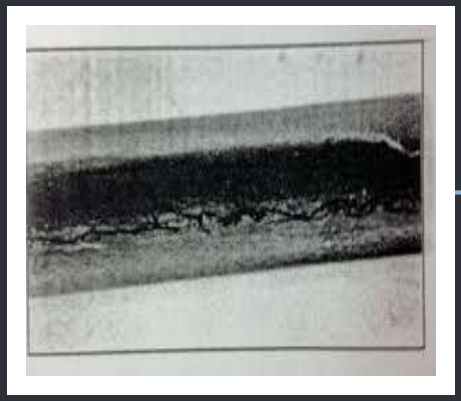
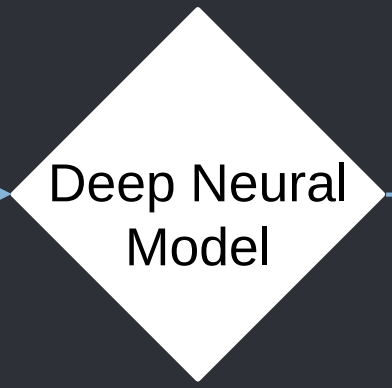
Metal Surface Defect Inspection Through Deep Neural Network

Goal



Test Image



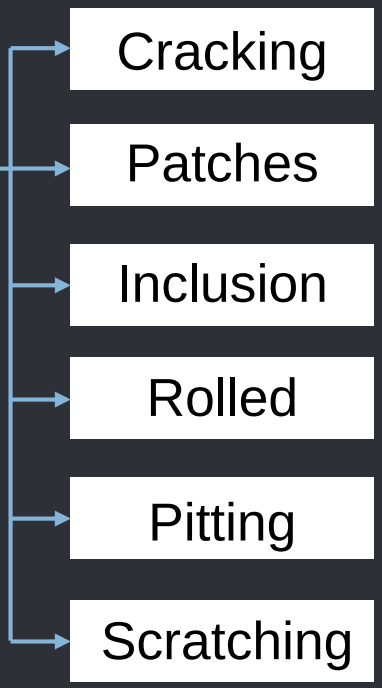


Figure : Image Classification Process

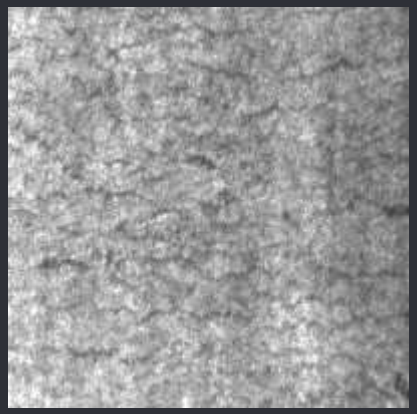
Agenda

- **Metal Surface Defects**
- **Deep Neural Network**
- **Defects Detection Method**

