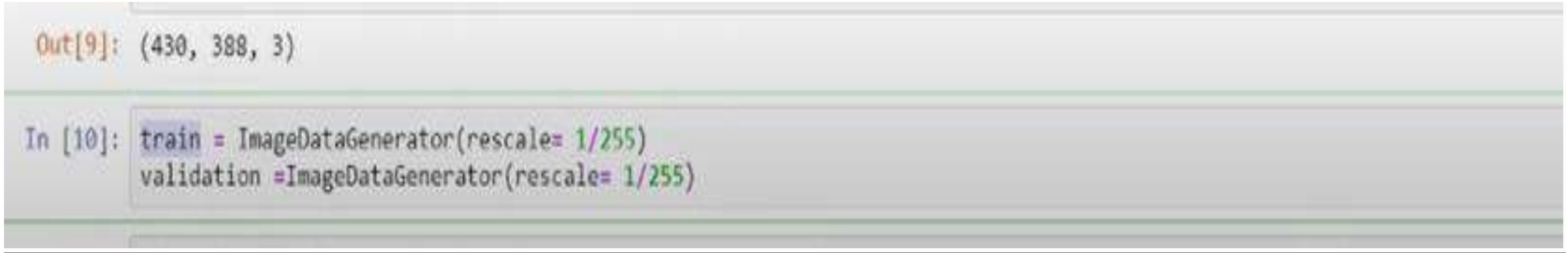
```
<matplotlib.image.AxesImage at 0x1eb75c1a588>
```



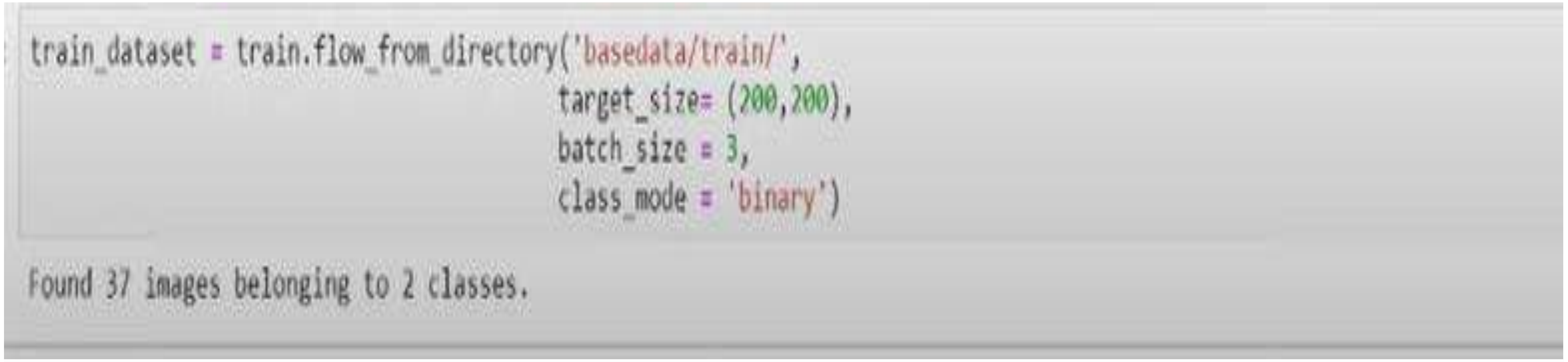
```
In [6]: cv2.imread("basedata/train/happy/3.PNG")

Out[6]: array([[[255, 255, 255],
                [255, 255, 255],
                [255, 255, 255],
                ...,
                [114, 110, 114],
                [110, 107, 111],
                [106, 104, 107]],

               [[255, 255, 255],
                [255, 255, 255],
```



```
cv2.imread("basedata/train/happy/3.PNG").shape
```

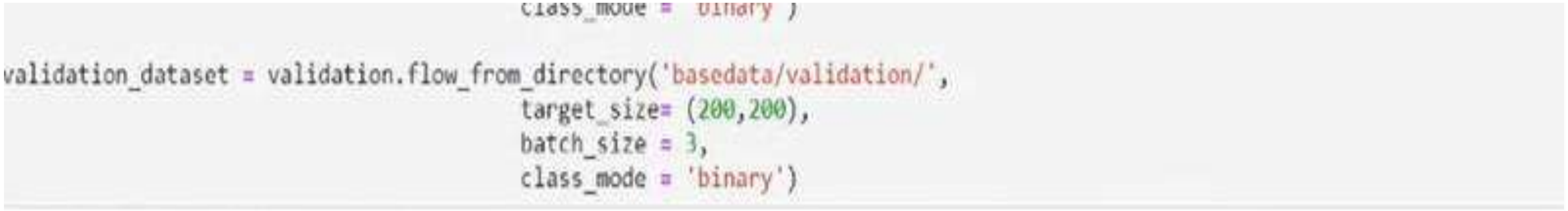


```
Out[9]: (430, 388, 3)

In [10]: train = ImageDataGenerator(rescale= 1/255)
         validation =ImageDataGenerator(rescale= 1/255)
```

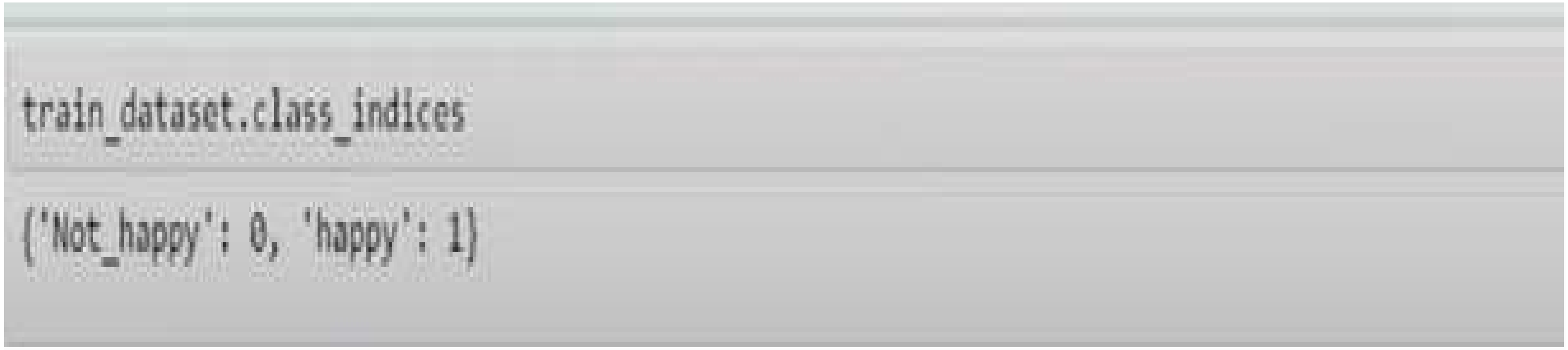


```
train_dataset = train.flow_from_directory('basedata/train/',
                                          target_size= (200,200),
                                          batch_size = 3,
                                          class_mode = 'binary')

Found 37 images belonging to 2 classes.
```

