CASTING PRODUCT INSPECTION USING TRANSFER LEARNING

BACHELOR OF TECHNOLOGY in
PRODUCTION & INDUSTRIAL ENGINEERING By

Shashi Nandan Prasad(2017UGPI015)
Chandan Kumar(2017UGPI043)
Pradeep Kumar Saroj(2017UGPI031)

UNDER THE ESTEEMED GUIDANCE OF:-

Dr. Kanika Prasad
Prof. Dinesh Kumar

DEPARTMENT OF PRODUCTION AND INDUSTRIAL ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
JAMSHEDPUR -831014

CERTIFICATE

This is to certify that the dissertation entitled "**Casting Product Inspection using Transfer Learning**" submitted by **Mr. Shashi Nandan Prasad, Reg. No. 2017UGPI015,** in partial fulfilment of the requirements for the award of Bachelor of Technology in Production and Industrial Engineering, Department of Production and Industrial Engineering, NIT Jamshedpur is an authentic work carried out by them under my supervision and guidance.

To the best of my knowledge, the matter embodied in the dissertation has not been submitted to any other University/Institute for the award of any Degree/Diploma.

Date:

Dr. Kanika Prasad
Department of Production and Industrial
Engineering
NIT Jamshedpur

Prof. Dinesh Kumar
Department of Production and Industrial
Engineering
NIT Jamshedpur

Dr. Shashi Bhushan Prasad
Head of Department
Production and Industrial Engineering
NIT Jamshedpur

CERTIFICATE

This is to certify that the dissertation entitled "**Casting Product Inspection using Transfer Learning"** submitted by **Mr. Chandan Kumar, Reg. No. 2017UGPI043,** in partial fulfilment of the requirements for the award of Bachelor of Technology in Production and Industrial Engineering, Department of Production and Industrial Engineering, NIT Jamshedpur is an authentic work carried out by them under my supervision and guidance.

To the best of my knowledge, the matter embodied in the dissertation has not been submitted to any other University/Institute for the award of any Degree/Diploma.

Date:

Dr. Kanika Prasad
Department of Production and Industrial Engineering
NIT Jamshedpur

Prof. Dinesh Kumar
Department of Production and Industrial Engineering
NIT Jamshedpur

Dr. Shashi Bhushan Prasad
Head of Department
Production and Industrial Engineering
NIT Jamshedpur

CERTIFICATE

This is to certify that the dissertation entitled “**Casting Product Inspection using Transfer Learning”** submitted by **Mr. Pradeep Kumar Saroj, Reg. No. 2017UGPI031,** in partial fulfilment of the requirements for the award of Bachelor of Technology in Production and Industrial Engineering, Department of Production and Industrial Engineering, NIT Jamshedpur is an authentic work carried out by them under my supervision and guidance.

To the best of my knowledge, the matter embodied in the dissertation has not been submitted to any other University/Institute for the award of any Degree/Diploma.

Date:

Dr. Kanika Prasad
Department of Production and Industrial Engineering
NIT Jamshedpur

Prof. Dinesh Kumar
Department of Production and Industrial Engineering
NIT Jamshedpur

Dr. Shashi Bhushan Prasad
Head of Department
Production and Industrial Engineering
NIT Jamshedpur

DECLARATION

I hereby declare that the work reported in this dissertation is original and has been carried out by me independently in the **Department of Production and Industrial Engineering, National Institute of Technology Jamshedpur** under the guidance of **Dr. Kanika Prasad and Prof. Dinesh Kumar.** I also declare that this work has not formed the basis for the award of any other Degree, Diploma, or similar title of any university or institution.

Date:
Place:

Shashi Nandan Prasad
Reg.No-2017UGPI015
Bachelors in Technology(Hons)
NIT Jamshedpur

DECLARATION

I hereby declare that the work reported in this dissertation is original and has been carried out by me independently in the **Department of Production and Industrial Engineering, National Institute of Technology Jamshedpur** under the guidance of **Dr. Kanika Prasad and Prof. Dinesh Kumar.** I also declare that this work has not formed the basis for the award of any other Degree, Diploma, or similar title of any university or institution.

Date:
Place:

Chandan Kumar
Reg.No-2017UGPI043
Bachelors in Technology(Hons)
NIT Jamshedpur

DECLARATION

I hereby declare that the work reported in this dissertation is original and has been carried out by me independently in the **Department of Production and Industrial Engineering, National Institute of Technology Jamshedpur** under the guidance of **Dr. Kanika Prasad and Prof. Dinesh Kumar.** I also declare that this work has not formed the basis for the award of any other Degree, Diploma, or similar title of any university or institution.

Date:
Place:

Pradeep Kumar Saroj
Reg.No-2017UGPI031
Bachelors in Technology(Hons)
NIT Jamshedpur

ACKNOWLEDGEMENT

One of the most pleasant parts of writing a report is the opportunity to thank those who have contributed towards it. Unfortunately, the list of expressions of thanks, no matter how extensive, is always incomplete and inadequate. These acknowledgements are no exceptions.

With immense pleasure, I would like to express my gratitude and sincere indebtedness to my guide **Dr. Kanika Prasad,** Assistant Professor and **Prof. Dinesh Kumar**,Assistant Professor,Department of Production and Industrial,NIT Jamshedpur for providing sagacious guidance, all sort of assistance, fruitful discussions, motivation throughout the dissertation period, without which it would have not been possible to bring out the present project report.

A word of thanks must also go to all the faculty members of Department of Production and Industrial Engineering, NIT Jamshedpur for their kind co-operation and support.

I extend my sincere gratitude to my parents and every members of my family for their endless inspiration, support, and guidance throughout my whole life.

Last but not least, I would like to thank & extend my hearties wishes to my all B.Tech friends for their warm friendship & cooperation rendered throughout the course.

Shashi Nandan Prasad(Reg.No-2017UGPI015)
Chandan Kumar(Reg.No-2017UGPI043)
Pradeep Kumar Saroj(Reg.No-2017UGPI031)

Date:
Place:

TABLE OF CONTENTS

ABSTRACT

- In the Manufacturing Industry,in order to check the quality of the casting product i.e whether the product is defective or not there is an inspection department where some samples out of the total products are checked and if there is no defective found in samples then all the products are declared Non defective and is proceeded further.

- It is also time consuming to check all those samples manually.
- Later on there might be possibility that out of all the products some products may be found defective due to which company may suffer loss as it may hamper their production flow.
- In order to inspect the quality of casting product,we can do it by applying the concept of Transfer Learning using VGG19 architecture.

- Rather than checking few samples,we can check the quality of all the products by training some datasets and based on which we can just insert the image(test image) in the directory.

- Based on the features acquired by the machine after getting trained on training datasets,it will be able to predict the quality of the product of the test datasets in just few seconds.

LITERATURE REVIEW

Paper Titles	Image Classification-Cat and Dog Images
Authors	Tushar Jajodia(Student, Department of Information Technology, Maharaja Agrasen Institute of Technology),
	Pankaj Garg(2Assistant Professor, Department of Information Technology, Maharaja Agrasen Institute of Technology)
Year	12th December,2019
Methods	Convolution Neural Network Technique
Validations	CNN along with Data Augmentation
Limitations	1.)Common features were not pretrained
	2.)Time Complexity was more
	3.)More epochs are required in order to increase the performance of the model

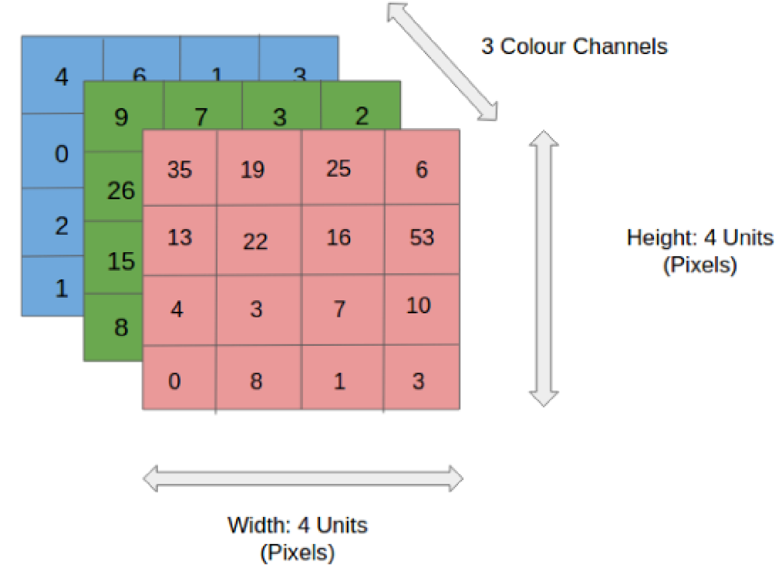
Paper Titles	CNN Features off-the-shelf: an Astounding Baseline for Recognition
Authors	Ali Sharif Razavian Hossein Azizpour
	Stefan Carlsson
	Josephine Sullivan
	CVAP, KTH (Royal Institute of Technology)
	Stockholm, Sweden
Year	2014
Methods	CNN
Validations	Linear SVM Classififer
Limitations	1.)High computational cost
	2.)Lot of training data is required
	3.)Common features were not pretrained

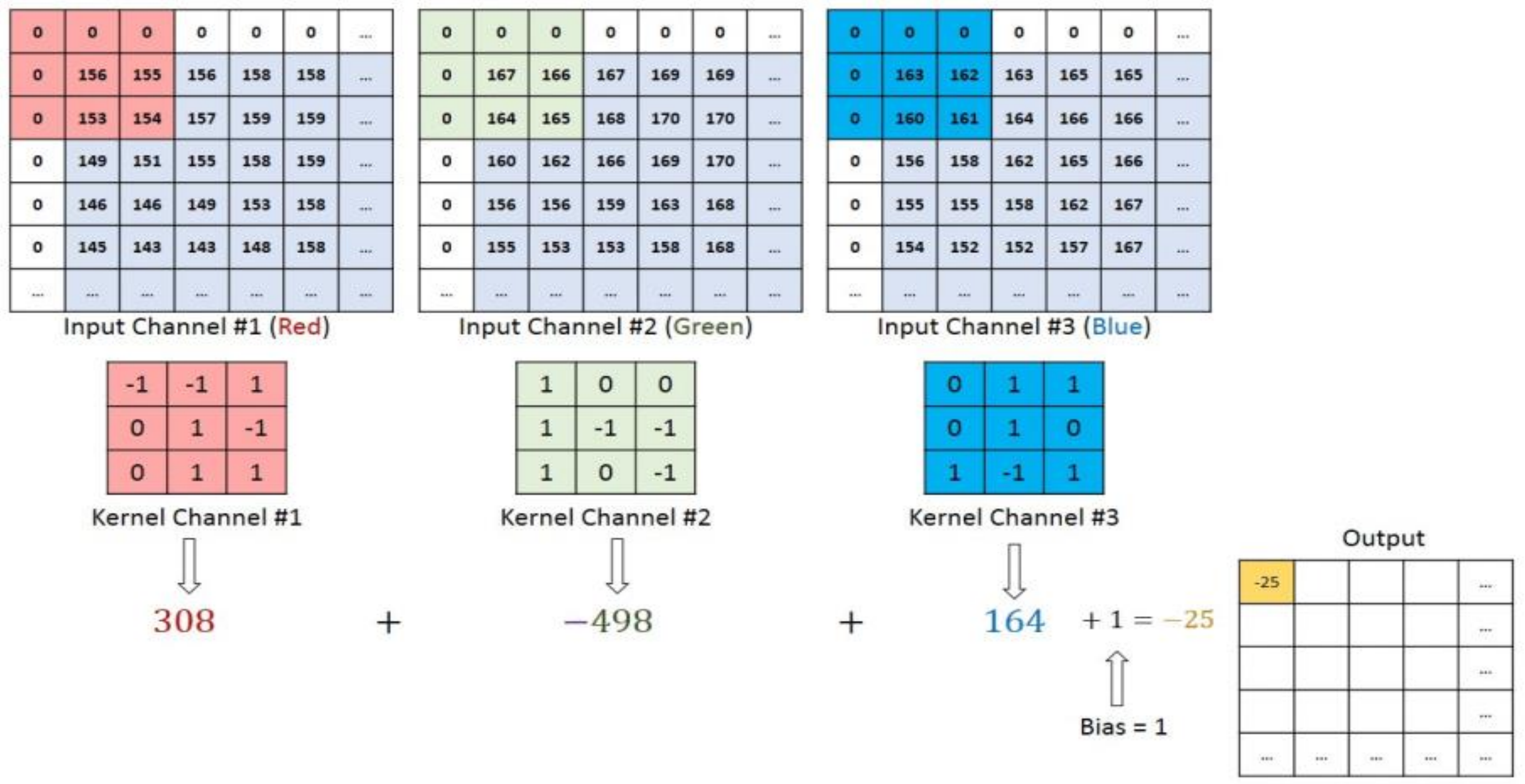
Paper Titles	CNN-RNN: A Unified Framework for Multi-label Image Classification
Authors	Jiang Wang1 Yi Yang, Junhua Mao2 Zhiheng Huang, Chang Huan Wei Xu1
	Baidu Research University of California at Los Angles
Year	27th-30th June 2016
Methods	CNN along with RNN framework
Validations	LSTM technique
Limitations	1.)Common features were not pretrained
	2.)More time complexity

CONVOLUTIONAL NEURAL NETWORK

- A **Convolutional Neural Network (CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

- In the figure, we have an RGB image which has been separated by its three color planes — Red, Green, and Blue. There are a number of such color spaces in which images exist — Grayscale, RGB, HSV, CMYK, etc.





0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

$$308 \quad + \quad -498 \quad + \quad 164 \quad + 1 = -25$$

Bias = 1

Output

-25				...
				...
				...
				...
...	...	...	...	...

- ❑ The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image.
- ❑ **Matrix multiplication** takes place between input image matrix and filter/kernel matrix in order to get convoluted features like vertical image edge, horizontal image edge features which help to identify different channels of images.
- ❑ The dimension of **Convoluted layer** is determined using the formula **n-f+1** where,

  n=input image dimension

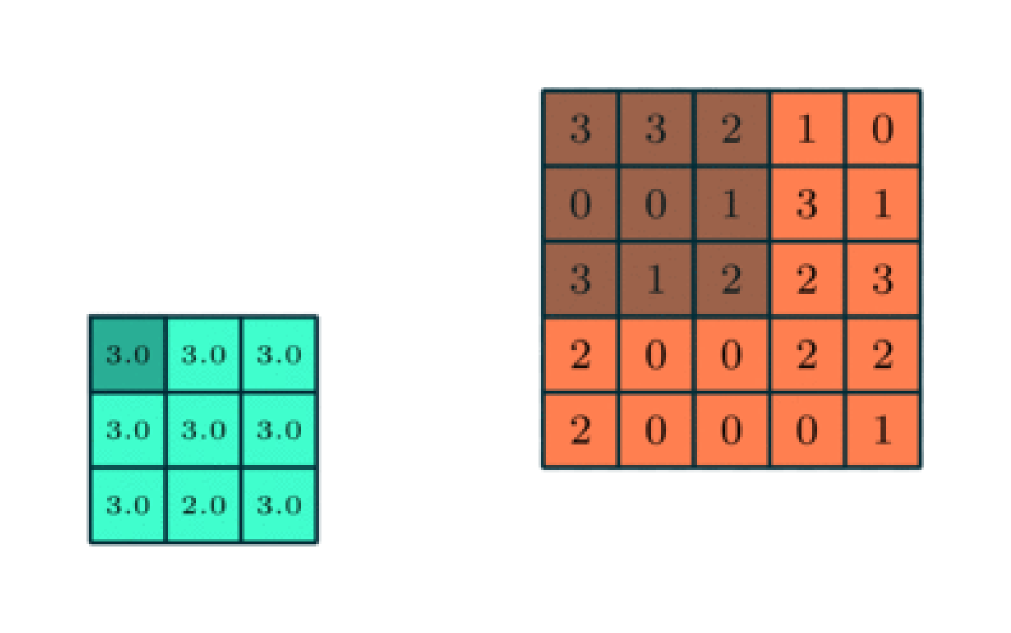
  f=kernel dimension

- **Max Pooling layer** is responsible for reducing the spatial size of the Convolved Feature.

- This is to **decrease the computational power required to process the data** through dimensionality reduction.

- It is useful for **extracting dominant features** which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

- **Max Pooling** returns the **maximum value** from the portion of the image covered by the Kernel.



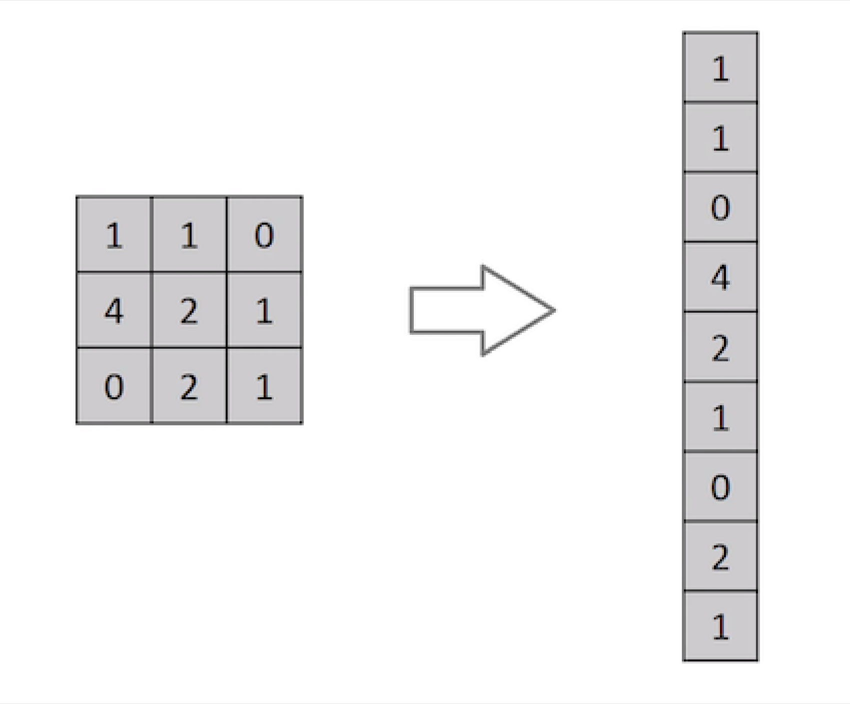
3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

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

❑ Flattening layer is obtained by reducing the dimension of max pooling layer into single feature vector i.e 1 dimensional form.