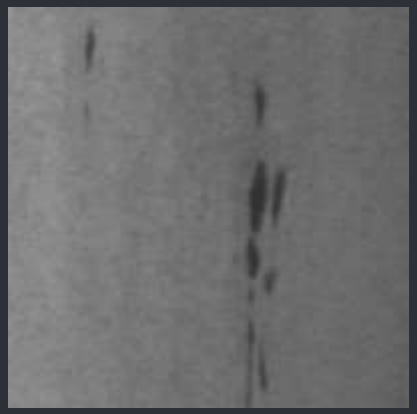
Metal Surface Defects

Types Of Defects Will Inspect

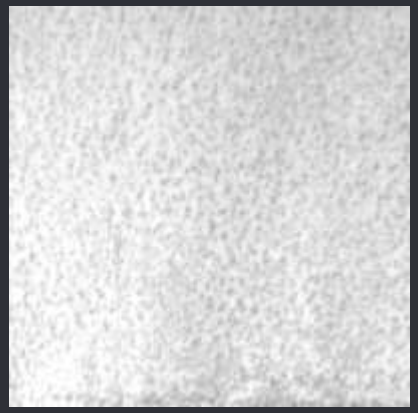
Scratching



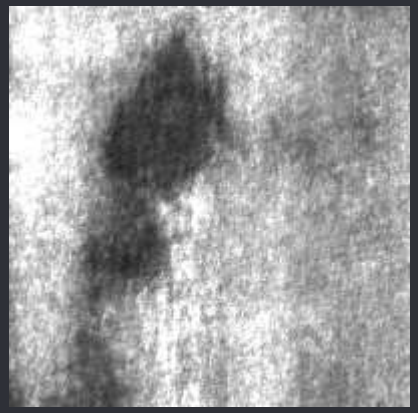
Cracking



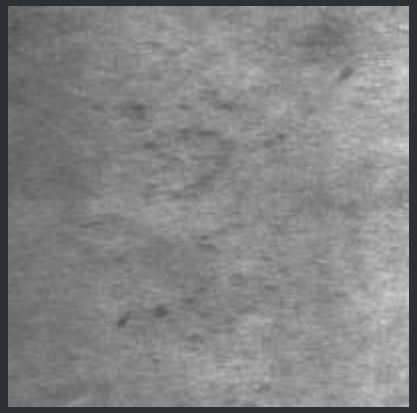
Inclusion



Pitting



Patches



Rolled

Figure : Six Types of Defect & Label

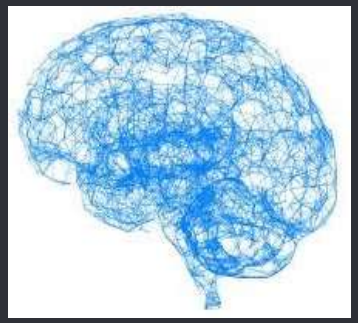
Why Metal Surface Defect Inspection Needed ?

Deep Neural Network

Artificial Intelligence

Artificial

Intelligence



Artificial Intelligence

‘‘Artificial Intelligence involves Machine that has Human Intellectual Properties’’