<u>2.INSERT FOR VALIDATION DATASET TOO :</u>

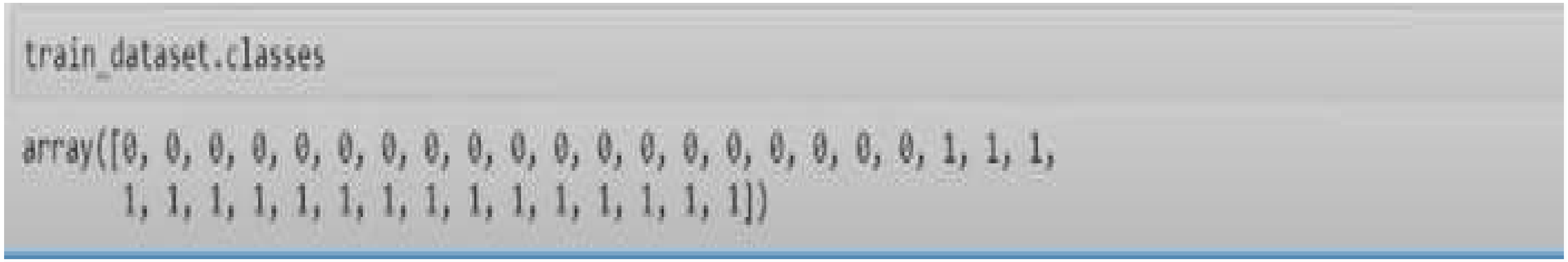


```
                                            class_mode = 'binary')

validation_dataset = validation.flow_from_directory('basedata/validation/',
                                            target_size= (200,200),
                                            batch_size = 3,
                                            class_mode = 'binary')
```



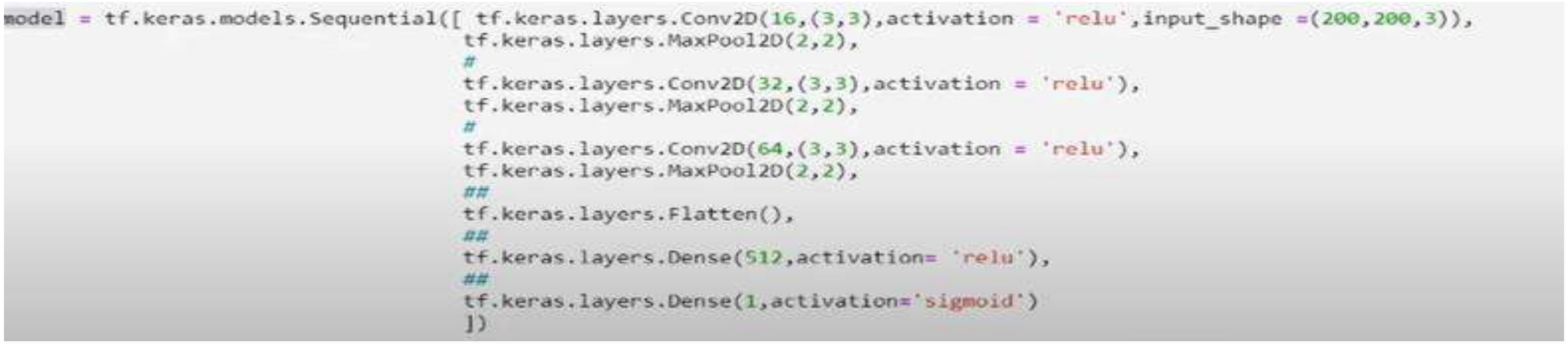
```
train_dataset.class_indices
```

```
{'Not_happy': 0, 'happy': 1}
```