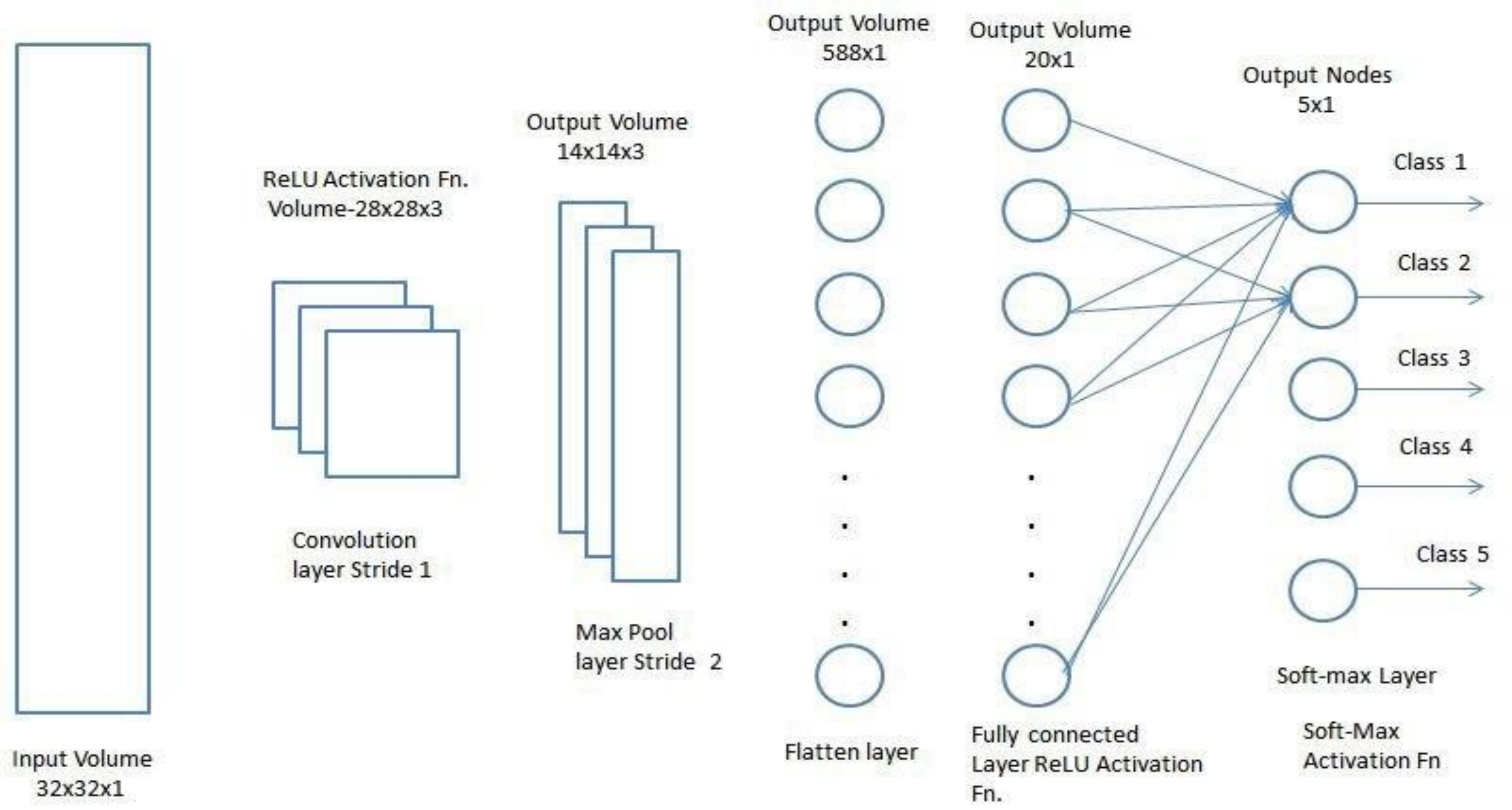
❑ The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights.

❑ After flattening,single feature vector is transferred to the dense layer.



$$\begin{bmatrix}1&1&0\\4&2&1\\0&2&1\end{bmatrix} \Rightarrow \begin{bmatrix}1\\1\\0\\4\\2\\1\\0\\2\\1\end{bmatrix}$$

Flattening of a 3x3 image matrix into a 9x1 vector

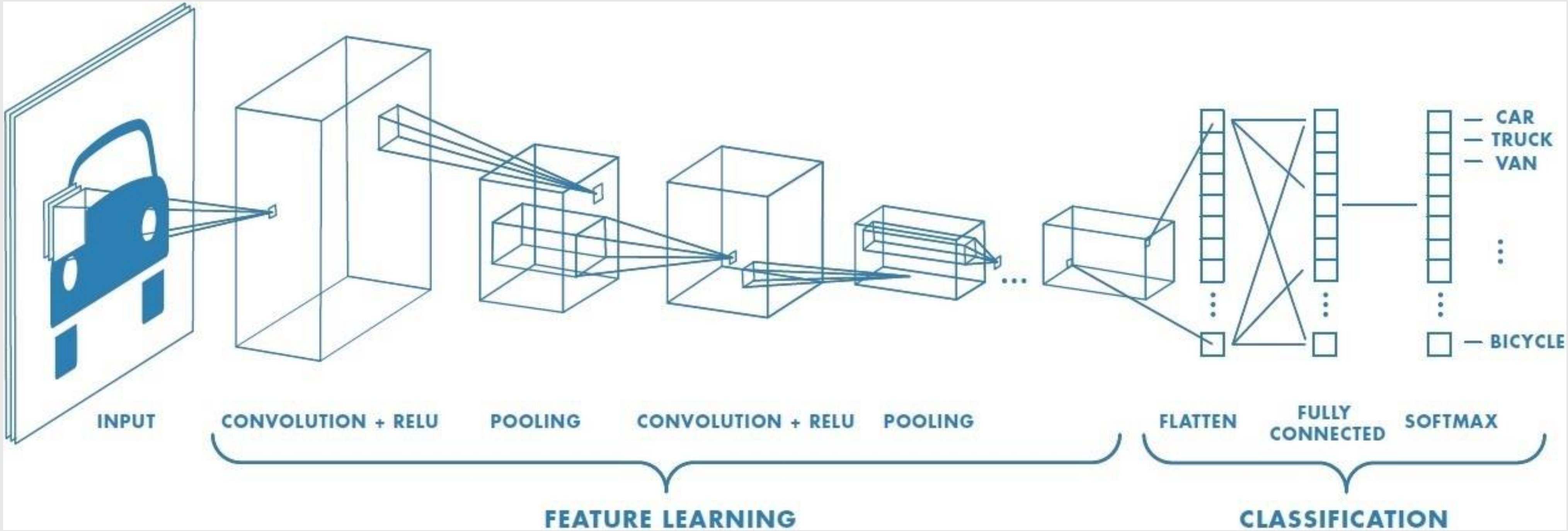


- ❑ Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer.

- ❑ The Fully-Connected layer is learning a possibly non-linear function in that space

❑ There are various architectures of CNNs available which have been key in building algorithms which power and shall power AI as a whole in the foreseeable future. Some of them have been listed below:

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



VGG 19 ARCHITECTURE

Application:

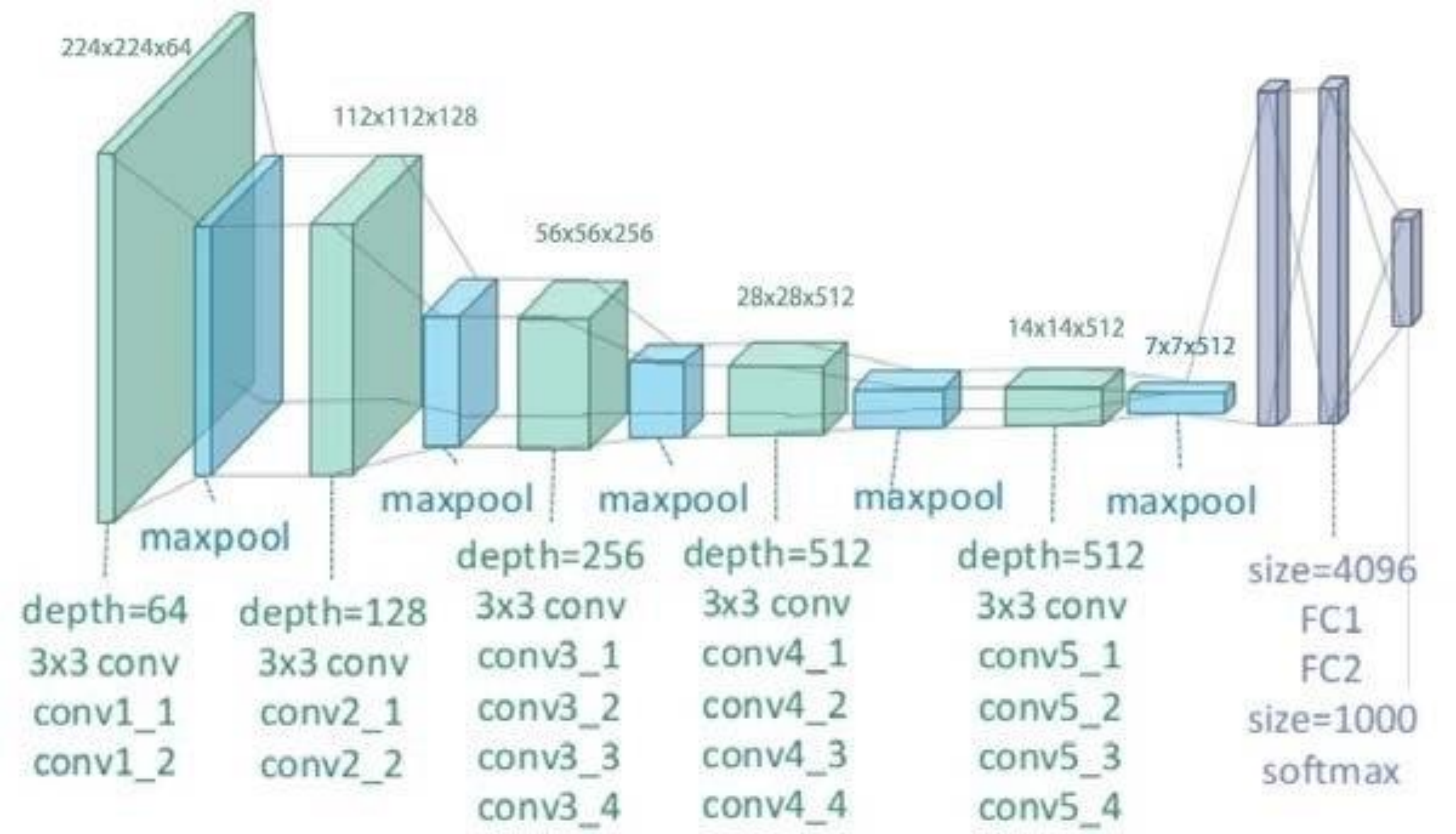
- Given image → find object name in the image
- It can detect any one of 1000 images
- It takes input image of size 224 * 224 * 3 (RGB image)

Built using:

- Convolutions layers (used only 3*3 size)
- Max pooling layers (used only 2*2 size)
- Fully connected layers at end
- Total 19 layers

VGG is a deep CNN used to classify images. The layers in VGG19 model are as follows:

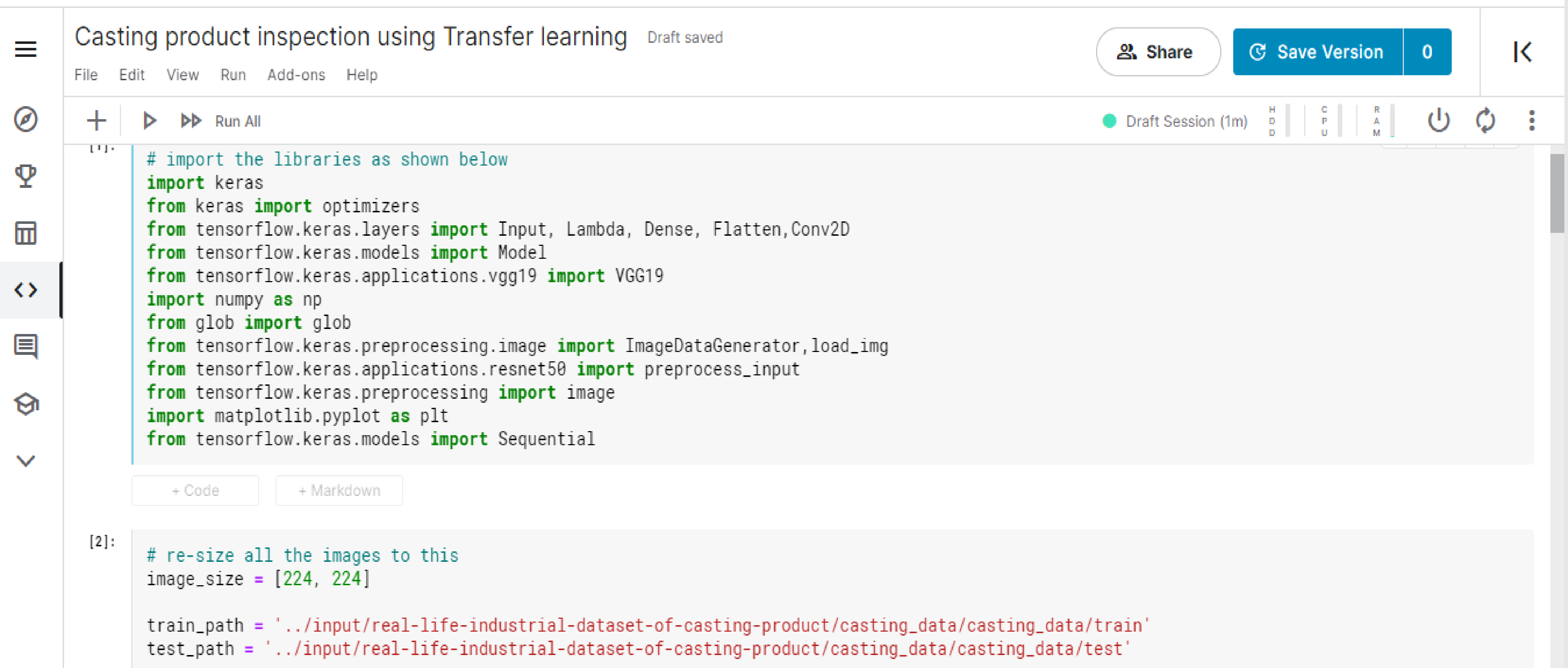
- Conv3x3 (64)
- Conv3x3 (64)
- MaxPool
- Conv3x3 (128)
- Conv3x3 (128)
- MaxPool
- Conv3x3 (256)
- Conv3x3 (256)
- Conv3x3 (256)
- Conv3x3 (256)
- MaxPool
- Conv3x3 (512)
- Conv3x3 (512)
- Conv3x3 (512)
- Conv3x3 (512)
- MaxPool
- Conv3x3 (512)
- Conv3x3 (512)
- Conv3x3 (512)
- Conv3x3 (512)
- MaxPool
- Fully Connected (4096)
- Fully Connected (4096)
- Fully Connected (1000)
- SoftMax



Architecture

- A fixed size of (224 * 224) RGB image was given as input to this network which means that the matrix was of shape (224,224,3).
- The only preprocessing that was done is that they subtracted the mean RGB value from each pixel, computed over the whole training set.
- Used kernels of (3 * 3) size with a stride size of 1 pixel, this enabled them to cover the whole notion of the image.
- spatial padding was used to preserve the spatial resolution of the image.
- max pooling was performed over a 2 * 2 pixel windows with sride 2.
- this was followed by Rectified linear unit(ReLu) to introduce non-linearity to make the model classify better and to improve computational time as the previous models used tanh or sigmoid functions this proved much better than those.
- implemented three fully connected layers from which first two were of size 4096 and after that a layer with 1000 channels for 1000-way *ILSVRC* classification and the final layer is a softmax function.