Figure : Artificial Intelligence Basic

Machine Learning

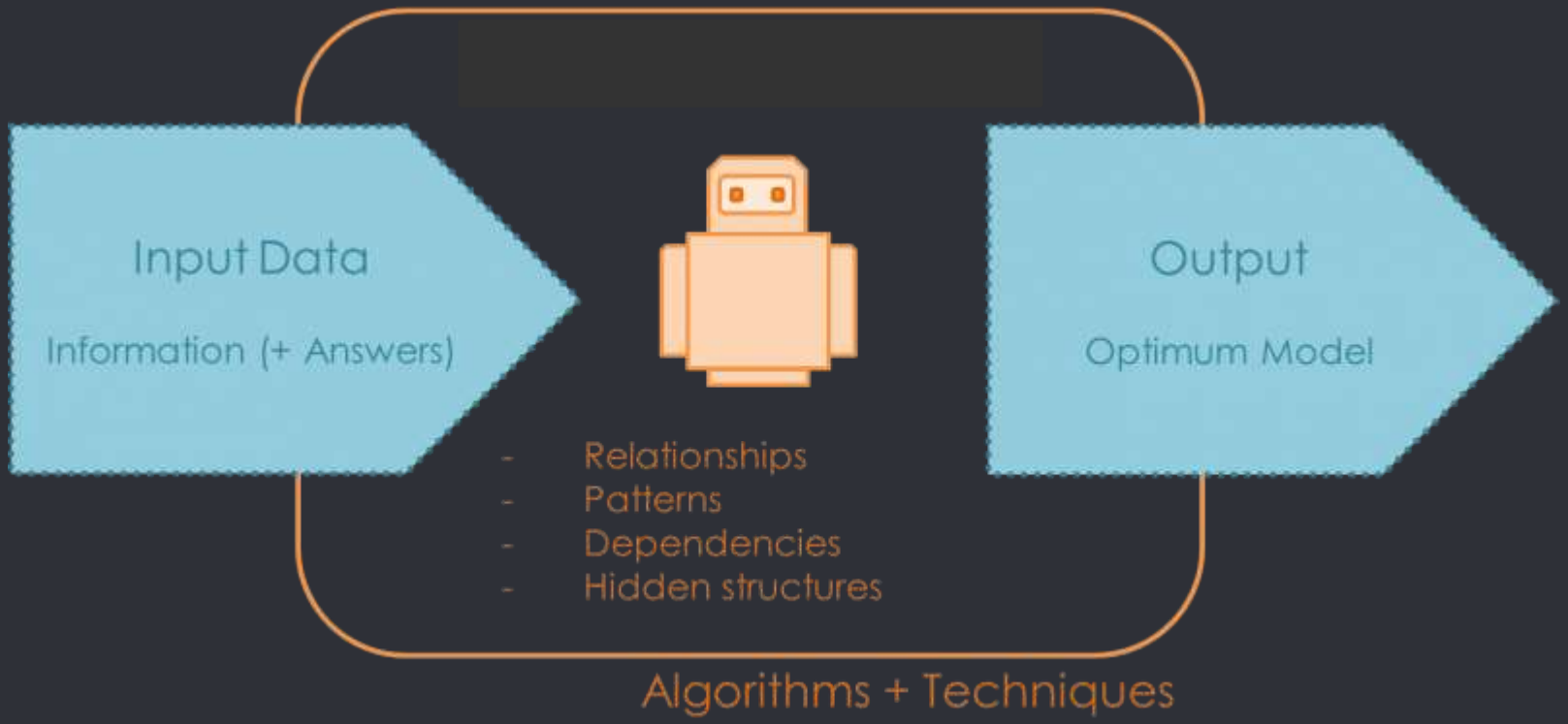


Figure : Machine Training Algorithm

Deep Learning

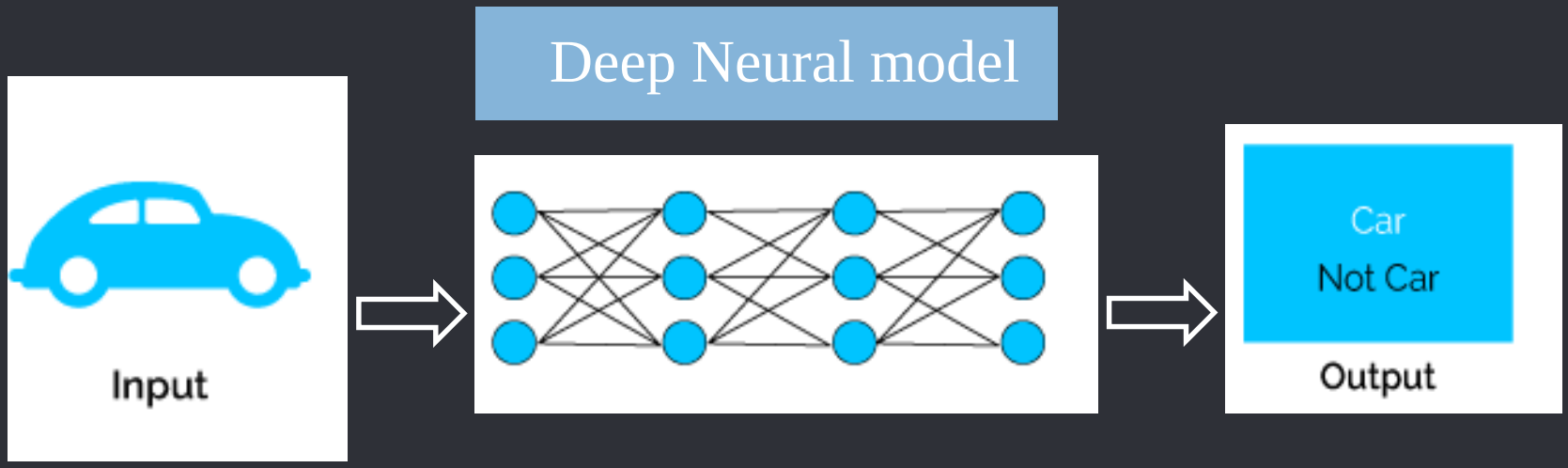


Figure : Deep Learning Algorithm

Defect Detection Method

Dataset

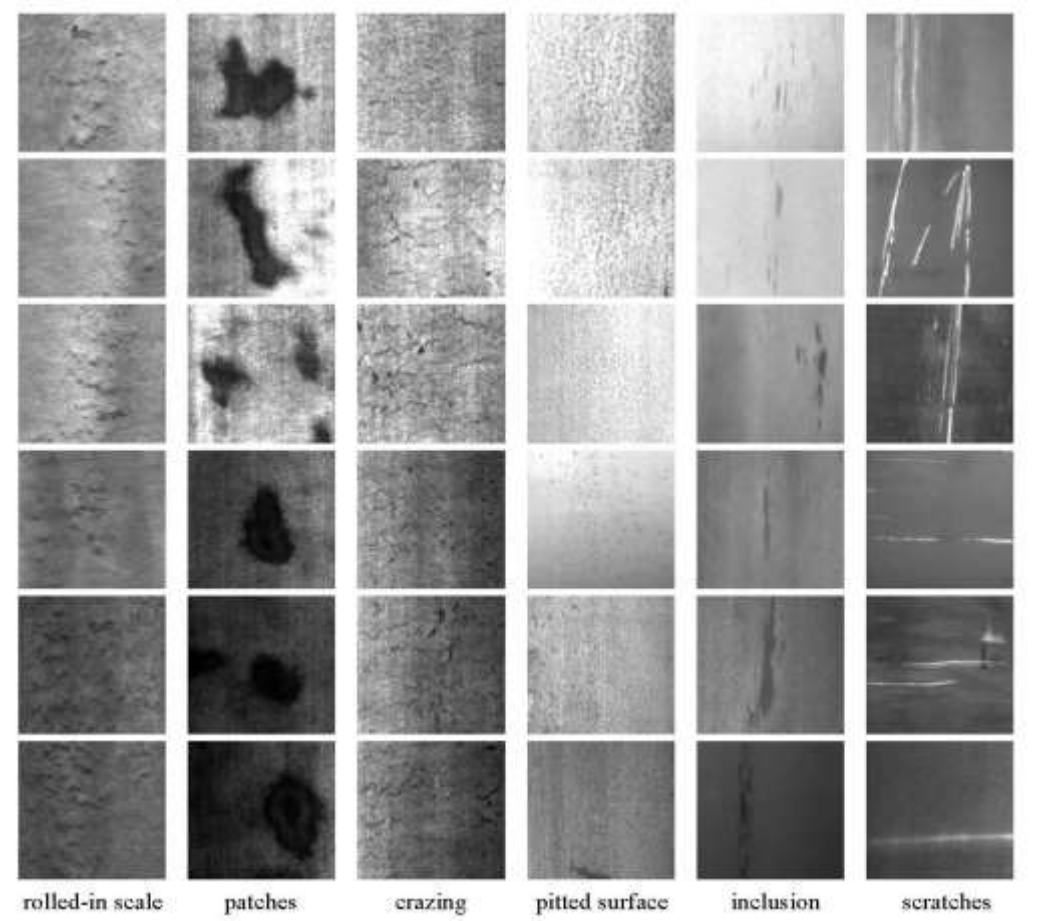


Figure : NEU surface defect database

Training Dataset

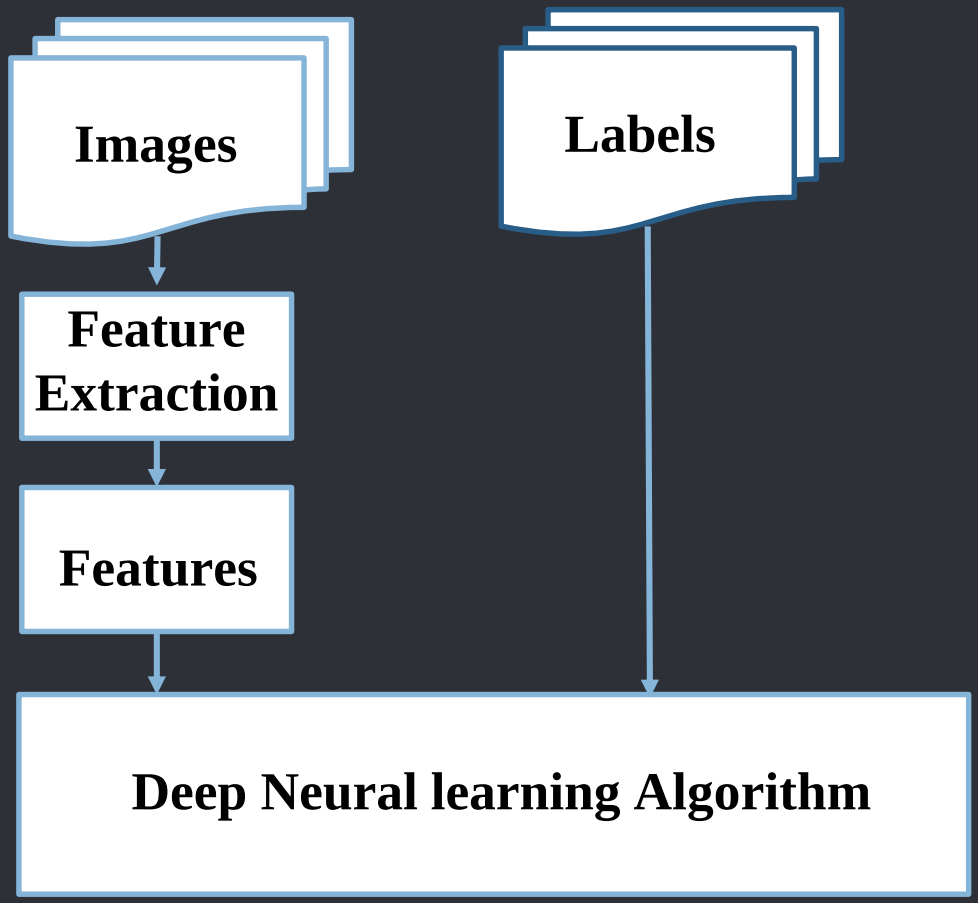


Figure : Dataset Training Algorithm

Training Algorithm Code

```
#Multiclass_defect_detection
# Imports
import cv2
import glob
import numpy as np
import os.path as path
from scipy import misc
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Activation, Dropout,Dense, Conv2D, MaxPooling2D,Input, Convolution2D, Flatten
from sklearn.metrics import accuracy_score, f1_score
from datetime import datetime
from keras.models import Model
import matplotlib.pyplot as plt
# IMAGE_PATH should be the path to the planesnet folder
IMAGE_PATH = 'Surface_defect'
file_paths = glob.glob(path.join(IMAGE_PATH, '*.jpg'))

# Load the images
images = [cv2.imread(path) for path in file_paths]
images=[cv2.resize(image,(50,50)) for image in images ]
images = np.asarray(images)

# Get image size
image_size = np.asarray([images.shape[1], images.shape[2], images.shape[3]])
print(image_size)

# Scale
X_train = images / 255
# Read the labels from the filenames
n_images = images.shape[0]
y_train =[]
for i in range(n_images):
    filename = path.basename(file_paths[i])[0]
    y_train.append(int(filename[0]))
```