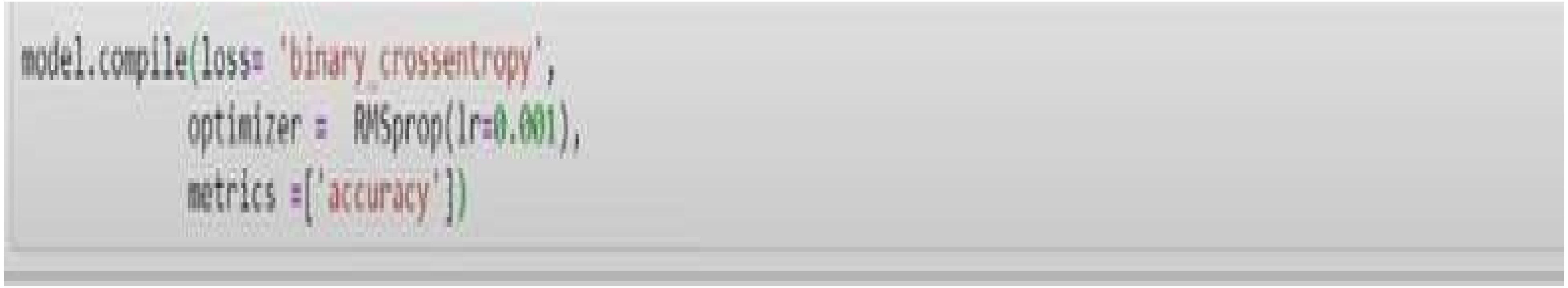


```
train_dataset.classes
```

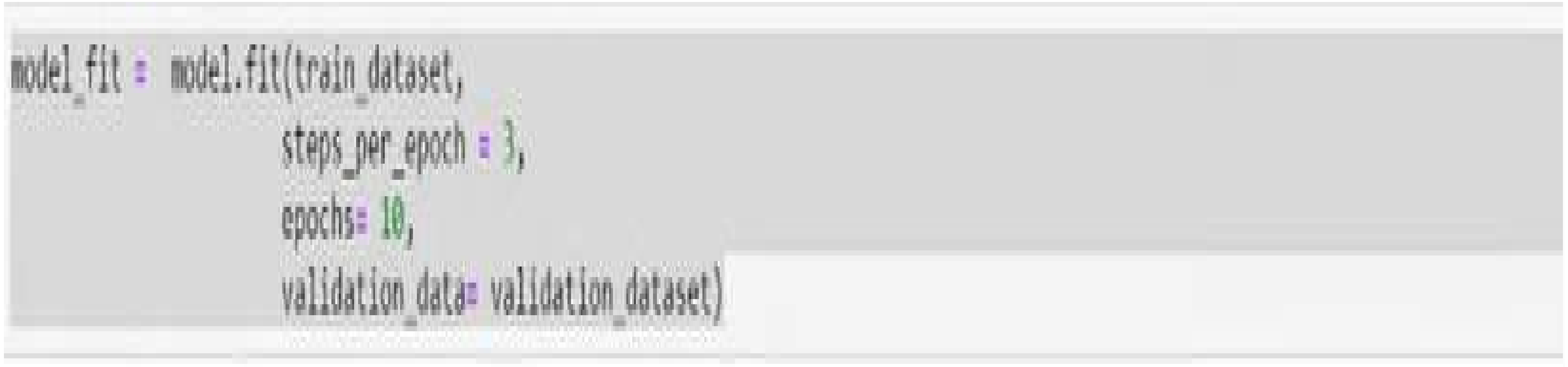
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```



```
model = tf.keras.models.Sequential([ tf.keras.layers.Conv2D(16,(3,3),activation = 'relu',input_shape =(200,200,3)),
                                    tf.keras.layers.MaxPool2D(2,2),
                                    #
                                    tf.keras.layers.Conv2D(32,(3,3),activation = 'relu'),
                                    tf.keras.layers.MaxPool2D(2,2),
                                    #
                                    tf.keras.layers.Conv2D(64,(3,3),activation = 'relu'),
                                    tf.keras.layers.MaxPool2D(2,2),
                                    ##
                                    tf.keras.layers.Flatten(),
                                    ##
                                    tf.keras.layers.Dense(512,activation= 'relu'),
                                    ##
                                    tf.keras.layers.Dense(1,activation='sigmoid')
                                    ])
```



```
model.compile(loss= 'binary_crossentropy',
              optimizer =  RMSprop(lr=0.001),
              metrics =['accuracy'])
```



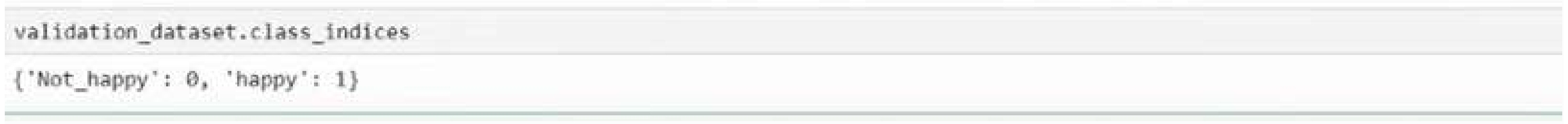
```
model_fit = model.fit(train_dataset,
                      steps_per_epoch = 3,
                      epochs= 10,
                      validation_data= validation_dataset)
```

Output :

?

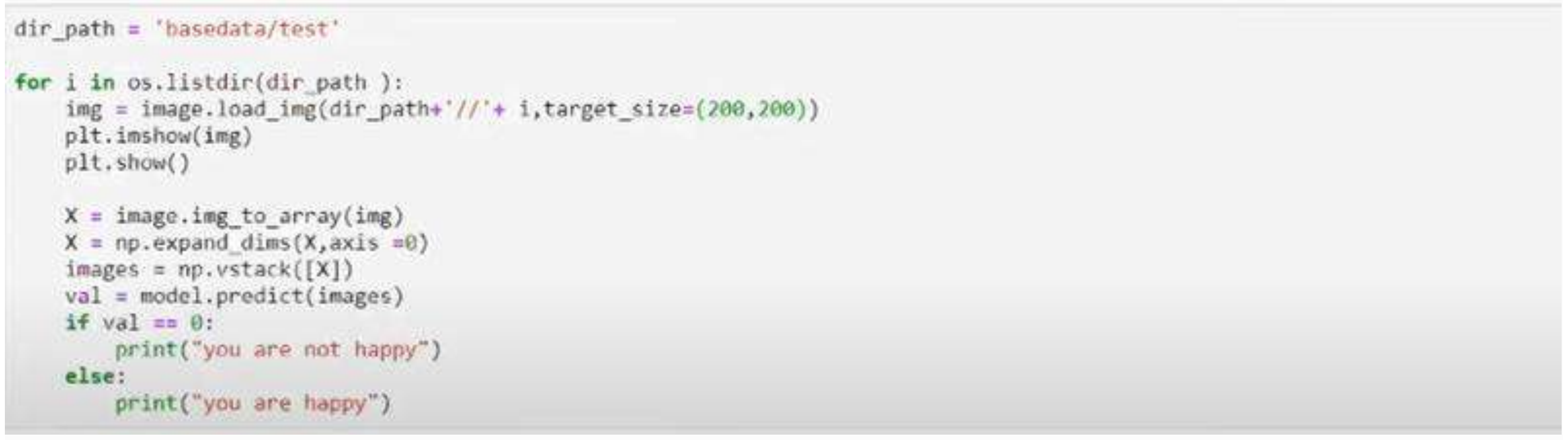
Change epoch size trail and error and compare the accuracy, then conclude at some epoch size.