Transfer Learning Algorithm

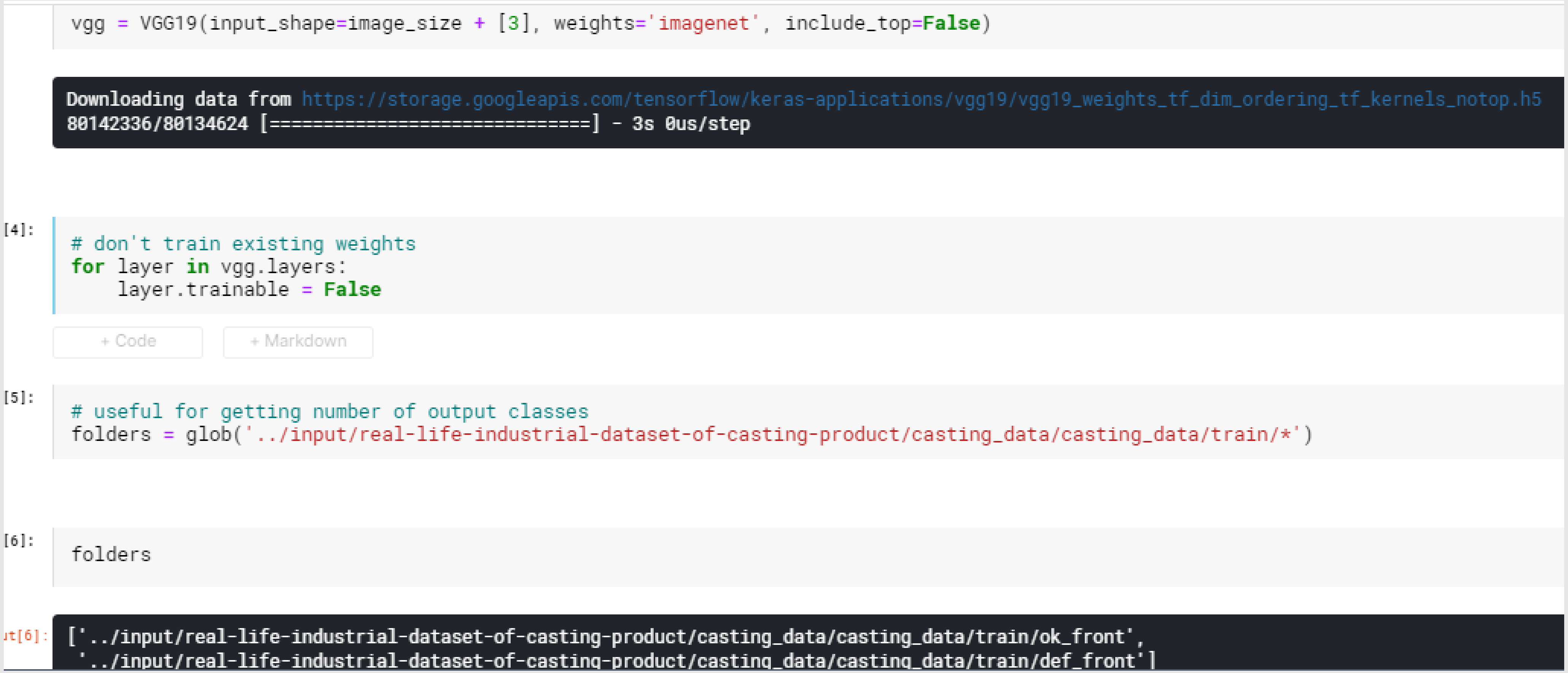

Casting product inspection using Transfer learning
Draft saved
File Edit View Run Add-ons Help
Share
Save Version 0
Run All
Draft Session (1m)
+ Code
+ Markdown

```
[1]:
# import the libraries as shown below
import keras
from keras import optimizers
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten,Conv2D
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg19 import VGG19
import numpy as np
from glob import glob
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
```

```
[2]:
# re-size all the images to this
image_size = [224, 224]

train_path = '../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/train'
test_path = '../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/test'
```

STEP-1:-

- ❑ IMPORTING REQUIRED LIBRARIES
- ❑ SETTING IMAGE SIZE
- ❑ SETTING TRAINING AND TEST PATH OF IMAGE DATASETS.



```
vgg = VGG19(input_shape=image_size + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 3s 0us/step
```

[4]:

```
# don't train existing weights
for layer in vgg.layers:
    layer.trainable = False
```

[5]:

```
# useful for getting number of output classes
folders = glob('../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/train/*')
```

[6]:

```
folders
```

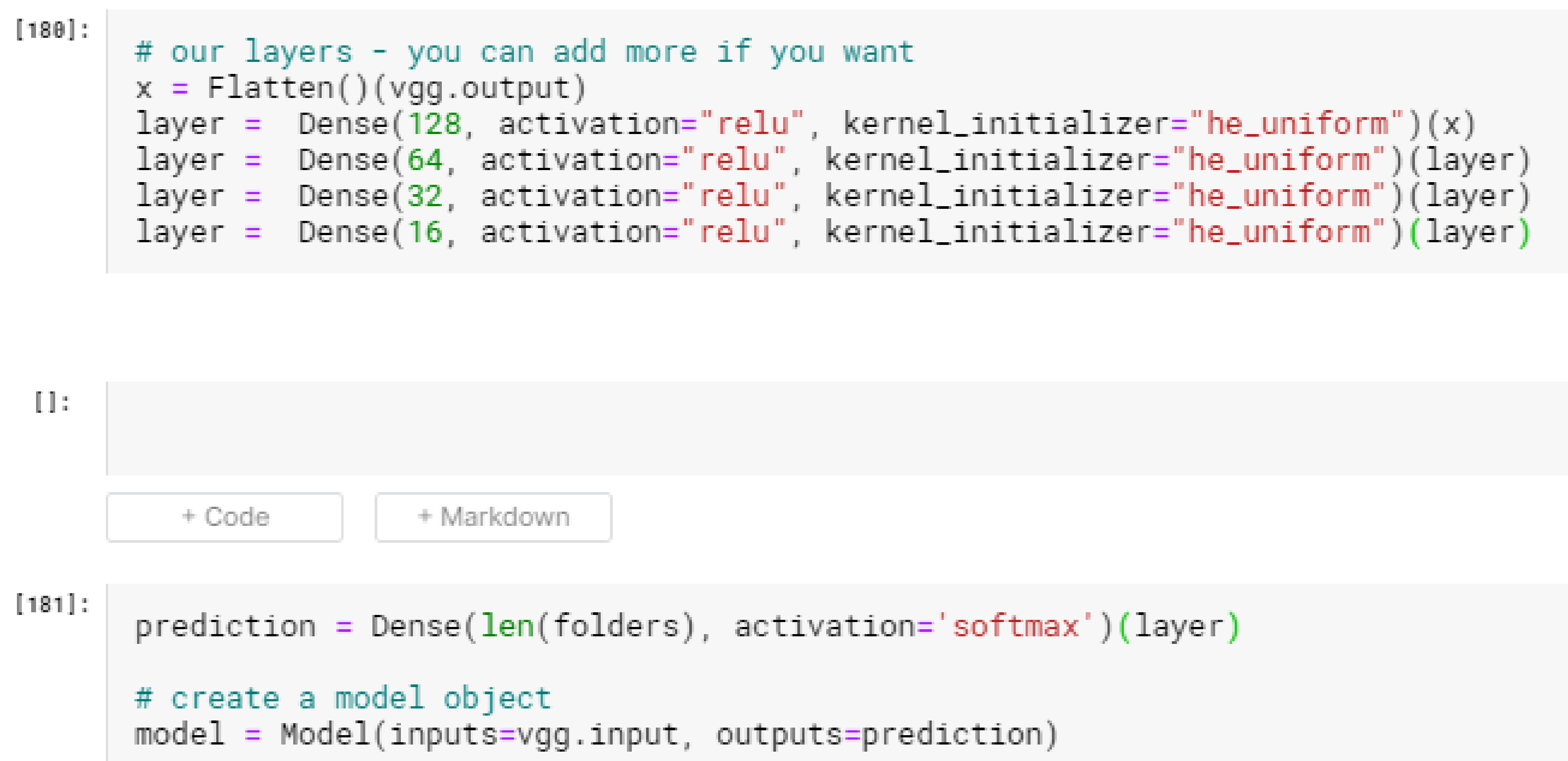
ıt[6]:

```
['../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/train/ok_front',
 '../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/train/def_front']
```

STEP-2:-

❑ CREATING AN OBJECT VGG OF CLASS **VGG19** AND ASSIGNING DESIRED PARAMETERS IN ORDER TO PERFORM TRANSFER LEARNING ALGORITHM.

❑ GETTING NUMBER OF OUTPUT CLASSES USING **GLOB** FUNCTION

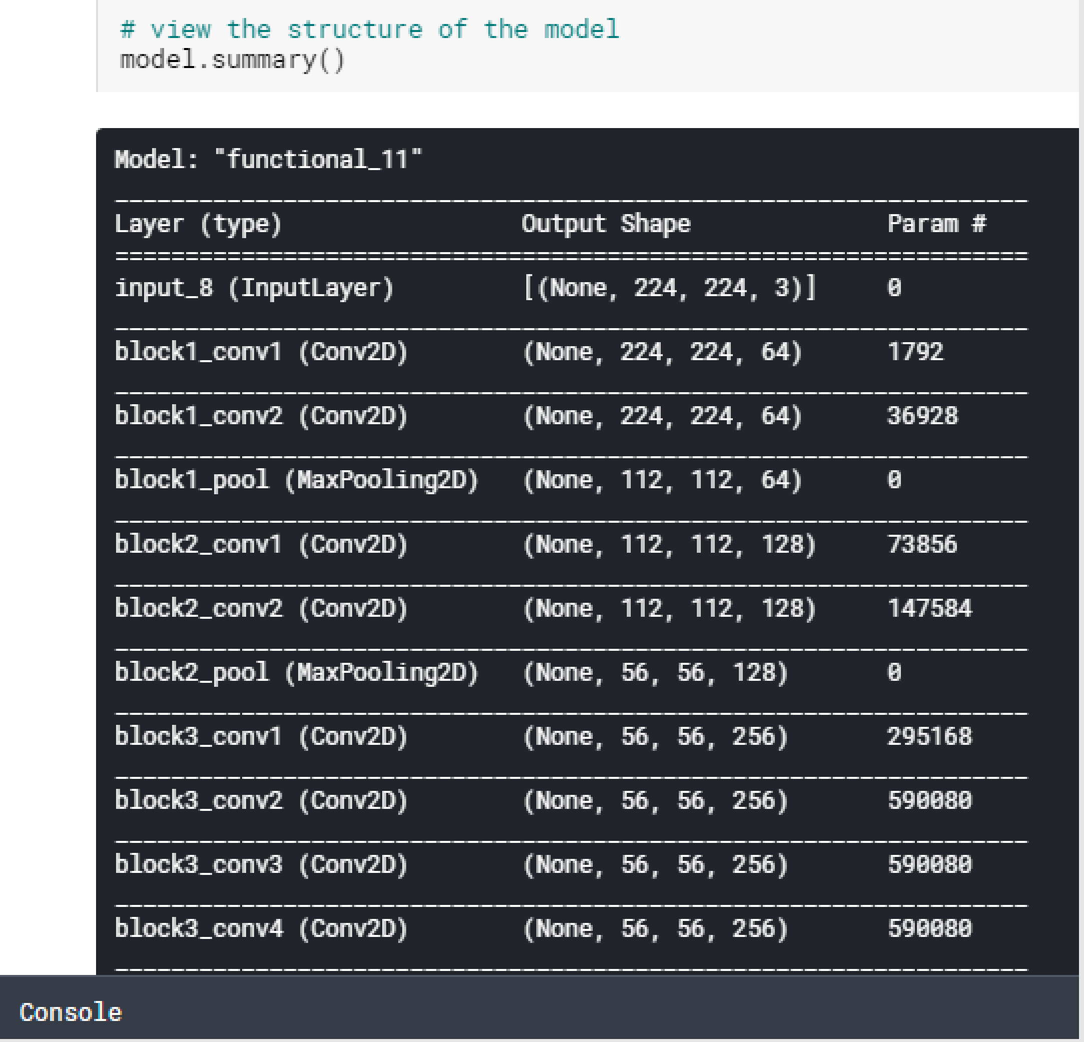


```
# our layers - you can add more if you want
x = Flatten()(vgg.output)
layer =  Dense(128, activation="relu", kernel_initializer="he_uniform")(x)
layer =  Dense(64, activation="relu", kernel_initializer="he_uniform")(layer)
layer =  Dense(32, activation="relu", kernel_initializer="he_uniform")(layer)
layer =  Dense(16, activation="relu", kernel_initializer="he_uniform")(layer)
```

```
prediction = Dense(len(folders), activation='softmax')(layer)

# create a model object
model = Model(inputs=vgg.input, outputs=prediction)
```

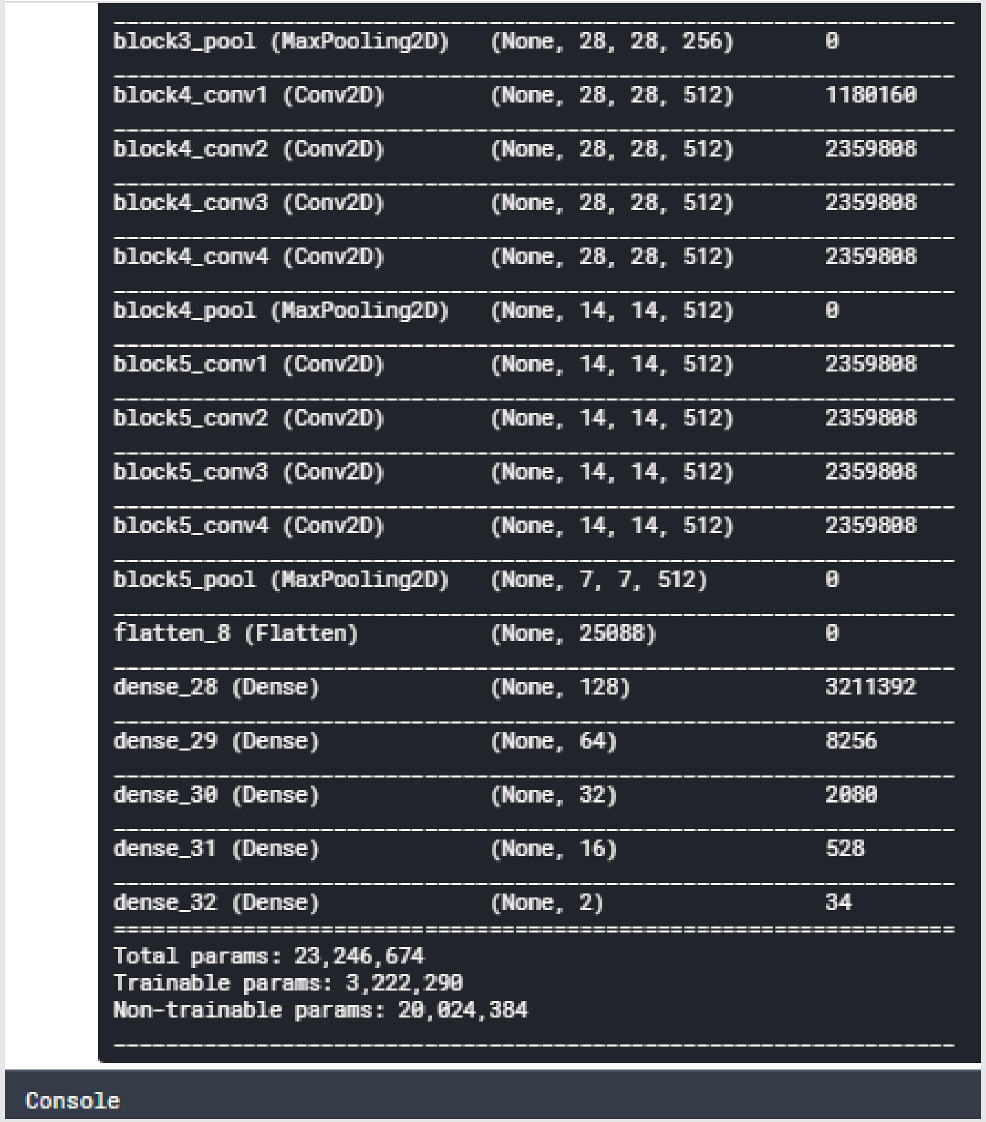
STEP-3:-

- ❑ FLATTENING USING MAX POOLING LAYER
- ❑ ADDING DENSE LAYER OF 128,64,32 AND 16 NEURONS AND INITIALISE WEIGHT USING KERNEL INITIALIZER
- ❑ APPLYING SOFTMAX ACTIVATION FUNCTION ON THE OUTPUT LAYER
- ❑ CREATE A MODEL USING MODEL CLASS.



```
# view the structure of the model
model.summary()
```