Figure : Training Algorithm Code

Training Algorithm Code

```
batch_size = 180
num_epochs = 20
kernel_size = 3
pool_size = 2
conv_depth_1 = 32
conv_depth_2 = 64
drop_prob_1 = 0.25
drop_prob_2 = 0.5
hidden_size = 512

num_train, height, width, depth = X_train.shape # there are 50000 training examples in CIFAR-10
#num_test = X_test.shape[0] # there are 10000 test examples in CIFAR-10
num_classes = np.unique(y_train).shape[0] # there are 10 image classes

X_train = X_train.astype('float32')
#X_test = X_test.astype('float32')
X_train /= np.max(X_train) # Normalise data to [0, 1] range
#X_test /= np.max(X_test) # Normalise data to [0, 1] range

Y_train = np_utils.to_categorical(y_train, num_classes) # One-hot encode the labels
#Y_test = np_utils.to_categorical(y_test, num_classes) # One-hot encode the labels

inp = Input(shape=(height, width, depth)) # depth goes last in TensorFlow back-end (first in Theano)
# Conv [32] -> Conv [32] -> Pool (with dropout on the pooling layer)
conv_1 = Convolution2D(conv_depth_1, (kernel_size, kernel_size), padding='same', activation='relu')(inp)
conv_2 = Convolution2D(conv_depth_1, (kernel_size, kernel_size), padding='same', activation='relu')(conv_1)
pool_1 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_2)
drop_1 = Dropout(drop_prob_1)(pool_1)
# Conv [64] -> Conv [64] -> Pool (with dropout on the pooling layer)
conv_3 = Convolution2D(conv_depth_2, (kernel_size, kernel_size), padding='same', activation='relu')(drop_1)
conv_4 = Convolution2D(conv_depth_2, (kernel_size, kernel_size), padding='same', activation='relu')(conv_3)
pool_2 = MaxPooling2D(pool_size=(pool_size, pool_size))(conv_4)
drop_2 = Dropout(drop_prob_1)(pool_2)
# Now flatten to 1D, apply FC -> ReLU (with dropout) -> softmax
flat = Flatten()(drop_2)
hidden = Dense(hidden_size, activation='relu')(flat)
drop_3 = Dropout(drop_prob_2)(hidden)
out = Dense(num_classes, activation='softmax')(drop_3)
```

Figure : Training Algorithm Code

Training Algorithm Code

```
model = Model(inputs=inp, outputs=out) # To define a model, just specify its input and output layers

model.compile(loss='categorical_crossentropy', # using the cross-entropy loss function
              optimizer='adam', # using the Adam optimiser
              metrics=['accuracy']) # reporting the accuracy

model.fit(X_train, Y_train,                 # Train the model using the training set...
          batch_size=batch_size, epochs=num_epochs,
          verbose=1, validation_split=0.1)

 # ...holding out 10% of the data for validation
#model.evaluate(X_test, Y_test, verbose=1)  # Evaluate the trained model on the test set!
```