3.TESTING PROCESS AND PRINTING RESULT:



```
validation_dataset.class_indices
```

```
{'Not_happy': 0, 'happy': 1}
```



```
dir_path = 'basedata/test'

for i in os.listdir(dir_path ):
    img = image.load_img(dir_path+'//'+ i,target_size=(200,200))
    plt.imshow(img)
    plt.show()

    X = image.img_to_array(img)
    X = np.expand_dims(X,axis =0)
    images = np.vstack([X])
    val = model.predict(images)
    if val == 0:
        print("you are not happy")
    else:
        print("you are happy")
```

OUTPUT :

?

PROGRAM 4 : (13/09/2023)

IMPLEMENT MAX POOLING AND AVERAGE POOLING IN CNN USING LINES OF CODE :

1)

```python
import numpy as np
from keras.models import Sequential
from keras.layers import MaxPooling2D

# define input image
image = np.array([[2, 2, 7, 3],
                  [9, 4, 6, 1],
                  [8, 5, 2, 4],
                  [3, 1, 2, 6]])
image = image.reshape(1, 4, 4, 1)

# define model containing just a single max pooling layer
model = Sequential(
    [MaxPooling2D(pool_size = 2, strides = 2)])

# generate pooled output
output = model.predict(image)

# print output image
output = np.squeeze(output)
print(output)
```

1. **Code #2 : Performing Average Pooling using keras**

Python3

```python
import numpy as np
from keras.models import Sequential
from keras.layers import AveragePooling2D

# define input image
image = np.array([[2, 2, 7, 3],
                  [9, 4, 6, 1],
                  [8, 5, 2, 4],
                  [3, 1, 2, 6]])
image = image.reshape(1, 4, 4, 1)

# define model containing just a single average pooling layer
model = Sequential(
    [AveragePooling2D(pool_size = 2, strides = 2)])

# generate pooled output
output = model.predict(image)

# print output image
output = np.squeeze(output)
print(output)
```

PROGRAMS FOR 13/09/2023 :

1) **BUILDING CNN ARCHITECTURE LAYER BY LAYER.**

2) **IMPLEMENT MIN POOLING MAX POOLING USING LINES OF CODE.**

3) **BUILD A IMAGE CLASSIFCATION MODEL USING EACH LAYER OF CNN BY CREATING YOUR OWN DATASET.**

Ex. No	**Date**	**Program name**	**Marks**	**Signature**
1.		**Classificaiton of flower species uing ANN**		
2.		**Predicting Medical Insurance Premium for customes using NN concepts.**		
3		**Implementing SLP concepts**		
4.		**BUILDING CNN ARCHITECTURE LAYER BY LAYER**		

5.		**IMPLEMENT MIN POOLING MAX POOLING USING LINES OF CODE.**		
6.		**BUILD A IMAGE CLASSIFCATION MODEL USING EACH LAYER OF CNN BY CREATING YOUR OWN DATASET.**		
7.				

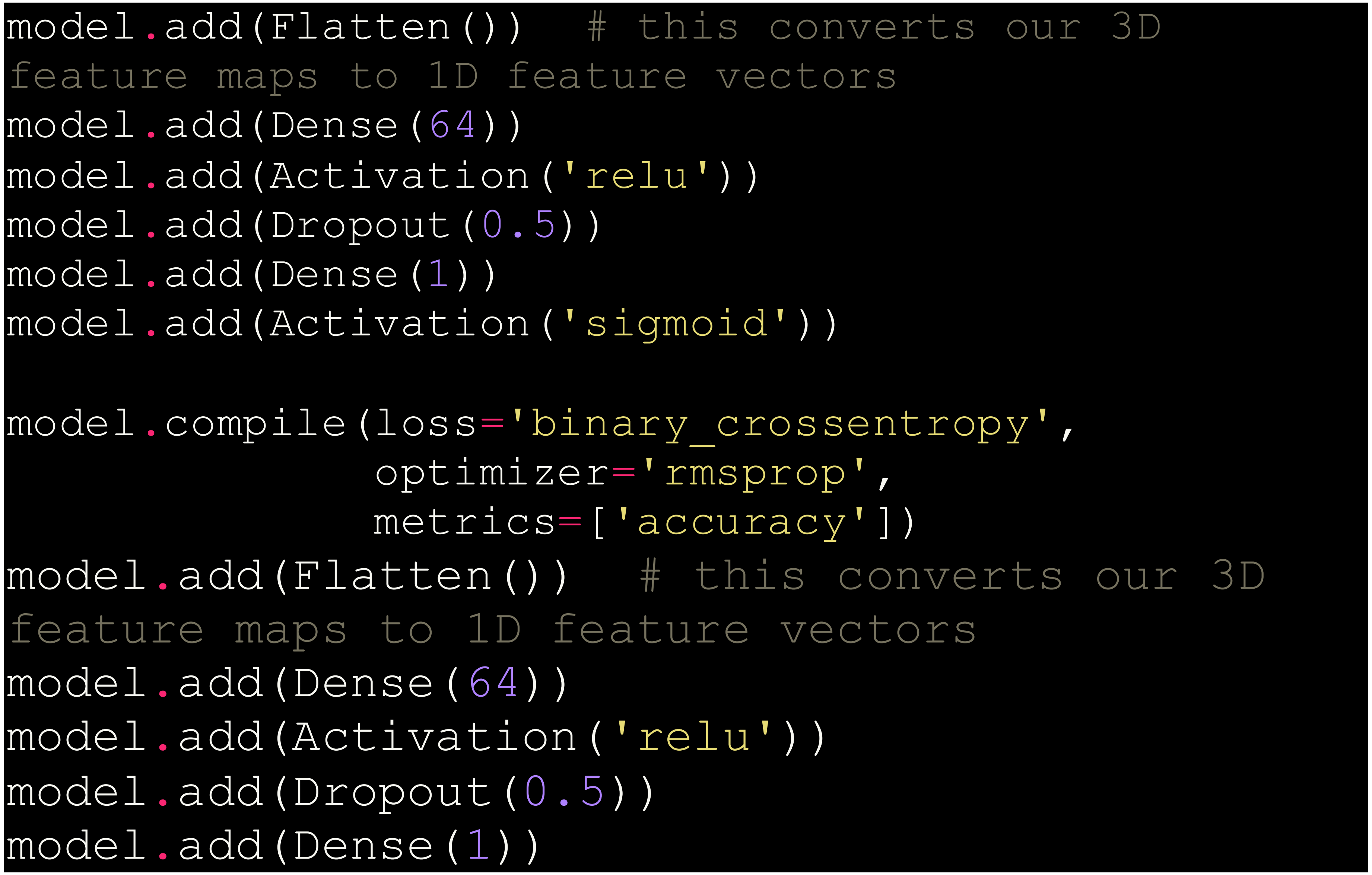
PROGRAM 4 :