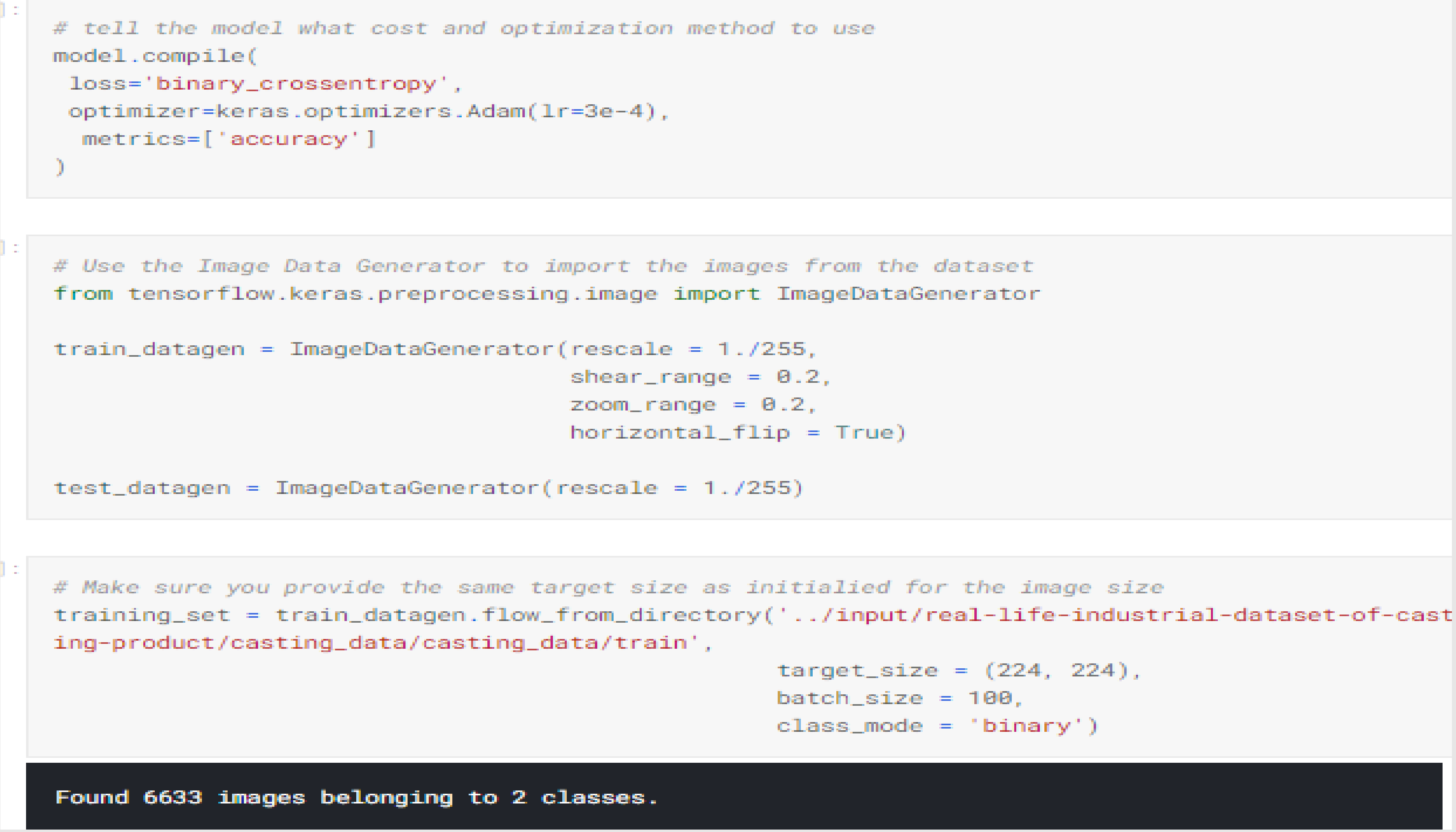
```
Model: "functional_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_8 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
```



```
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_8 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_28 (Dense)             (None, 128)               3211392
_________________________________________________________________
dense_29 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_30 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_31 (Dense)             (None, 16)                528
_________________________________________________________________
dense_32 (Dense)             (None, 2)                 34
=================================================================
Total params: 23,246,674
Trainable params: 3,222,290
Non-trainable params: 20,024,384
_________________________________________________________________
```

STEP-4

- ❑ GETTING SUMMARY OF THE MODEL
- ❑ TOTAL PARAMETERS(TRAINABLE AND NON-TRAINABLE)CAN BE ANALYSED USING IT.
- ❑ FUNCTIONING OF THE MODEL CAN BE ANALYSED BY OBSERVING DIFFERENT LAYERS LIKE CONVOLUTIONAL,MAX-POOLING etc.



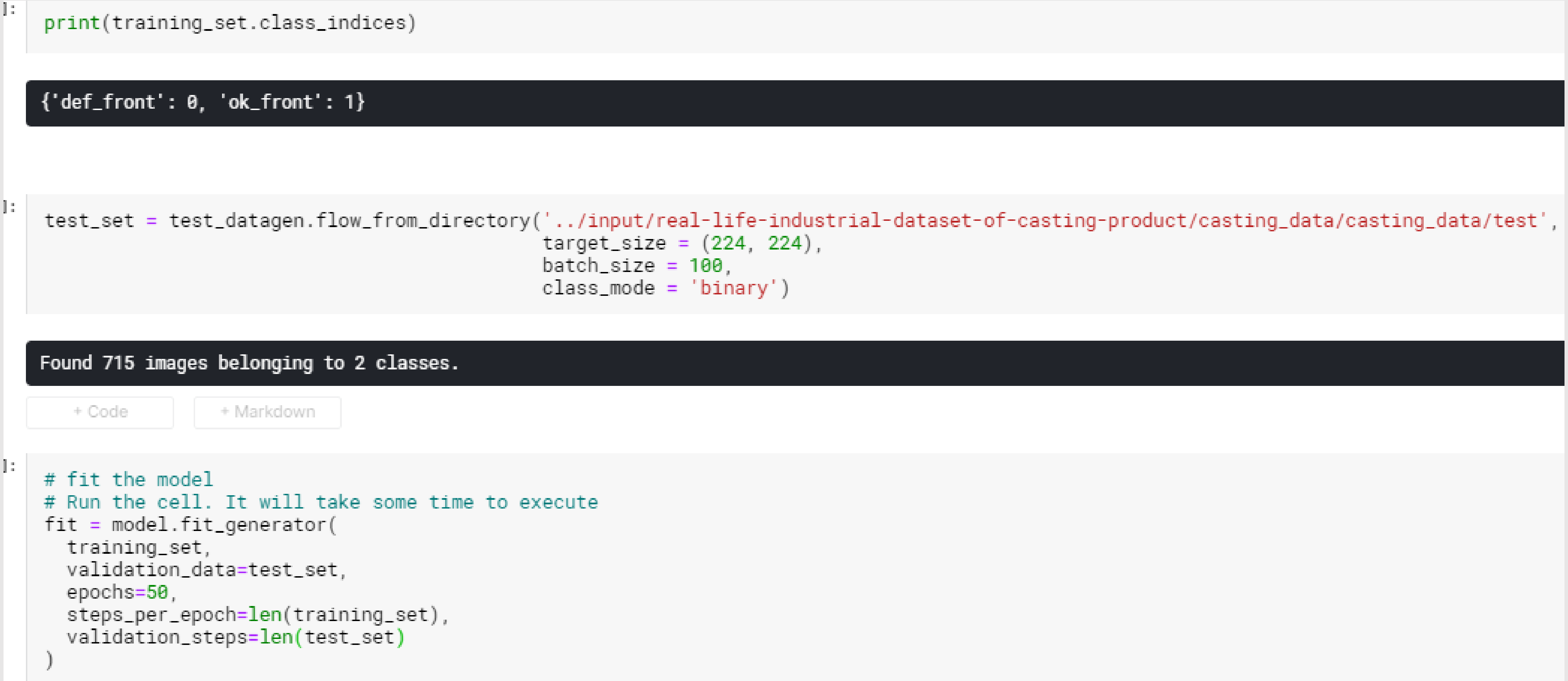
```
# tell the model what cost and optimization method to use
model.compile(
 loss='binary_crossentropy',
 optimizer=keras.optimizers.Adam(lr=3e-4),
  metrics=['accuracy']
)
```

```
# Use the Image Data Generator to import the images from the dataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

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

```
# Make sure you provide the same target size as initialied for the image size
training_set = train_datagen.flow_from_directory('../input/real-life-industrial-dataset-of-cast
ing-product/casting_data/casting_data/train',
                                                 target_size = (224, 224),
                                                 batch_size = 100,
                                                 class_mode = 'binary')
```

```
Found 6633 images belonging to 2 classes.
```

- ❑ ASSIGNING LOSS FUNCTION AND OPTIMISER FUNCTION WHILE PERFORMING BACKPROPAGATION
- ❑ ASSIGNING PARAMETERS FOR PERFORMING DATA AUGMENTATION AND FURTHER APPLYING IT ON THE TRAINING DATASET.



```
print(training_set.class_indices)
```

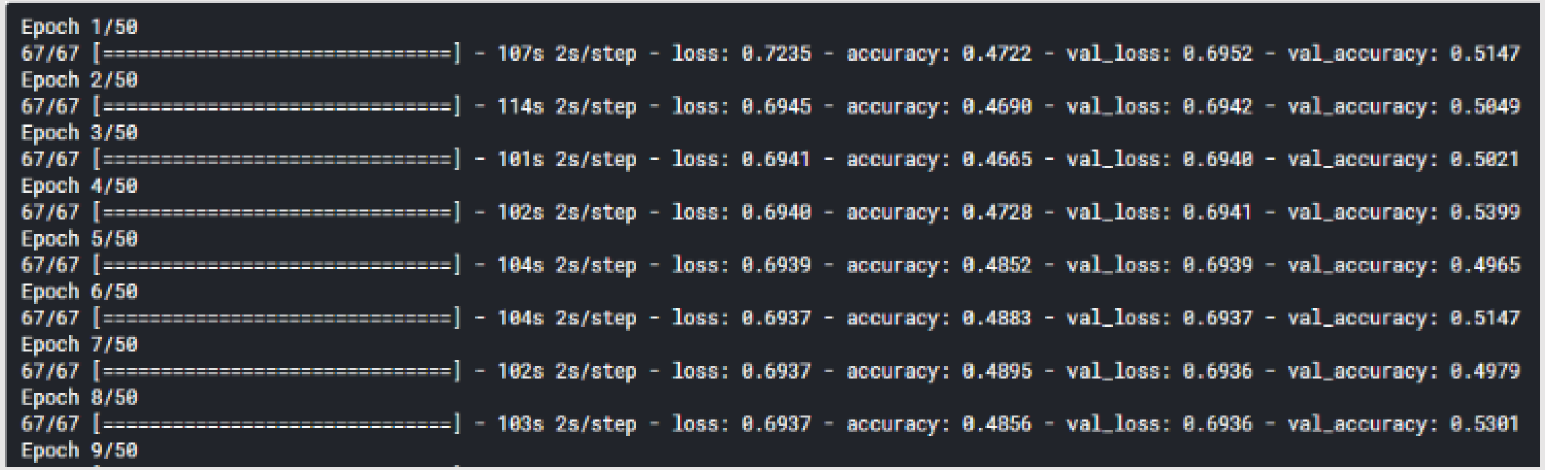
```
{'def_front': 0, 'ok_front': 1}
```

```
test_set = test_datagen.flow_from_directory('../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/test',
                                            target_size = (224, 224),
                                            batch_size = 100,
                                            class_mode = 'binary')
```

```
Found 715 images belonging to 2 classes.
```

```
# fit the model
# Run the cell. It will take some time to execute
fit = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=50,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

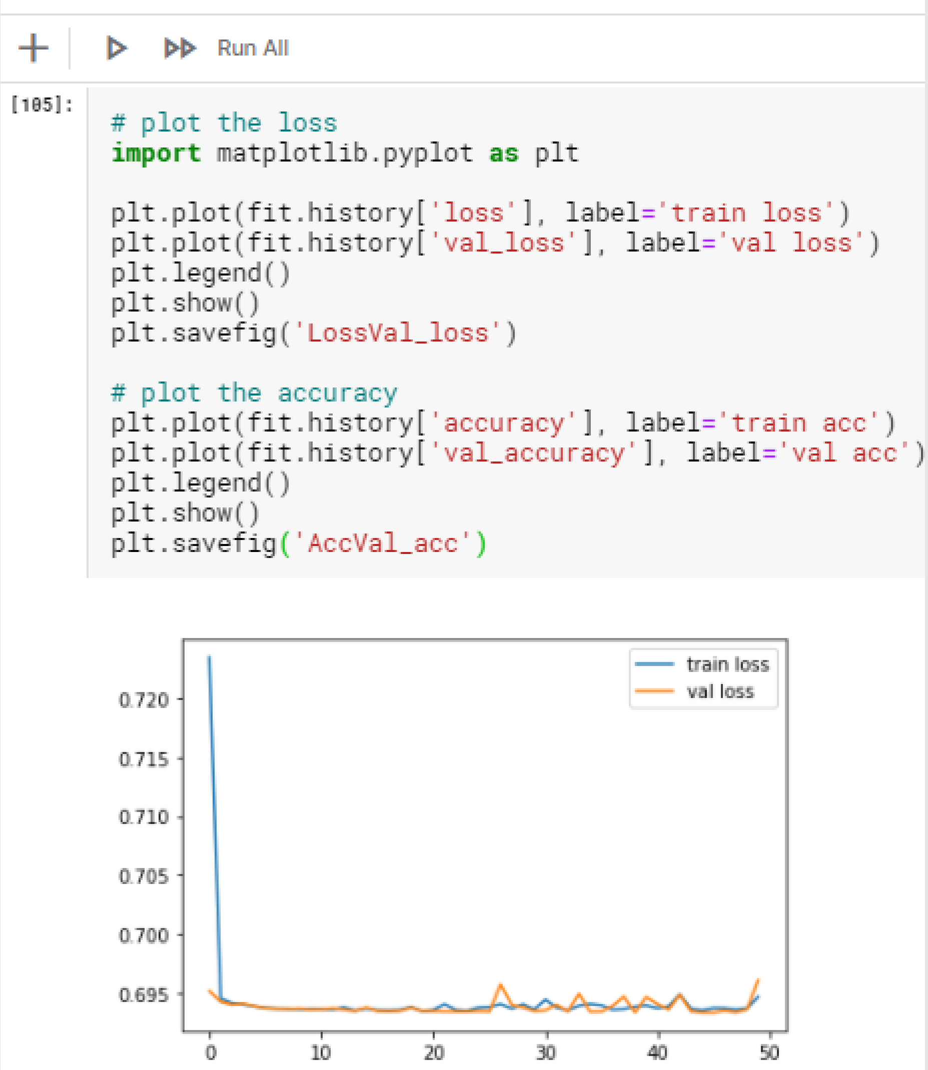
- ❑ ASSIGNING PARAMETERS TO THE TEST DATASET LIKE TARGET_SIZE,BATCH_SIZE etc.
- ❑ FITTING THE MODEL WITH EPOCHS=50



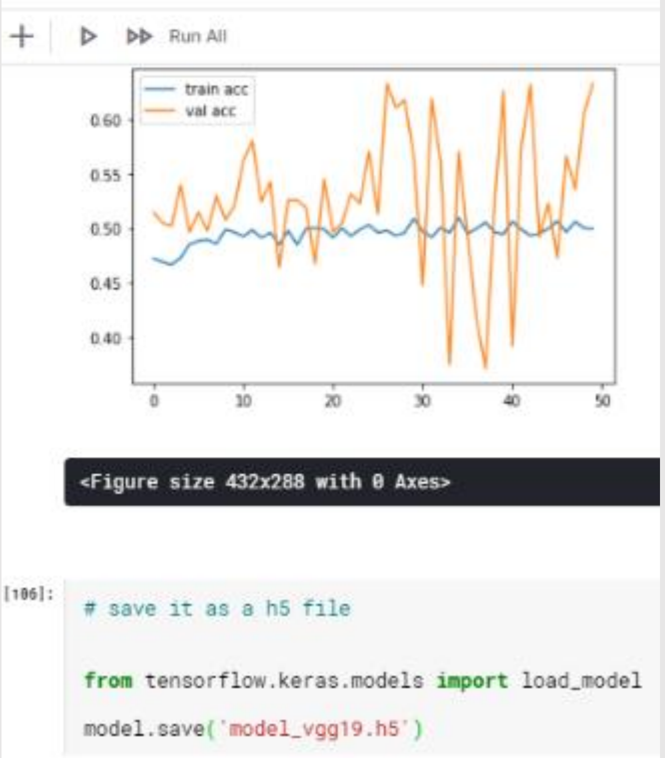
```
Epoch 1/50
67/67 [==============================] - 107s 2s/step - loss: 0.7235 - accuracy: 0.4722 - val_loss: 0.6952 - val_accuracy: 0.5147
Epoch 2/50
67/67 [==============================] - 114s 2s/step - loss: 0.6945 - accuracy: 0.4690 - val_loss: 0.6942 - val_accuracy: 0.5049
Epoch 3/50
67/67 [==============================] - 101s 2s/step - loss: 0.6941 - accuracy: 0.4665 - val_loss: 0.6940 - val_accuracy: 0.5021
Epoch 4/50
67/67 [==============================] - 102s 2s/step - loss: 0.6940 - accuracy: 0.4728 - val_loss: 0.6941 - val_accuracy: 0.5399
Epoch 5/50
67/67 [==============================] - 104s 2s/step - loss: 0.6939 - accuracy: 0.4852 - val_loss: 0.6939 - val_accuracy: 0.4965
Epoch 6/50
67/67 [==============================] - 104s 2s/step - loss: 0.6937 - accuracy: 0.4883 - val_loss: 0.6937 - val_accuracy: 0.5147
Epoch 7/50
67/67 [==============================] - 102s 2s/step - loss: 0.6937 - accuracy: 0.4895 - val_loss: 0.6936 - val_accuracy: 0.4979
Epoch 8/50
67/67 [==============================] - 103s 2s/step - loss: 0.6937 - accuracy: 0.4856 - val_loss: 0.6936 - val_accuracy: 0.5301
Epoch 9/50
```

❑ FORWARD AND BACKWARD PROPAGATION ARE PERFORMED IN ORDER TO UPDATE THE WEIGHTS.

❑ FURTHER UPDATION OF WEIGHTS WILL REDUCE THE GAP BETWEEN PREDICTED AND ACTUAL OUTPUT.