Figure : Training Algorithm Code

Testing Dataset

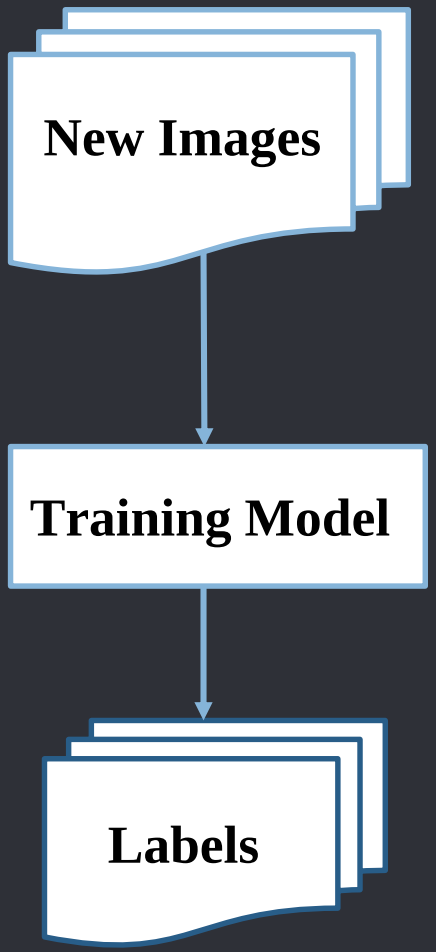


Figure : Dataset Testing Algorithm

Testing Algorithm Code

```
n_classes=6

IMAGE_PATH = 'X_test'
file_paths = glob.glob(path.join(IMAGE_PATH, '*.jpg'))
# Load the images
images = [cv2.imread(path) for path in file_paths]
act_image=[cv2.resize(image,(200,200)) for image in images ]
images=[cv2.resize(image,(50,50)) for image in images ]
images= np.asarray(images)
# Get image size
image_size = np.asarray([images.shape[1], images.shape[2], images.shape[3]])
print(image_size)
# Scale
images = images / 255
X_test=images
X_test = X_test.astype('float32')
X_test /= np.max(X_test)
test_predictions = model.predict(X_test)
a=len(X_test)
proba = model.predict(X_test[0:a])
test_predictions = np.round(test_predictions)
```

Figure : Testing Algorithm Code

Testing Dataset

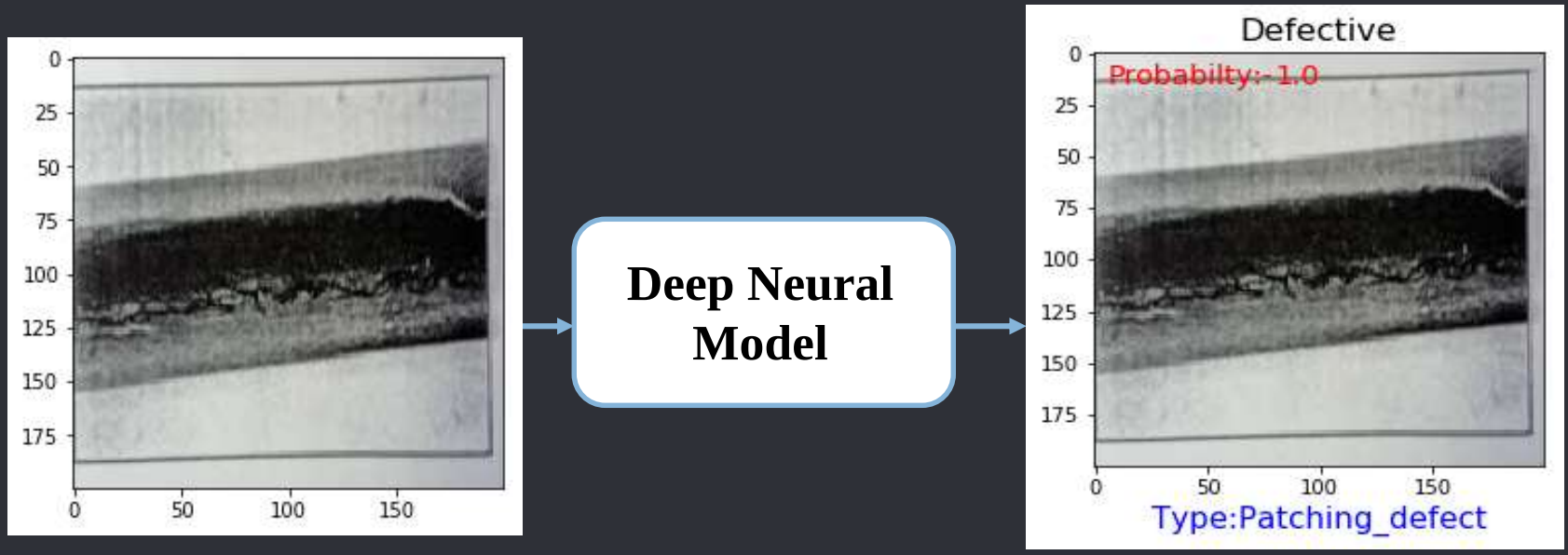
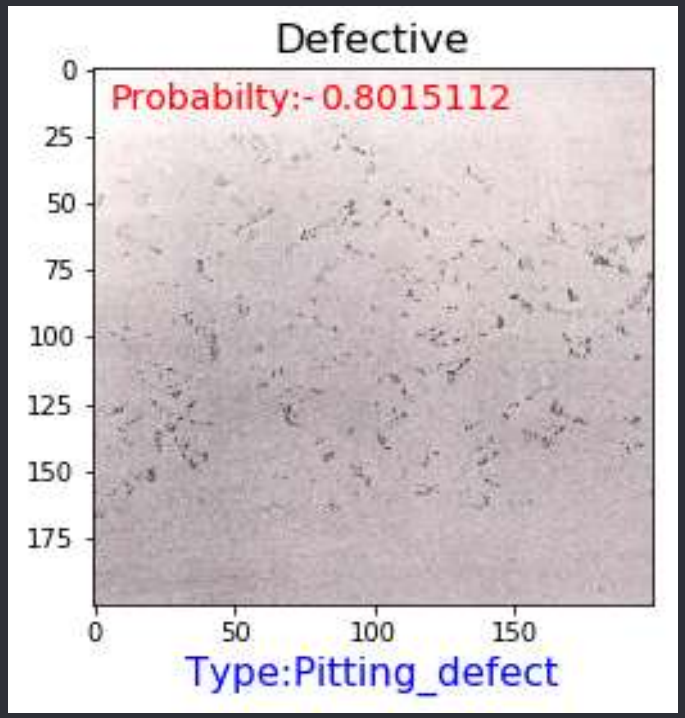