Training a small convnet from scratch: 80% accuracy in 40 lines of code

The code snippet below is our first model, a simple stack of 3 convolution layers with a ReLU activation and followed by max-pooling layers. This is very similar to the architectures that Yann LeCun advocated in the 1990s for image classification (with the exception of ReLU).

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout,
Flatten, Dense
```

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(3, 150,
150)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# the model so far outputs 3D feature maps (height,
width, features)
```



```
model.add(Flatten())  # this converts our 3D
feature maps to 1D feature vectors
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
model.add(Flatten())  # this converts our 3D
feature maps to 1D feature vectors
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

```
batch_size = 16

# this is the augmentation configuration we
will use for training
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
```

```
# this is the augmentation configuration we
will use for testing:
# only rescaling
test_datagen =
ImageDataGenerator(rescale=1./255)

# this is a generator that will read pictures
found in
# subfolers of 'data/train', and indefinitely
generate
# batches of augmented image data
train_generator =
train_datagen.flow_from_directory(
        'data/train',  # this is the target
directory
```

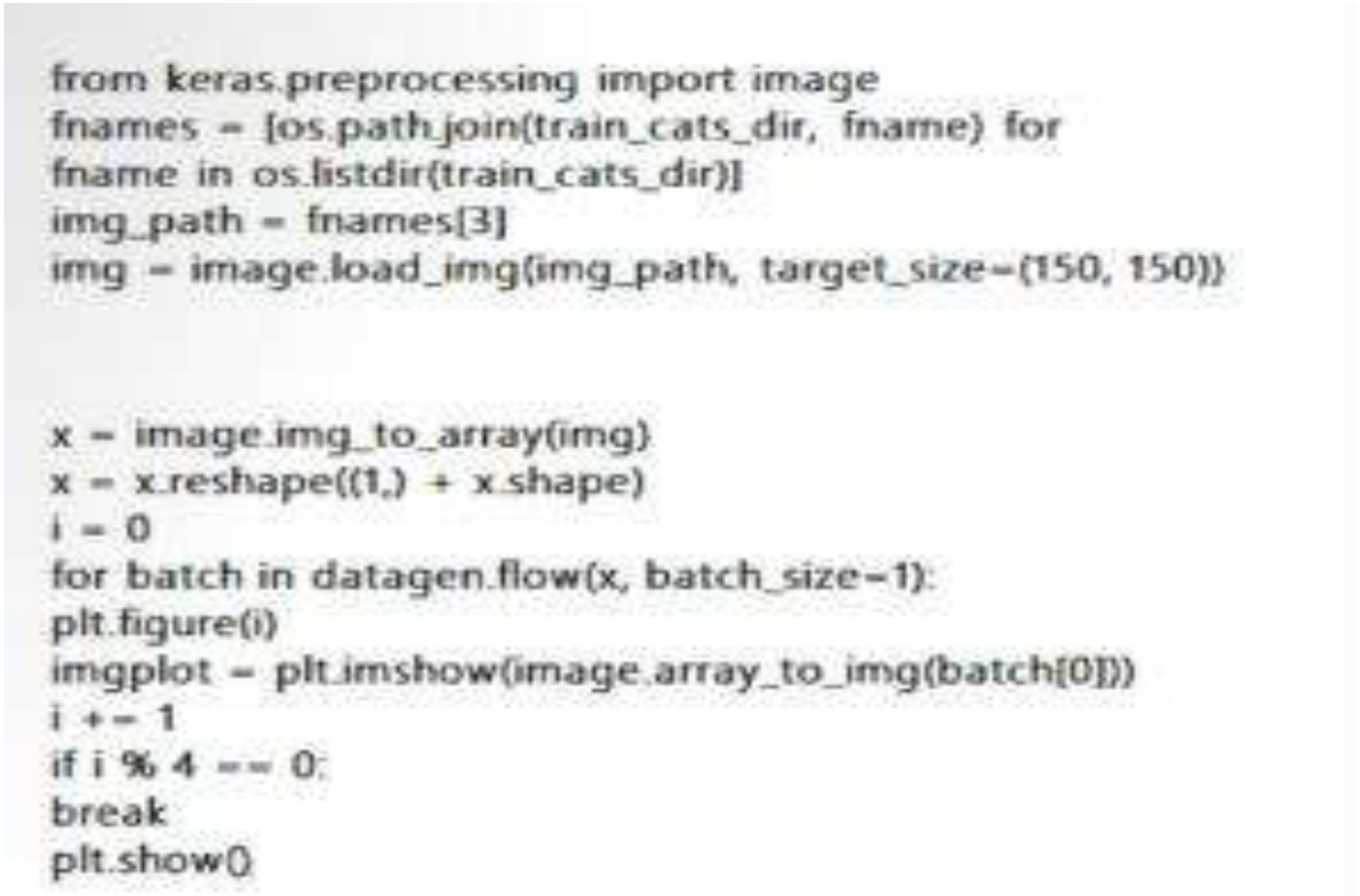
```
        target_size=(150, 150),  # all images
will be resized to 150x150
        batch_size=batch_size,
        class_mode='binary')  # since we use
binary_crossentropy loss, we need binary labels