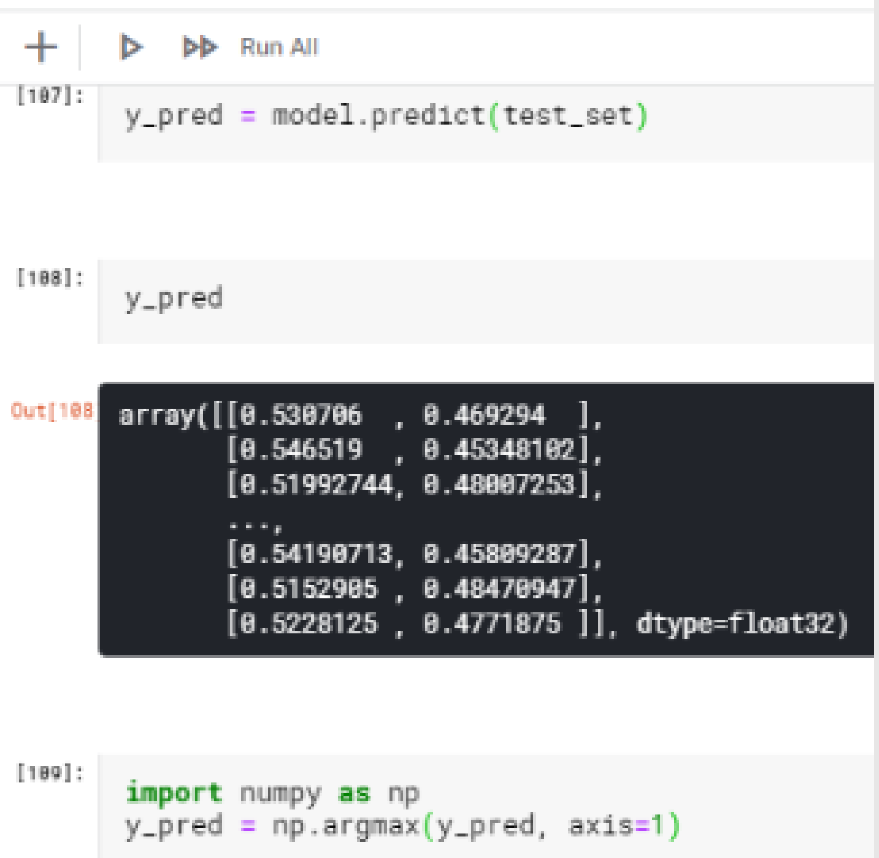
```
# plot the loss
import matplotlib.pyplot as plt

plt.plot(fit.history['loss'], label='train loss')
plt.plot(fit.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# plot the accuracy
plt.plot(fit.history['accuracy'], label='train acc')
plt.plot(fit.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```



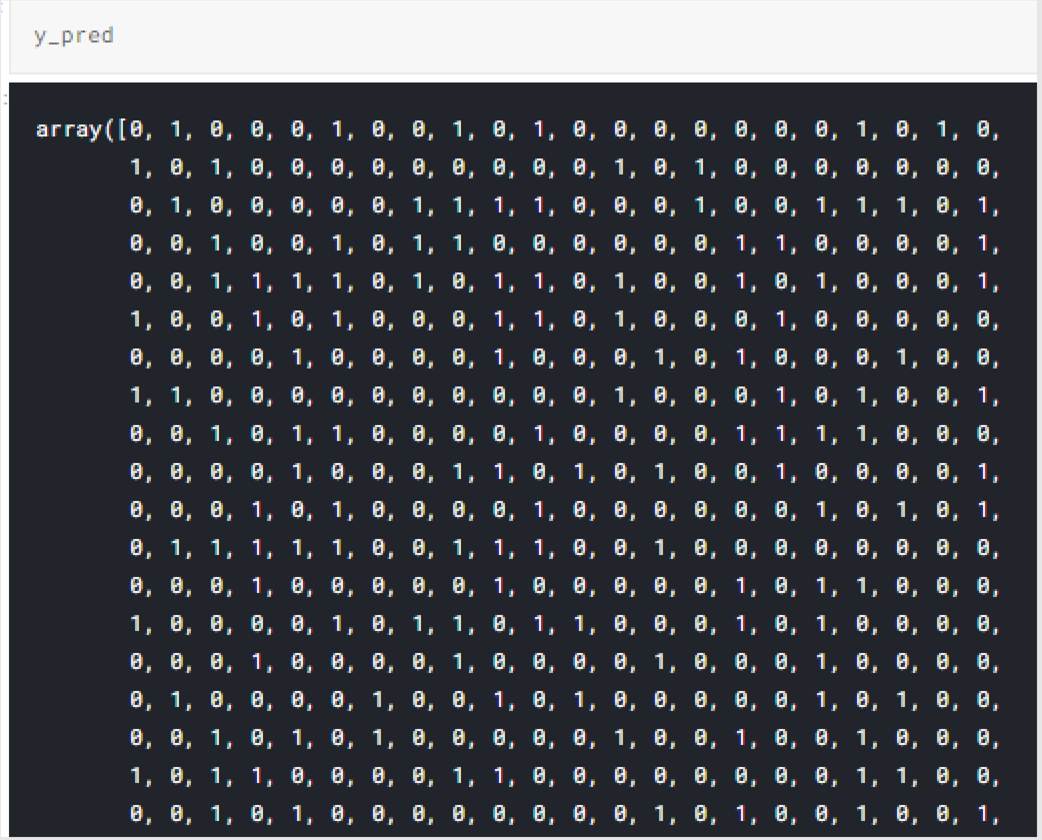
```
<Figure size 432x288 with 0 Axes>
```

```
# save it as a h5 file

from tensorflow.keras.models import load_model

model.save('model_vgg19.h5')
```

- ❑ FIRST FIGURE REPRESENTS THE VARIATION BETWEEN AND TRAINING AND TEST DATASETS IN TERMS OF LOSS.
- ❑ SECOND FIGURE REPRESENTS THE VARIATION BETWEEN AND TRAINING AND TEST DATASETS IN TERMS OF ACCURACY OBTAINED AFTER RUNNING EPOCHS.

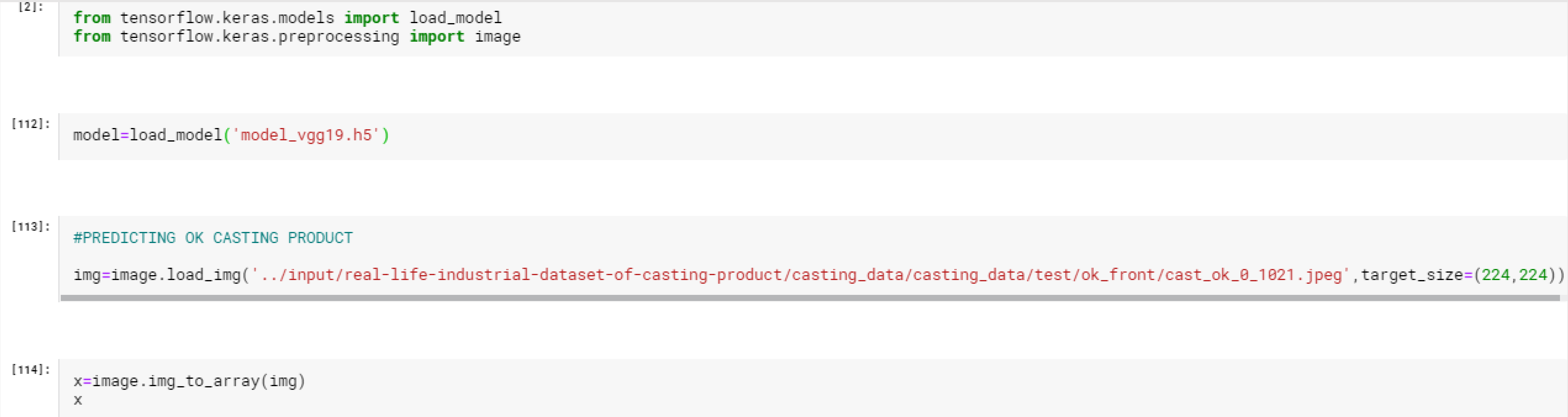


```
[107]: y_pred = model.predict(test_set)

[108]: y_pred

Out[108]: array([[0.530706  , 0.469294  ],
       [0.546519  , 0.45348102],
       [0.51992744, 0.48007253],
       ...,
       [0.54190713, 0.45809287],
       [0.5152905 , 0.48470947],
       [0.5228125 , 0.4771875 ]], dtype=float32)

[109]: import numpy as np
       y_pred = np.argmax(y_pred, axis=1)
```



```
y_pred

array([0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1,
```

❑ PREDICTING THE TEST DATASET IMAGES WHICH ARE REPRESENTED IN THE FORM OF ARRAY.



[2]:

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
```

[112]:

```
model=load_model('model_vgg19.h5')
```

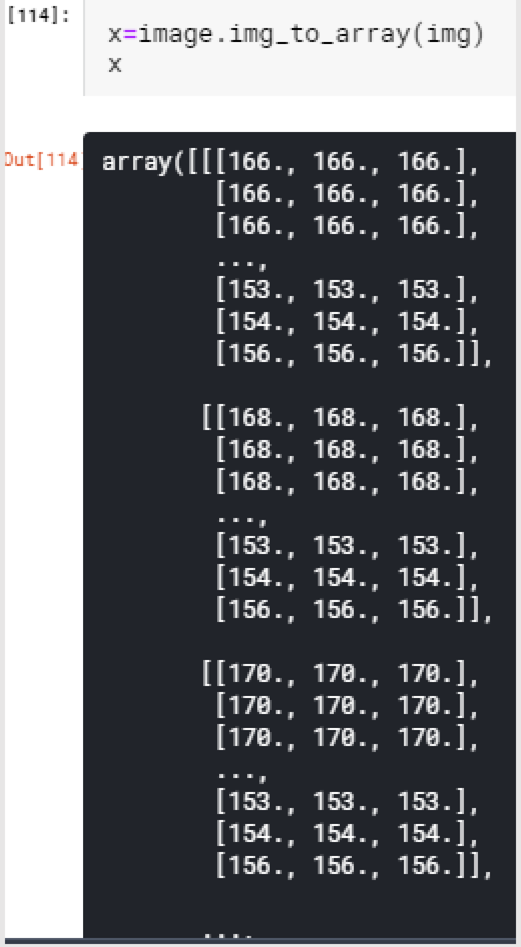
[113]:

```
#PREDICTING OK CASTING PRODUCT

img=image.load_img('../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/test/ok_front/cast_ok_0_1021.jpeg',target_size=(224,224))
```

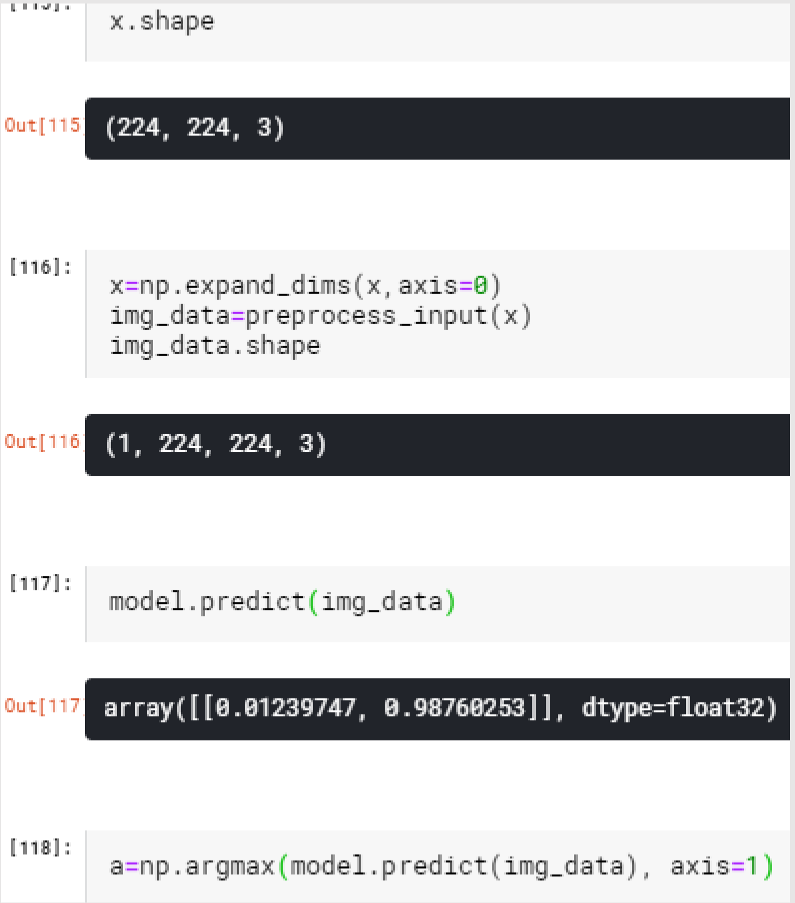
[114]:

```
x=image.img_to_array(img)
x
```

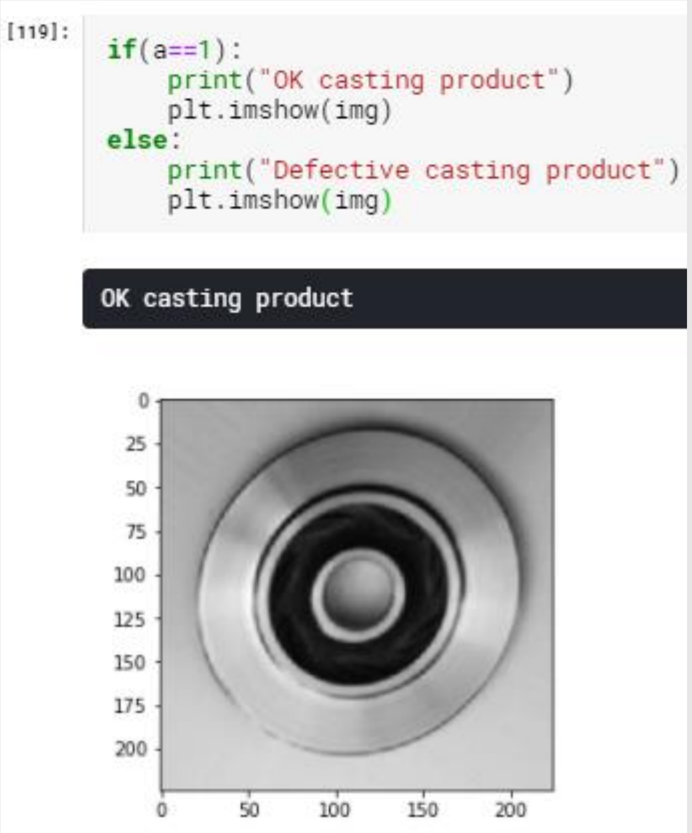
- ❑ SAVED FILE IS LAODED USING LOAD_MODEL CLASS
- ❑ INSERT THE PATH OF IMAGE AND LOAD IT
- ❑ CONVERION OF JPEG FORMAT TO ARRAY FORMAT.



```
[114]:  x=image.img_to_array(img)
        x

Out[114 array([[[166., 166., 166.],
                [166., 166., 166.],
                [166., 166., 166.],
                ...,
                [153., 153., 153.],
                [154., 154., 154.],
                [156., 156., 156.]],

               [[168., 168., 168.],
                [168., 168., 168.],
                [168., 168., 168.],
                ...,
                [153., 153., 153.],
                [154., 154., 154.],
                [156., 156., 156.]],

               [[170., 170., 170.],
                [170., 170., 170.],
                [170., 170., 170.],
                ...,
                [153., 153., 153.],
                [154., 154., 154.],
                [156., 156., 156.]],

               ...
```

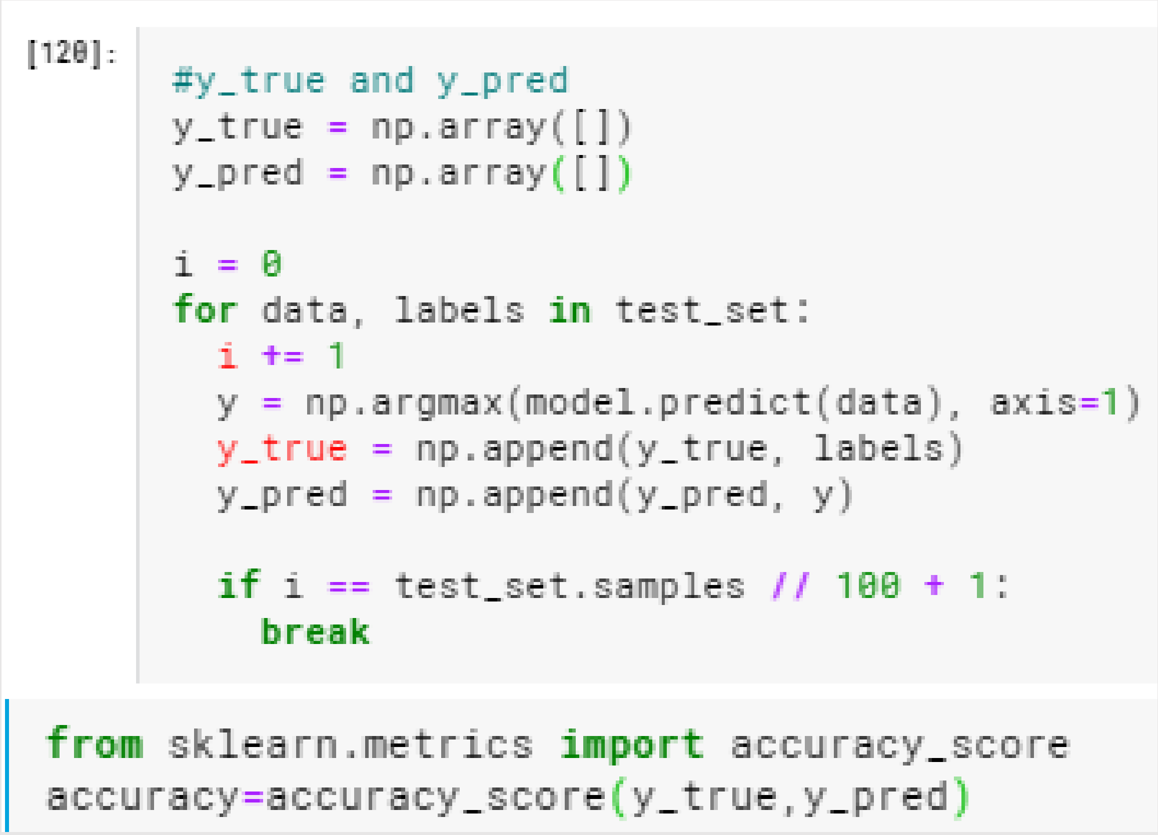


```
        x.shape

Out[115 (224, 224, 3)

[116]:  x=np.expand_dims(x,axis=0)
        img_data=preprocess_input(x)
        img_data.shape

Out[116 (1, 224, 224, 3)

[117]:  model.predict(img_data)

Out[117 array([[0.01239747, 0.98760253]], dtype=float32)

[118]:  a=np.argmax(model.predict(img_data), axis=1)
```



```
[119]:  if(a==1):
            print("OK casting product")
            plt.imshow(img)
        else:
            print("Defective casting product")
            plt.imshow(img)

        OK casting product
```

- ❑ EXPANDING THE DIMENSION OF ARRAY AND PREPROCESSING IT
- ❑ FINALLY PREDICT THE IMAGE USING TRAINED MODEL.
- ❑ RETURN THE INDEX OF MAXIMUM VALUE OF ARRAY USING ARGMAX FUNCTION.
- ❑ IN THIS CASE,OUTPUT COMES TO BE “a=1” AND HENCE THE IMAGE IS“OK CASTING PRODUCT”.



```
[120]:
#y_true and y_pred
y_true = np.array([])
y_pred = np.array([])

i = 0
for data, labels in test_set:
  i += 1
  y = np.argmax(model.predict(data), axis=1)
  y_true = np.append(y_true, labels)
  y_pred = np.append(y_pred, y)

  if i == test_set.samples // 100 + 1:
    break
```

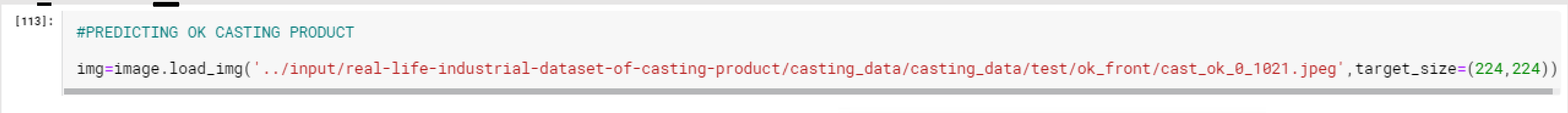
```
from sklearn.metrics import accuracy_score
accuracy=accuracy_score(y_true,y_pred)
```

❑ THE OVERALL ACCURACY OF THE MODEL IS APPROXIMATELY 54.12%.