Figure : Testing A Sample Image

• Result



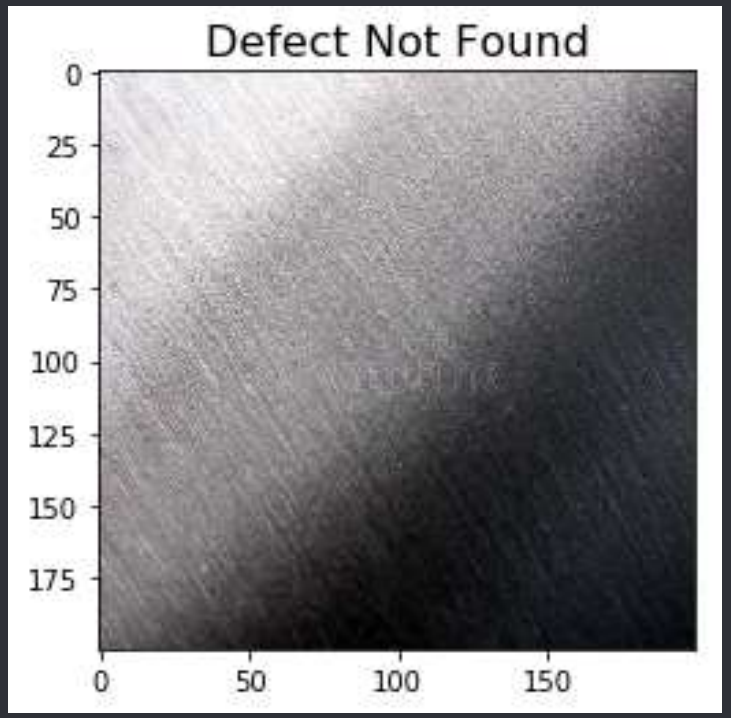


Figure : Images Label Detected

Thank You