# this is a similar generator, for validation
data
validation_generator =
test_datagen.flow_from_directory(
        'data/validation',
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')
```

```
model.fit_generator(
        train_generator,
        steps_per_epoch=2000 // batch_size,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=800 // batch_size)
model.save_weights('first_try.h5')
```

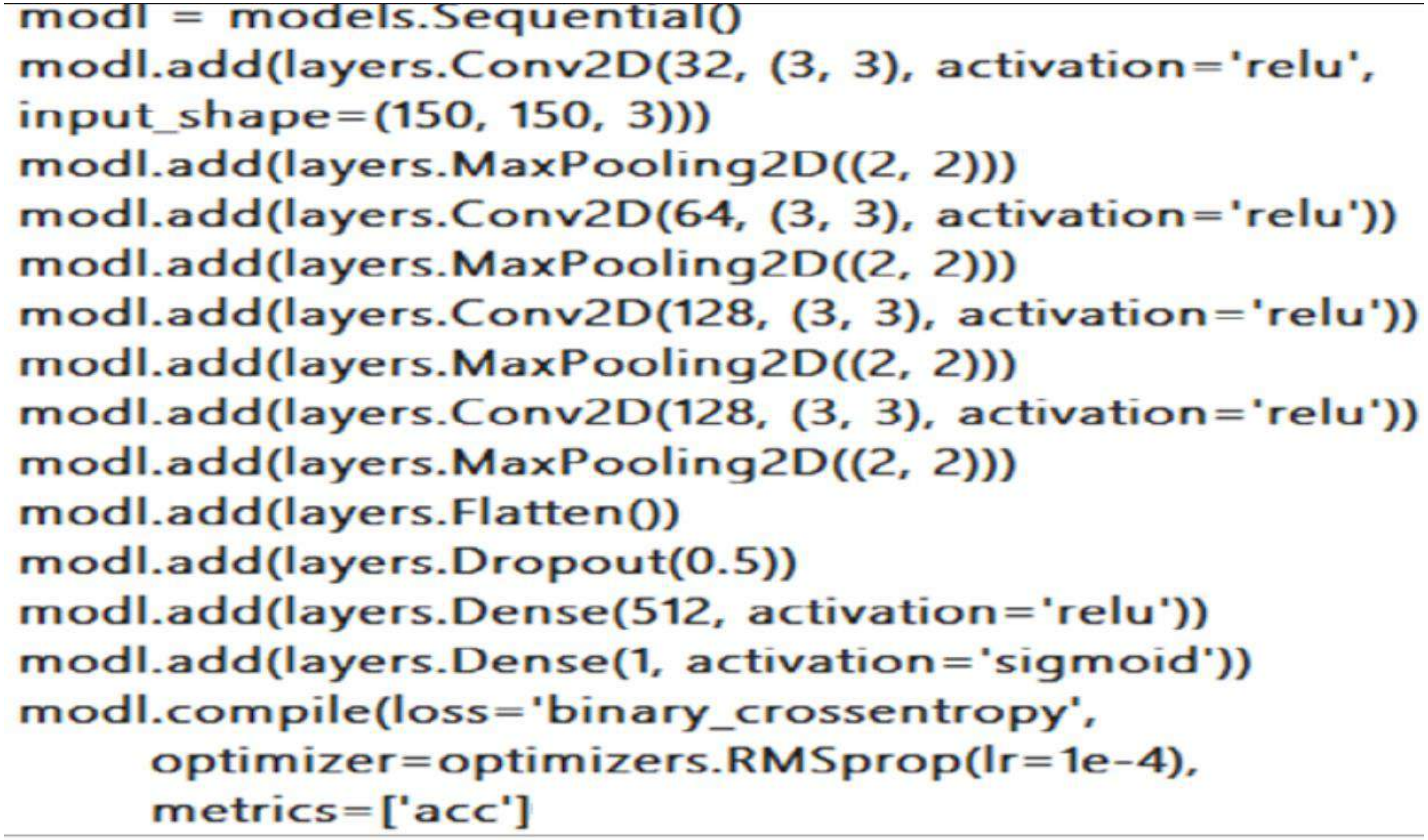
<u>output :</u>

This approach gets us to a validation accuracy of 0.79-0.81 after 50 epochs (a number that was picked arbitrarily --because the model is small and uses aggressive dropout, it does not seem to be overfitting too much by that point). So at the time the Kaggle competition was launched, we would be already be "state of the art" --with 8% of the data, and no effort to optimize our architecture or hyperparameters. In fact, in the Kaggle competition, this model would have scored in the top 100 (out of 215 entrants). I guess that at least 115 entrants weren't using deep learning ;)

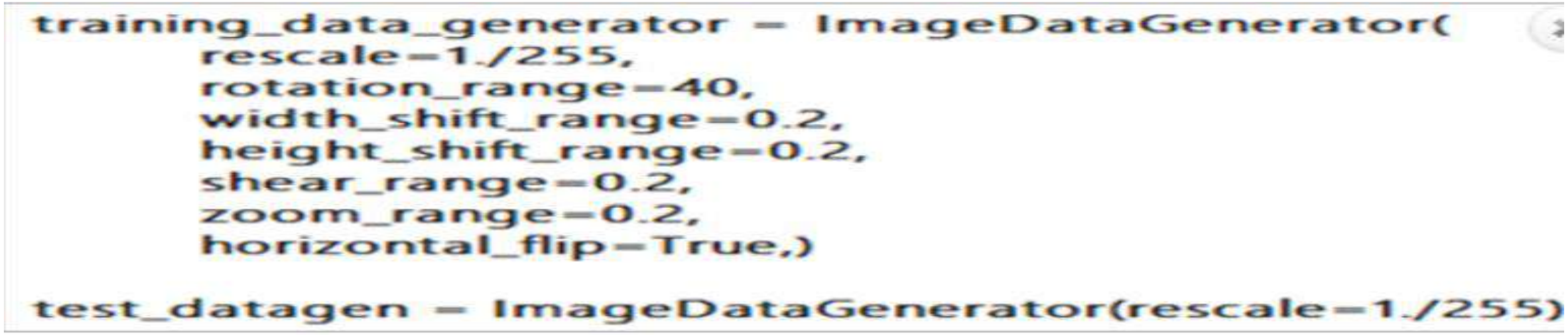
Program 2 :



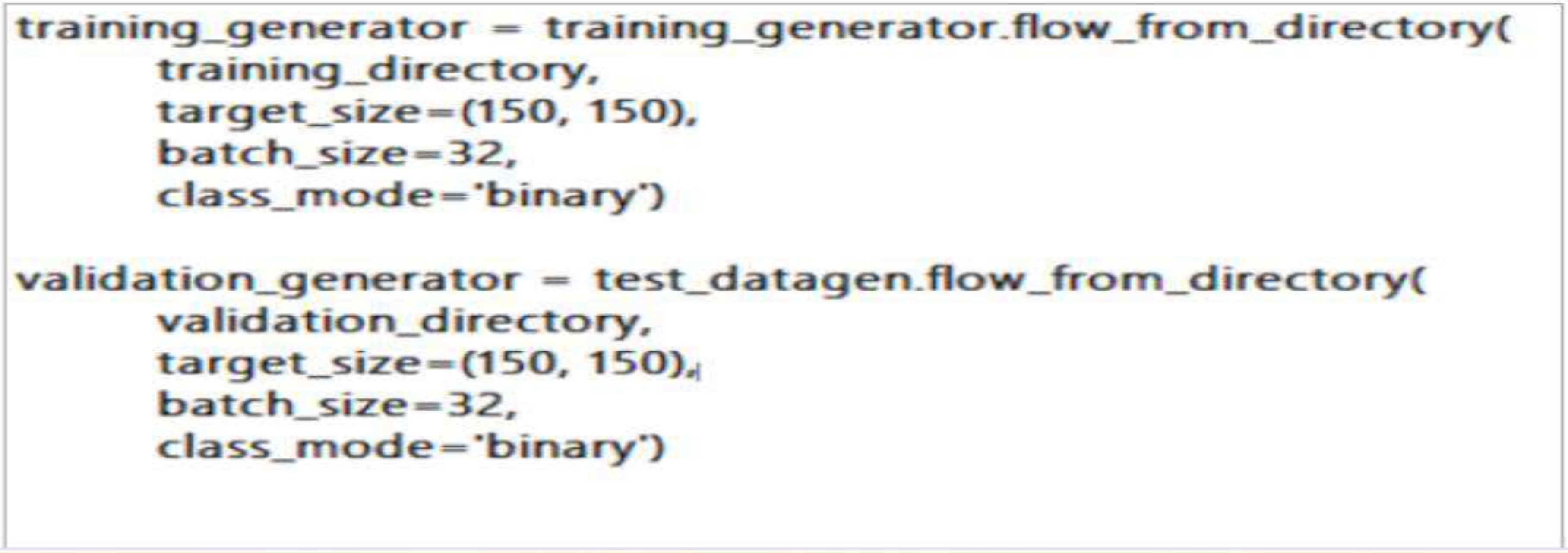
```
from keras.preprocessing import image
fnames = [os.path.join(train_cats_dir, fname) for
fname in os.listdir(train_cats_dir)]
img_path = fnames[3]
img = image.load_img(img_path, target_size=(150, 150))