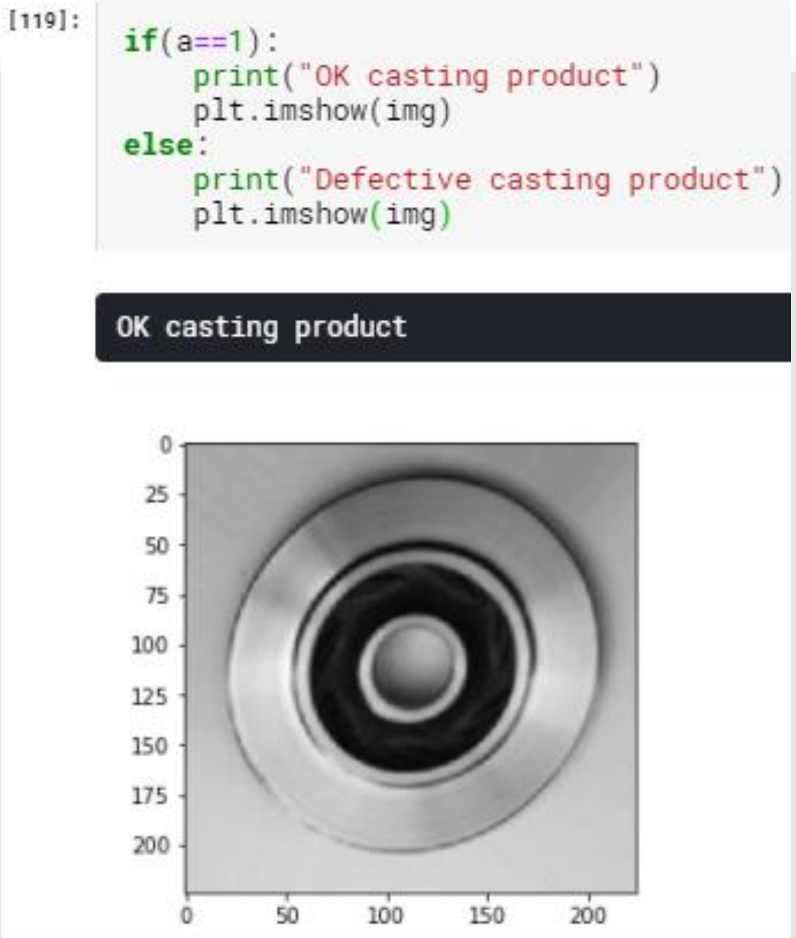
PREDICTING OUTPUT OF CASTING PRODUCTS

CORRECT_PREDICTIONS:

-TEST_1:-



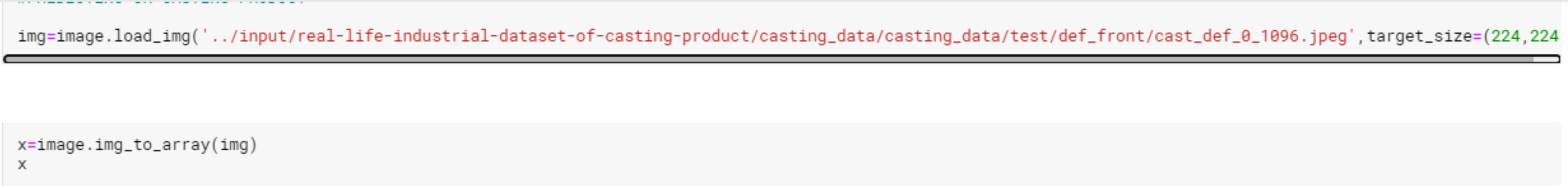
[113]:
```
#PREDICTING OK CASTING PRODUCT

img=image.load_img('../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/test/ok_front/cast_ok_0_1021.jpeg',target_size=(224,224))
```



[119]:
```
if(a==1):
    print("OK casting product")
    plt.imshow(img)
else:
    print("Defective casting product")
    plt.imshow(img)
```

```
OK casting product
```

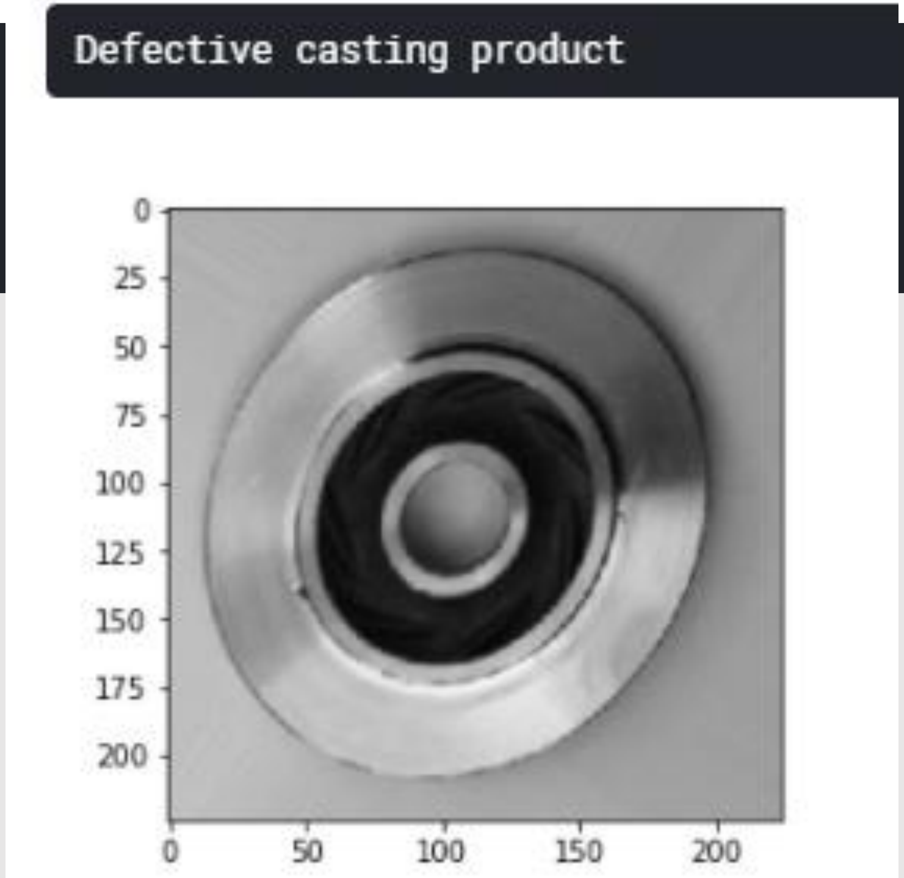
TEST_2



```
img=image.load_img('../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/test/def_front/cast_def_0_1096.jpeg',target_size=(224,224
```

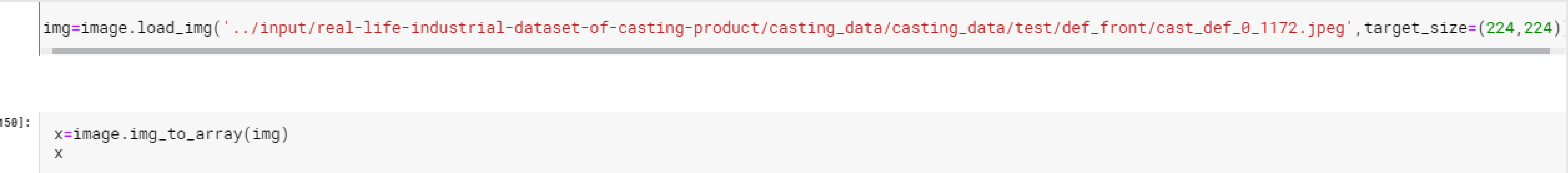
```
x=image.img_to_array(img)
x
```

```
array([[[170., 170., 170.],
        [174., 174., 174.],
        [177., 177., 177.],
        ...,
        [151., 151., 151.],
        [151., 151., 151.],
        [150., 150., 150.]],
```



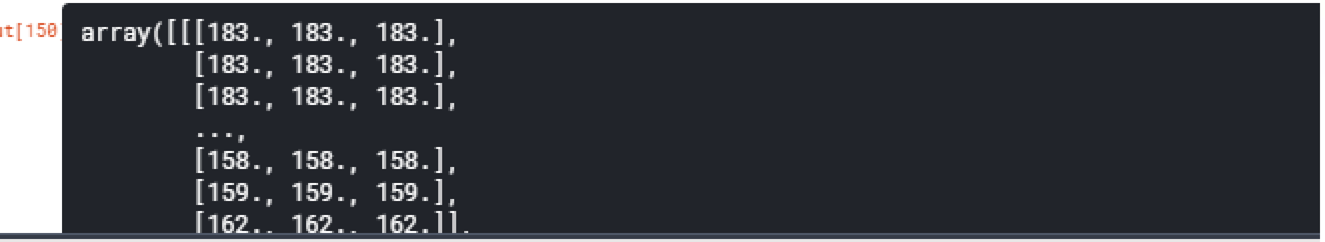
Defective casting product

TEST_3

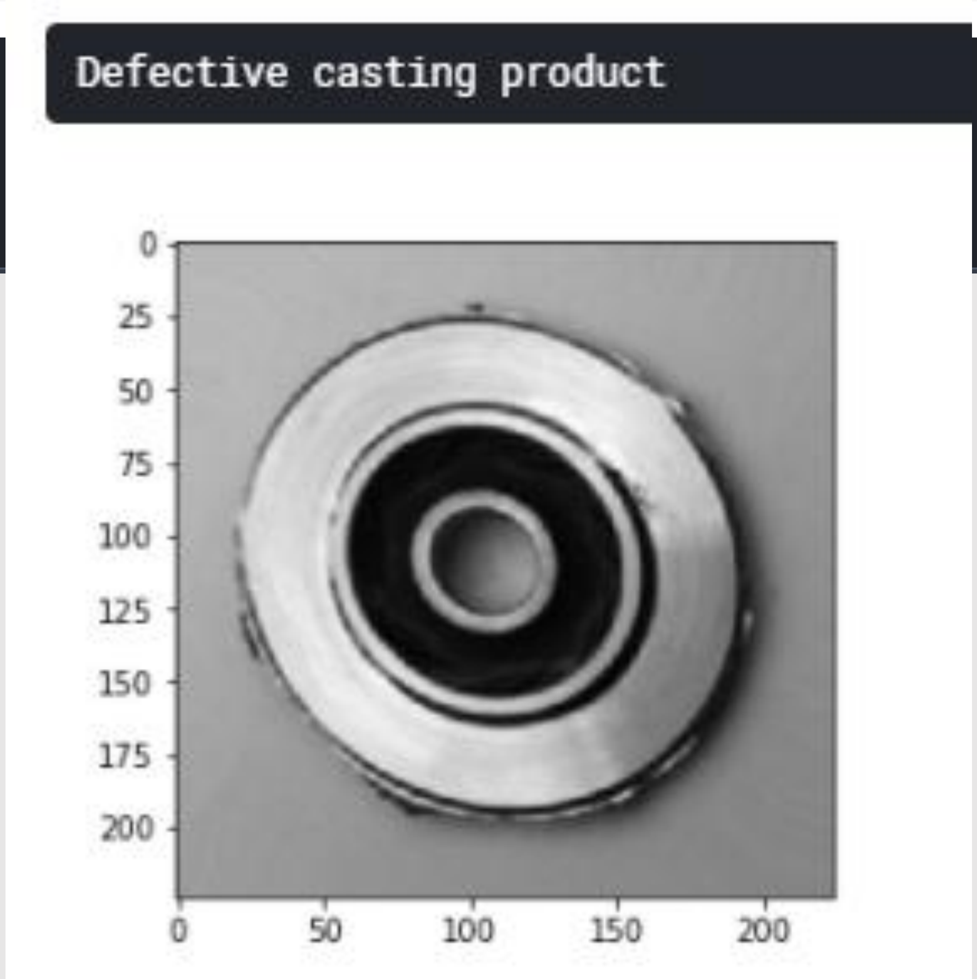


```
img=image.load_img('../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/test/def_front/cast_def_0_1172.jpeg',target_size=(224,224)
```

```
x=image.img_to_array(img)
x
```

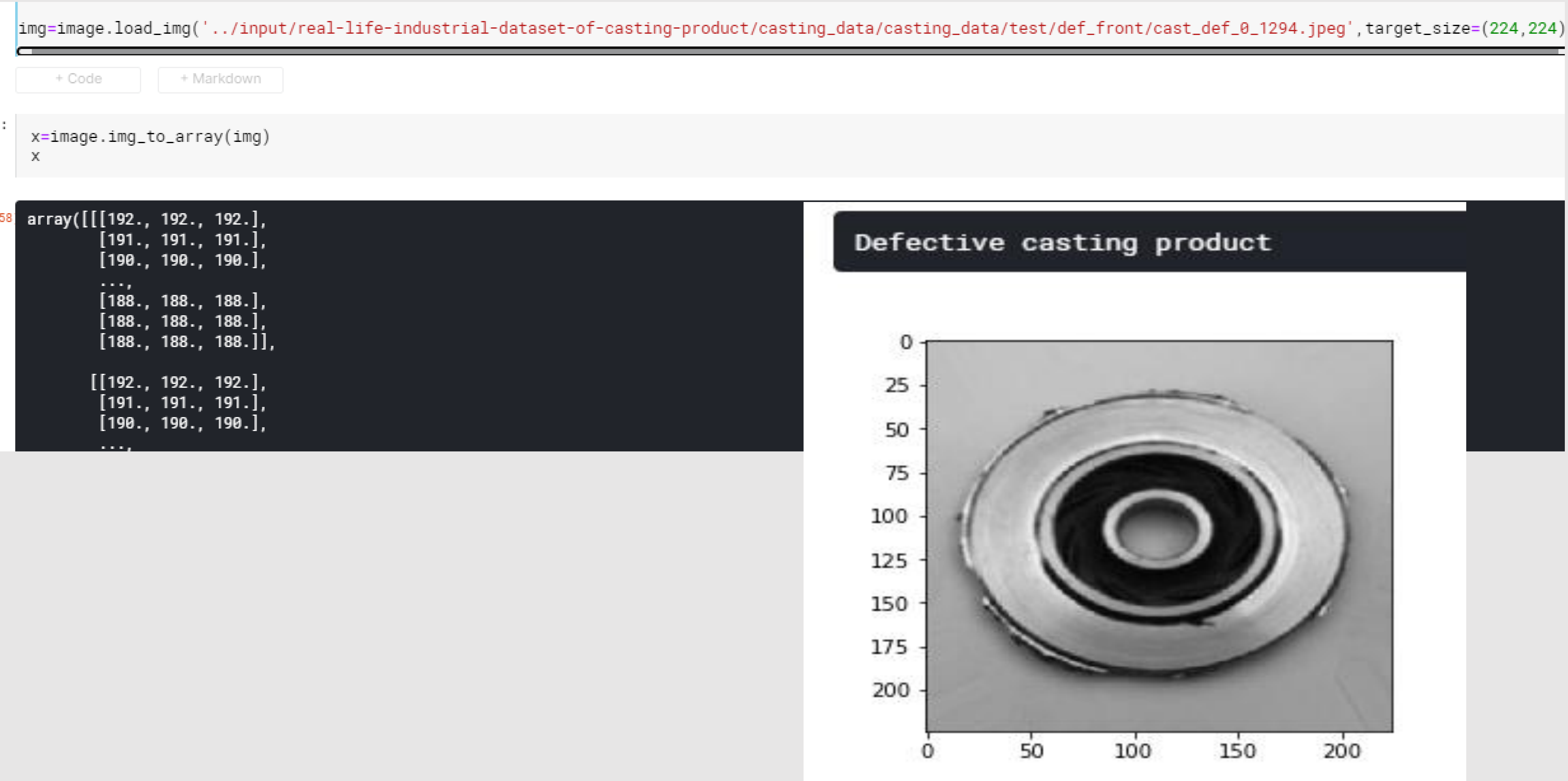


```
array([[[183., 183., 183.],
        [183., 183., 183.],
        [183., 183., 183.],
        ...,
        [158., 158., 158.],
        [159., 159., 159.],
        [162., 162., 162.]],
```