x = image.img_to_array(img)
x = x.reshape((1,) + x.shape)
i = 0
for batch in datagen.flow(x, batch_size=1):
plt.figure(i)
imgplot = plt.imshow(image.array_to_img(batch[0]))
i += 1
if i % 4 == 0:
break
plt.show()
```



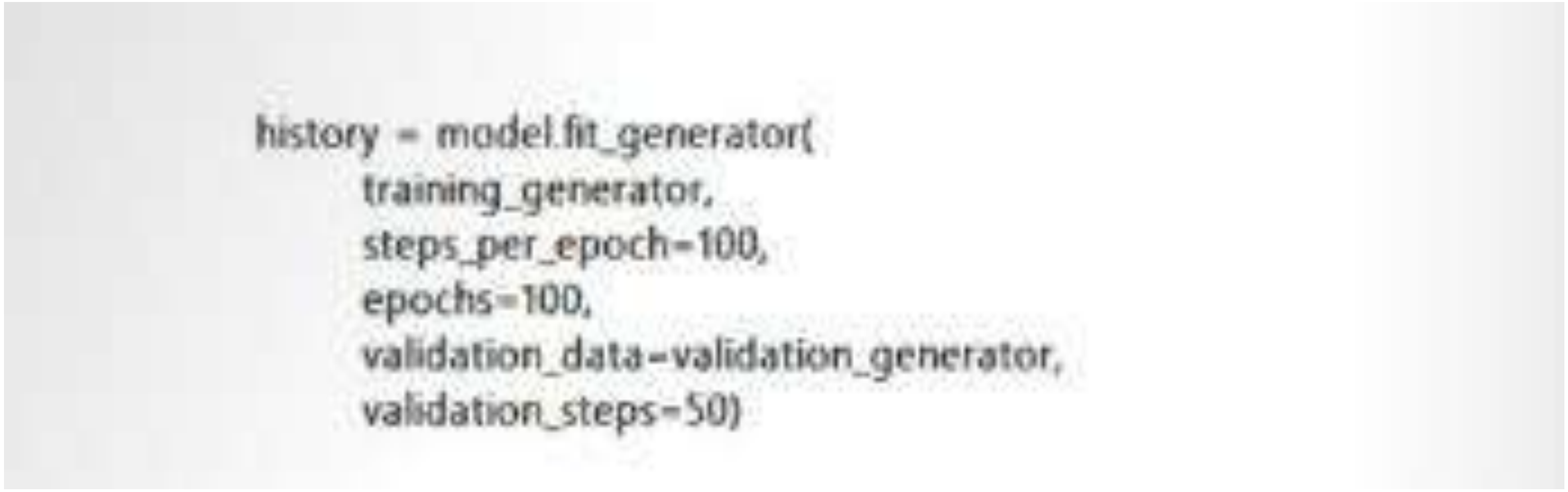
```
modl = models.Sequential()
modl.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(150, 150, 3)))
modl.add(layers.MaxPooling2D((2, 2)))
modl.add(layers.Conv2D(64, (3, 3), activation='relu'))
modl.add(layers.MaxPooling2D((2, 2)))
modl.add(layers.Conv2D(128, (3, 3), activation='relu'))
modl.add(layers.MaxPooling2D((2, 2)))
modl.add(layers.Conv2D(128, (3, 3), activation='relu'))
modl.add(layers.MaxPooling2D((2, 2)))
modl.add(layers.Flatten())
modl.add(layers.Dropout(0.5))
modl.add(layers.Dense(512, activation='relu'))
modl.add(layers.Dense(1, activation='sigmoid'))
modl.compile(loss='binary_crossentropy',
     optimizer=optimizers.RMSprop(lr=1e-4),
     metrics=['acc']
```