Defective casting product

TEST_4



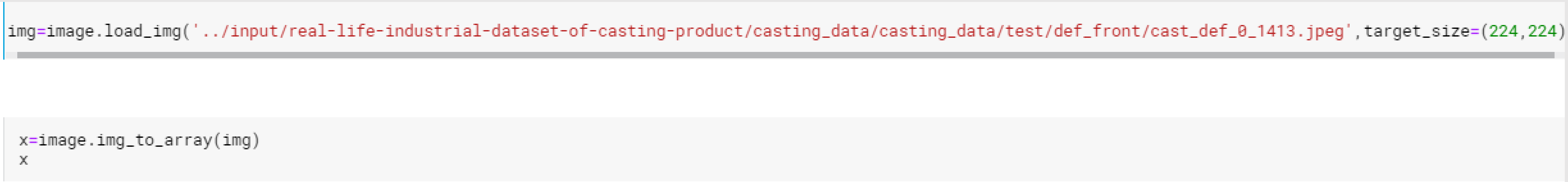
```
img=image.load_img('../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/test/def_front/cast_def_0_1294.jpeg',target_size=(224,224)
```

```
x=image.img_to_array(img)
x
```

```
array([[[192., 192., 192.],
        [191., 191., 191.],
        [190., 190., 190.],
        ...,
        [188., 188., 188.],
        [188., 188., 188.],
        [188., 188., 188.]],

       [[192., 192., 192.],
        [191., 191., 191.],
        [190., 190., 190.],
        ...,
```

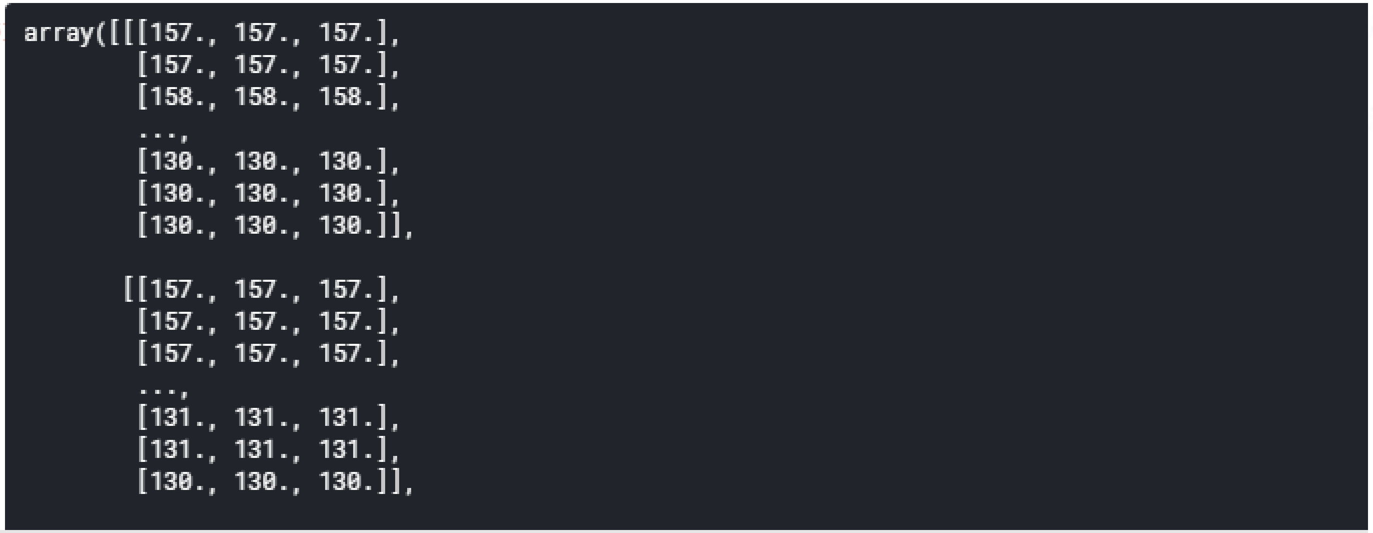
Defective casting product

TEST_5

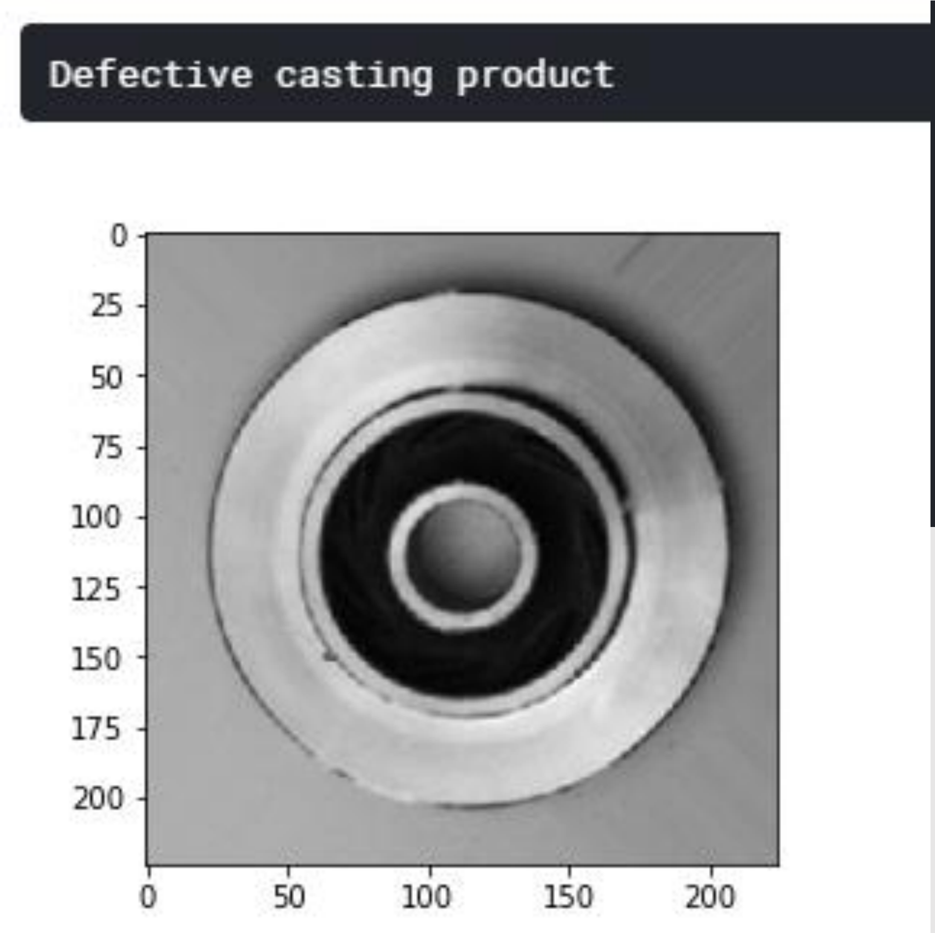


```
img=image.load_img('../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/test/def_front/cast_def_0_1413.jpeg',target_size=(224,224)
```

```
x=image.img_to_array(img)
x
```

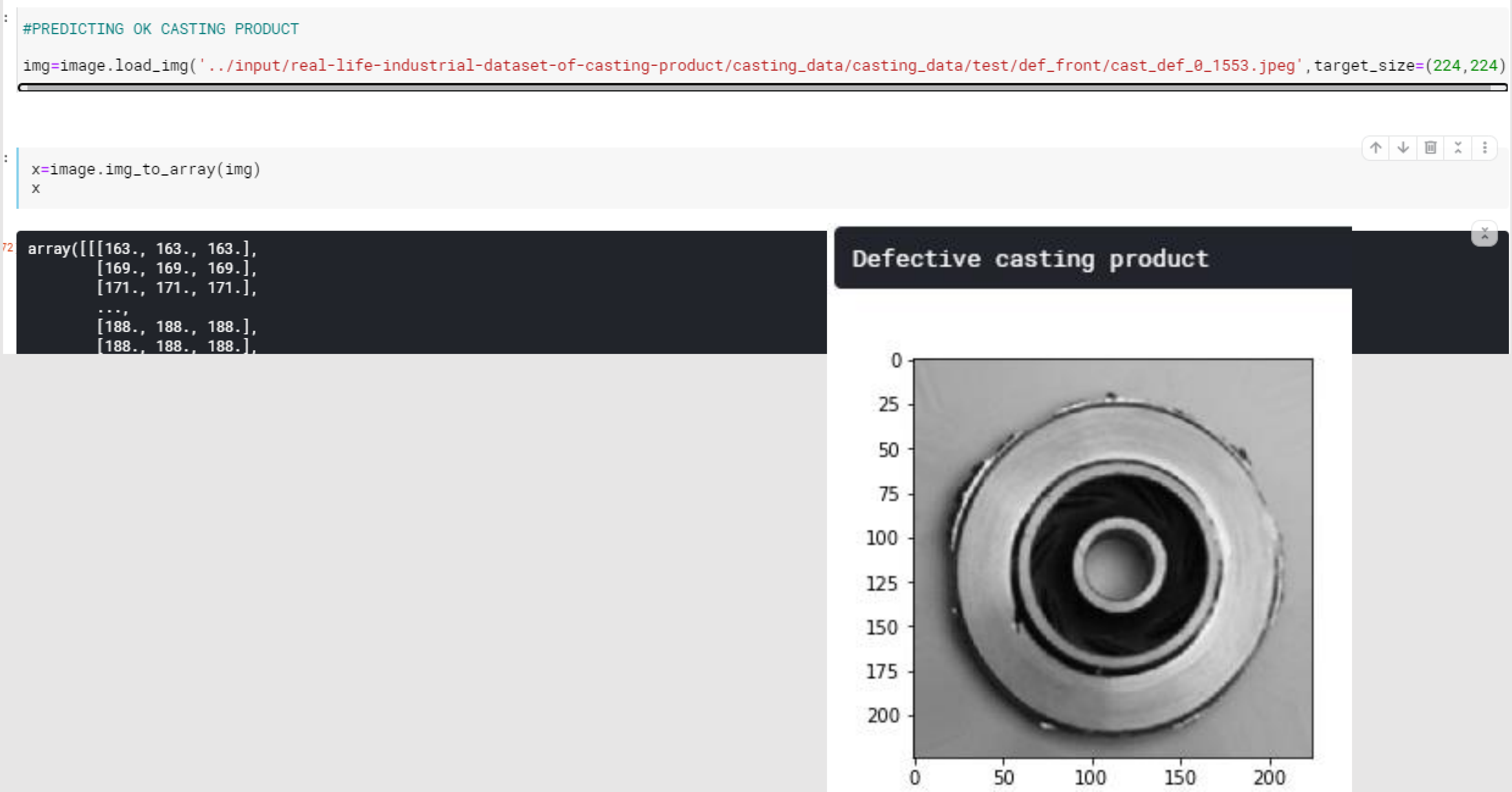


```
array([[[157., 157., 157.],
        [157., 157., 157.],
        [158., 158., 158.],
        ...,
        [130., 130., 130.],
        [130., 130., 130.],
        [130., 130., 130.]],

       [[157., 157., 157.],
        [157., 157., 157.],
        [157., 157., 157.],
        ...,
        [131., 131., 131.],
        [131., 131., 131.],
        [130., 130., 130.]],
```



Defective casting product

TEST_6



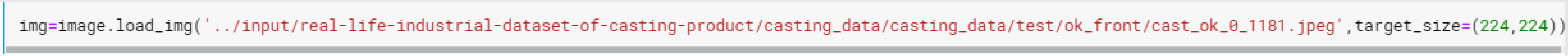
```
#PREDICTING OK CASTING PRODUCT

img=image.load_img('../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/test/def_front/cast_def_0_1553.jpeg',target_size=(224,224)
```

```
x=image.img_to_array(img)
x
```

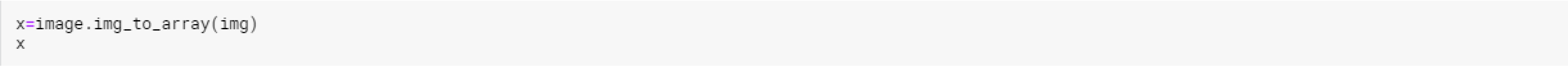
```
array([[[163., 163., 163.],
        [169., 169., 169.],
        [171., 171., 171.],
        ...,
        [188., 188., 188.],
        [188., 188., 188.],
```

Defective casting product

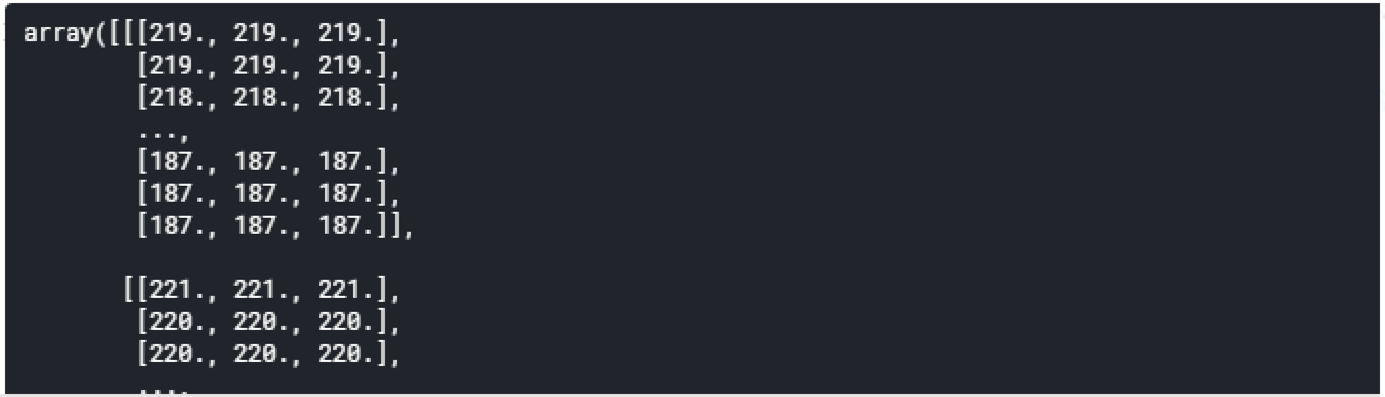
TEST_7



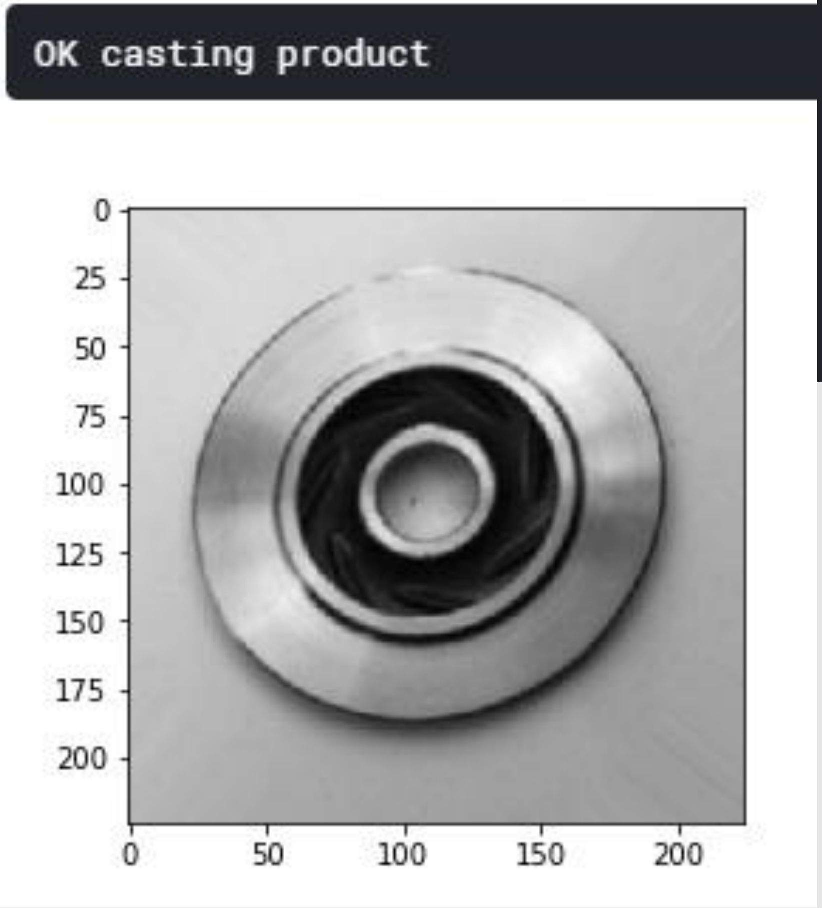
```
img=image.load_img('../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/test/ok_front/cast_ok_0_1181.jpeg',target_size=(224,224))
```



```
x=image.img_to_array(img)
x
```