```
training_data_generator = ImageDataGenerator(
      rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,)

test_datagen = ImageDataGenerator(rescale=1./255)
```

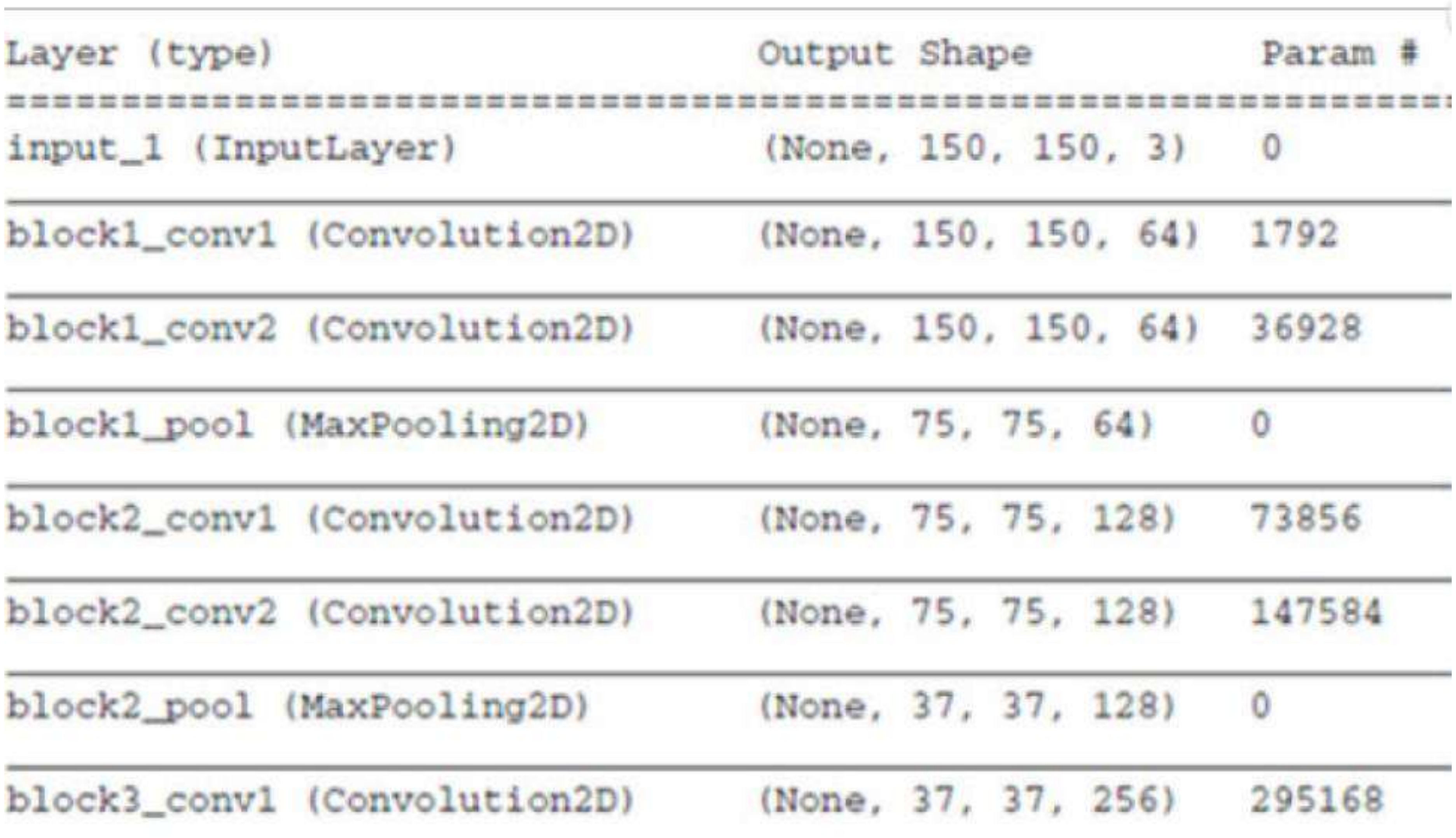


```
training_generator = training_generator.flow_from_directory(
        training_directory,
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_directory,
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')
```



```
history = model.fit_generator(
        training_generator,
        steps_per_epoch=100,
        epochs=100,
        validation_data=validation_generator,
        validation_steps=50)
```

Output :



```
Layer (type)                     Output Shape            Param #
=================================================================
input_1 (InputLayer)             (None, 150, 150, 3)     0
_________________________________________________________________
block1_conv1 (Convolution2D)     (None, 150, 150, 64)    1792
_________________________________________________________________
block1_conv2 (Convolution2D)     (None, 150, 150, 64)    36928
_________________________________________________________________
block1_pool (MaxPooling2D)       (None, 75, 75, 64)      0
_________________________________________________________________
block2_conv1 (Convolution2D)     (None, 75, 75, 128)     73856
_________________________________________________________________
block2_conv2 (Convolution2D)     (None, 75, 75, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)       (None, 37, 37, 128)     0
_________________________________________________________________
block3_conv1 (Convolution2D)     (None, 37, 37, 256)     295168
```