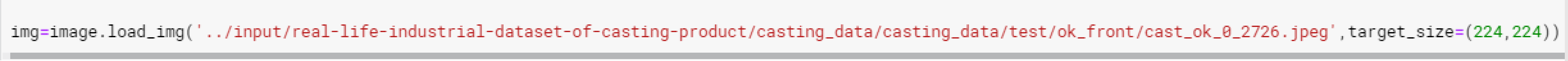
```
array([[[219., 219., 219.],
        [219., 219., 219.],
        [218., 218., 218.],
        ...,
        [187., 187., 187.],
        [187., 187., 187.],
        [187., 187., 187.]],

       [[221., 221., 221.],
        [220., 220., 220.],
        [220., 220., 220.],
        ...,
```



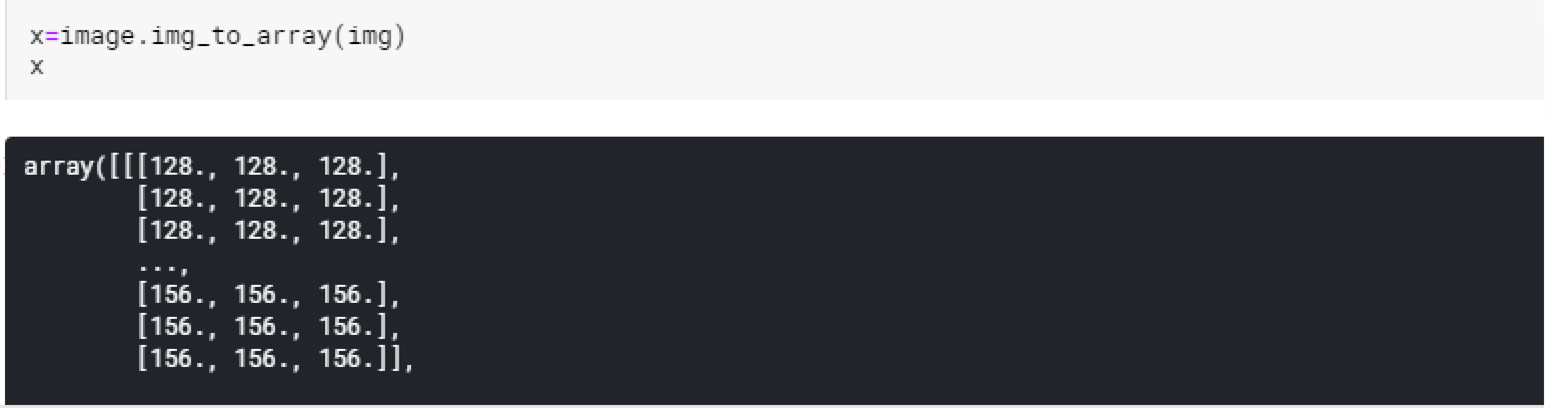
OK casting product

INCORRECT_PREDICTIONS:-

TEST_8

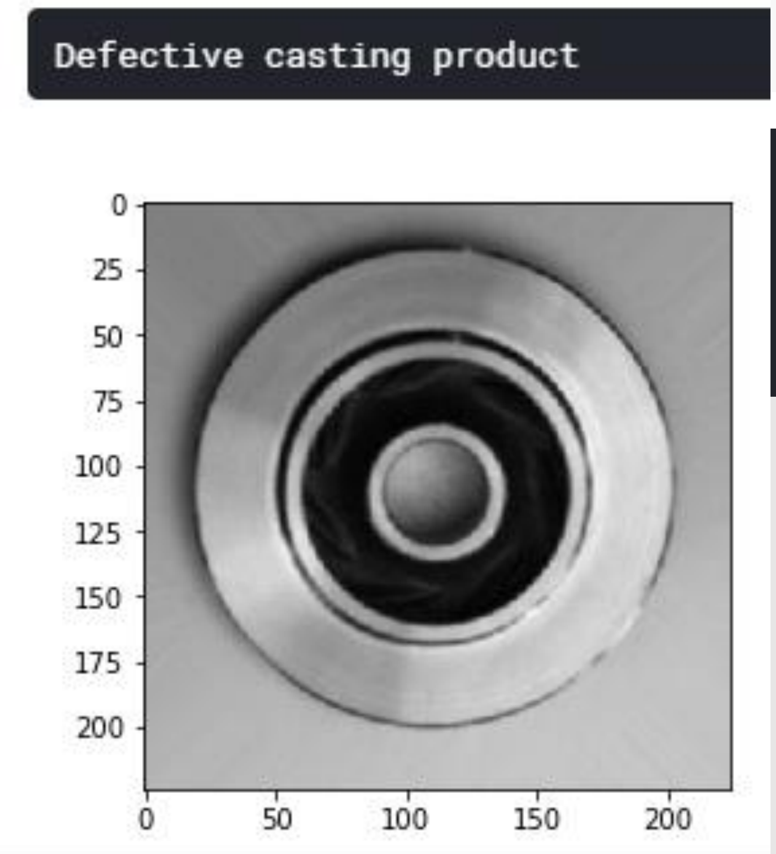


```
img=image.load_img('../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/test/ok_front/cast_ok_0_2726.jpeg',target_size=(224,224))
```



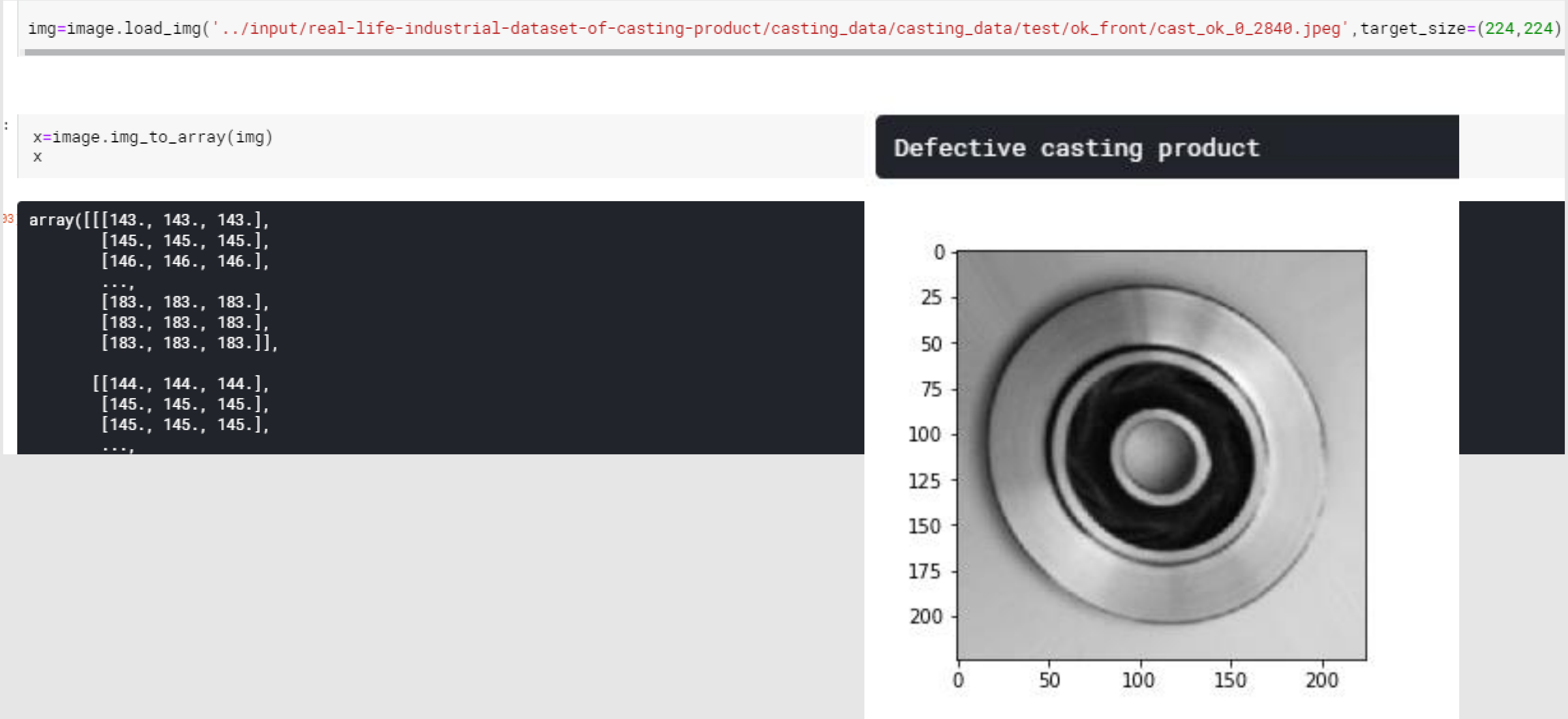
```
x=image.img_to_array(img)
x
```

```
array([[[128., 128., 128.],
        [128., 128., 128.],
        [128., 128., 128.],
        ...,
        [156., 156., 156.],
        [156., 156., 156.],
        [156., 156., 156.]],
```



```
Defective casting product
```

TEST_9



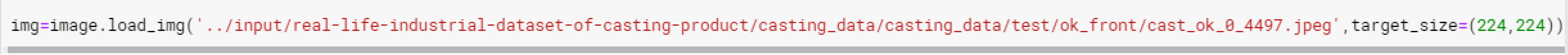
```
img=image.load_img('../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/test/ok_front/cast_ok_0_2840.jpeg',target_size=(224,224)
```

```
x=image.img_to_array(img)
x
```

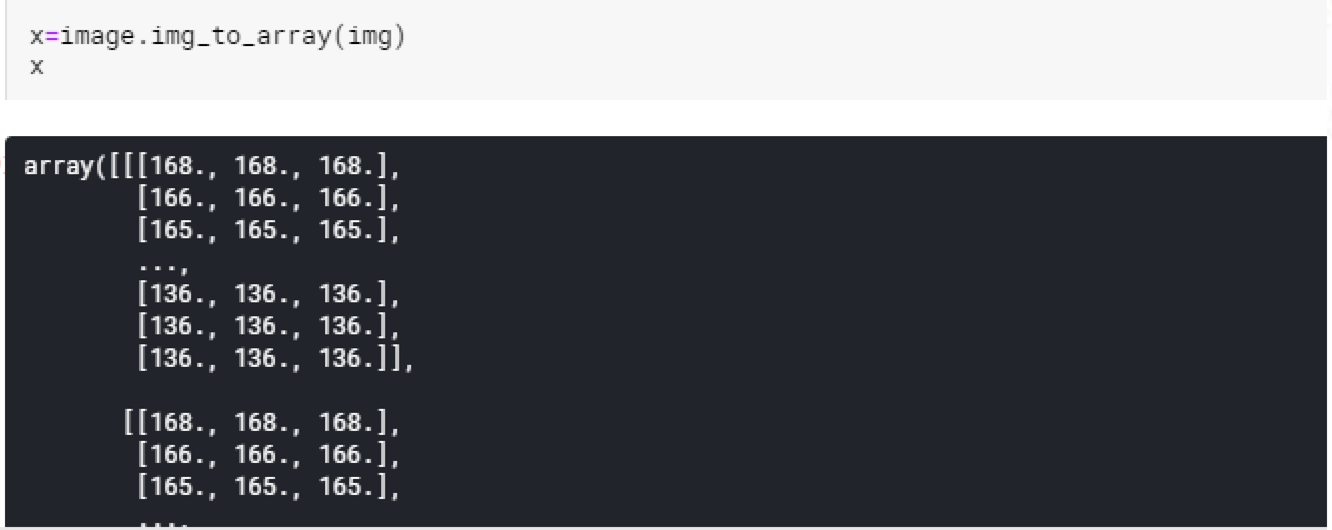
```
array([[[143., 143., 143.],
        [145., 145., 145.],
        [146., 146., 146.],
        ...,
        [183., 183., 183.],
        [183., 183., 183.],
        [183., 183., 183.]],

       [[144., 144., 144.],
        [145., 145., 145.],
        [145., 145., 145.],
        ...,
```

Defective casting product

TEST_10

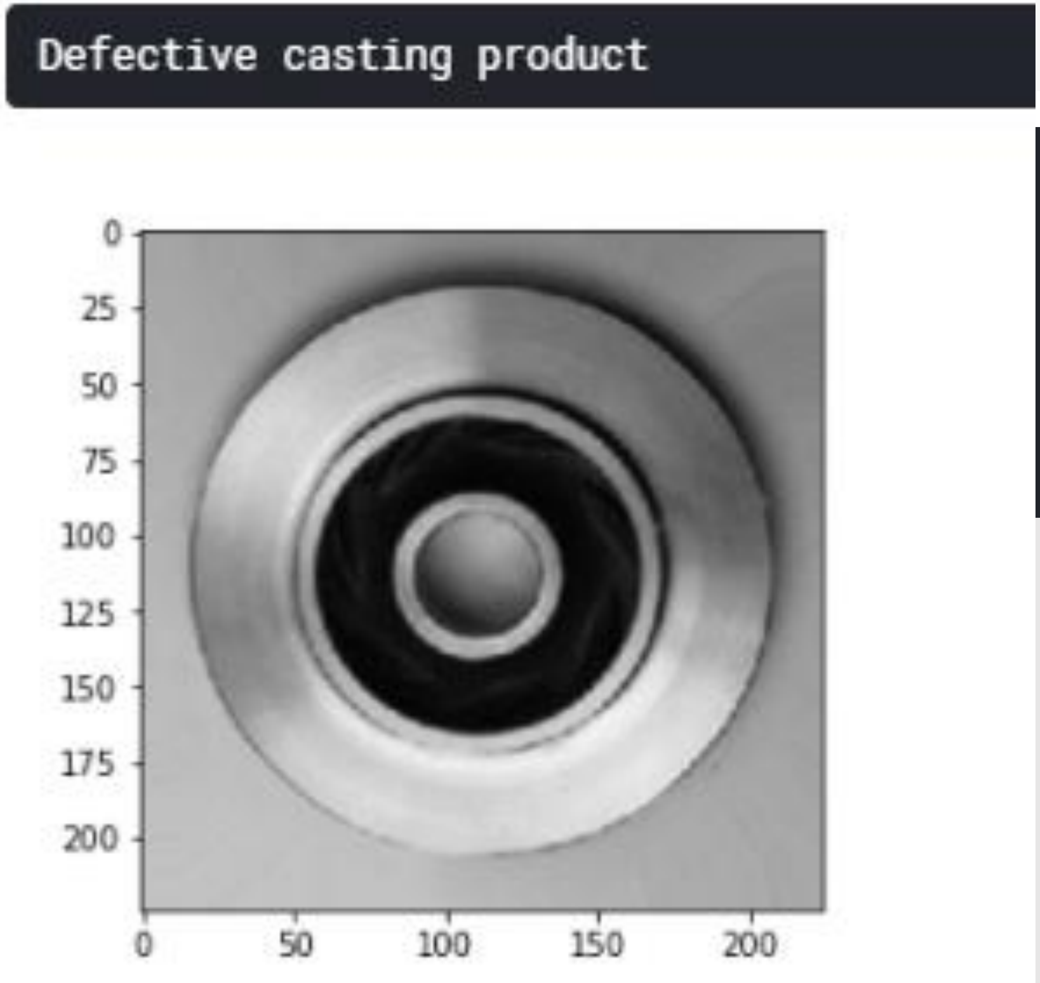


```
img=image.load_img('../input/real-life-industrial-dataset-of-casting-product/casting_data/casting_data/test/ok_front/cast_ok_0_4497.jpeg',target_size=(224,224))
```



```
x=image.img_to_array(img)
x
```

```
array([[[168., 168., 168.],
        [166., 166., 166.],
        [165., 165., 165.],
        ...,
        [136., 136., 136.],
        [136., 136., 136.],
        [136., 136., 136.]],

       [[168., 168., 168.],
        [166., 166., 166.],
        [165., 165., 165.],
        ...,
```



Defective casting product

RESULTS AND CONCLUSIONS:-

- First of all,we trained the model on 6633 images using vgg19 architecture.
- Thereafter we randomly picked 10 images out of 715 untrained images,the trained model predicted 7 images correctly i.e whether they are defected casting product or not.
- The overall accuracy of the model is approximately 54.12%.
- Therefore by introducing such technique in manufacturing industry,we will be able to reduce the time of inspection as well as it will increase the efficiency of production flow.
- Also rather than checking few samples manually,we will be able to check each and every sample through this technique in less span of time.

REFERENCES:-

- https://d1wqtxts1xzle7.cloudfront.net/61712988/IRJET-V6I127120200107-110252-1qwlegs.pdf?1578467836=&response-content-disposition=inline%3B+filename%3DIRJET_Image_Classification_Cat_and_Dog_I.pdf&Expires=1609868773&Signature=UoI~Vf3kwBJV4atLJHmEneDpWlkJf1vyb39qeOMXgYnjY6NAPqJKU5Qb7zJJrq0uuNYD6d5W6ESths8WgOTE1lXdKiDxX6JjaGC0b0UxyhkwAGXv6349YcdKUHHQENkdJa6KvmuLy4MnNXxZ9JArdFVXOQRfy0Ud-mq~2RcDzf~hZ38CrfjhjSicT5deVl7V76K9MGLz4sdS11KvFBCkvizFUKKBjcNKOB5~wM3gISreLAVqmxE-koz-XlLMl41y87Nz5qSAvBjlnJEamr3Nd--WdqRk~d95KTHnfJT0hjI4-zzVzoMW1MBJqElW3MBrCiK7iSYjgugjIXaPYBhd~A__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA
- https://www.cv-foundation.org/openaccess/content_cvpr_workshops_2014/W15/html/Razavian_CNN_Features_Off-the-Shelf_2014_CVPR_paper.html
- https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Wang_CNN-RNN_A_Unified_CVPR_2016_paper.html
- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
- https://www.quora.com/What-is-the-VGG-neural-network
- https://www.kaggle.com/ravirajsinh45/real-life-industrial-dataset-of-casting-product