Model Explainability in Industrial Image Detection

Keywords : Image Classification | Data Augmentation | CNN | Model Explanation | Error Analysis

---

Table of Contents

Overview

Task Details

- Select or create a dataset that includes images of industrial equipment labelled as 'defective' or 'non-defective', with additional labels for the type of defect in defective images.
- Train a machine learning model to classify images into the two main categories.
- Evaluate the model's performance using classification metrics such as accuracy, precision, and recall.

About Dataset

This dataset provides image data of impellers for submersible pumps.

Submersible Pump

Impeller

The image data is labeled with **ok(normal)** for non-defective equipment and **def(defect/anomaly)** for defective equipment.

Table of Contents

Import libraries

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.image import imread
import cv2
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import holoviews as hv
from holoviews import opts
hv.extension('bokeh')
import json

from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from keras.models import Sequential, load_model
from keras.layers import Activation, Dropout, Flatten, Dense, Conv2D, MaxPooling2D
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.utils import plot_model
from keras import backend
from sklearn.metrics import confusion_matrix, classification_report
```

Table of Contents

Load the dataset

In [ ]:
```
train_path = 'C:\\Users\\usaid\\Downloads\\casting_data\\train'
test_path = 'C:\\Users\\usaid\\Downloads\\casting_data\\test'
```

In [ ]:
```
plt.figure(figsize=(10,8))
ok = plt.imread(train_path + '\\ok_front\\cast_ok_0_1.jpeg')
plt.subplot(1, 2, 1)
plt.axis('off')
plt.title("ok", weight='bold', size=20)
plt.imshow(ok,cmap='gray')
plt.show()

ng = plt.imread(train_path + '\\def_front\\cast_def_0_8.jpeg')
plt.subplot(1, 2, 2)
plt.axis('off')
plt.title("def", weight='bold', size=20)
plt.imshow(ng,cmap='gray')

plt.show()
```

def

Table of Contents

Pre-Processing

In training models for image classification, **Data Augmentation** techniques are needed to build more robust models.

What is Data Augmentation?

When training with image data without data augmentation, we simply need the specified number of data and create a mini-batch. When executing data augmentation, after acquiring the data, various augmentation techniques are applied to the image to create a new mini-batch.
The main parameters of data augmentation techniques are as follows :

- **rotation_range** : Rotate the image (ex. 50 -> rotate randomly in -50°~50°)
- **zoom_range** : Zoom in/out on the image (ex. 0.5 -> zoom in/out randomly in 1-0.5~1+0.5)
- **brightness_range** : Change the brightness (ex. [0.3,1.0] -> change randomly in [0.3,1.0])
- **vertical_flip** : Flip the image upside down
- **horizontal_flip** : Flip the image left or right
- **height_shift_range** : Move the image up or down in parallel (ex. 0.3 -> move up/down randomly in [-0.3*Height, 0.3*Height])
- **width_shift_range** : Move the image left or right in parallel (ex. 0.3 -> move left/right randomly in [-0.3*Width, 0.3*Width])
- **rescale** : The image is normalized by multiplying each pixel value by a constant. (ex. 1/255 -> normalize the RGB value of each pixel between 0.0 and 1.0)

In [ ]:
```
img = cv2.imread(train_path + '\\ok_front\\cast_ok_0_1.jpeg')
img_4d = img[np.newaxis]
plt.figure(figsize=(25,10))
generators = {"rotation":ImageDataGenerator(rotation_range=180),
              "zoom":ImageDataGenerator(zoom_range=0.7),
```

```
              "brightness":ImageDataGenerator(brightness_range=[0.2,1.0]),
              "height_shift":ImageDataGenerator(height_shift_range=0.7),
              "width_shift":ImageDataGenerator(width_shift_range=0.7)}

plt.subplot(1, 6, 1)
plt.title("Original", weight='bold', size=15)
plt.imshow(img)
plt.axis('off')
cnt = 2
for param, generator in generators.items():
    image_gen = generator
    gen = image_gen.flow(img_4d, batch_size=1)
    batches = next(gen)
    g_img = batches[0].astype(np.uint8)
    plt.subplot(1, 6, cnt)
    plt.title(param, weight='bold', size=15)
    plt.imshow(g_img)
    plt.axis('off')
    cnt += 1
plt.show()
```

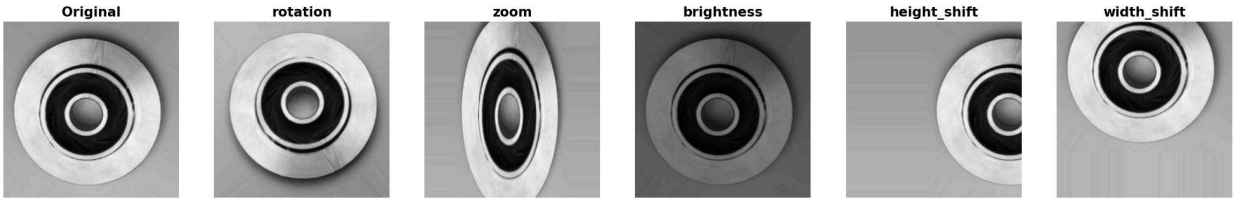


Table of Contents

Execute Data Augmentation

In Keras, you can pass ImageDataGenerator class as a dataset when training a model, and it creates a mini-batch by randomly applying the parameters. Empirically, if the parameter is set to an extreme high/low value, the image will be strongly converted and the training will be difficult to proceed. In addition, it seems that fine adjustment of parameters is required for each target data or analysis objectives in order to proceed with training successfully.

In [ ]:
```
image_gen = ImageDataGenerator(rescale=1/255,
                               zoom_range=0.1,
                               brightness_range=[0.9,1.0])
```

In [ ]:
```
image_shape = (300,300,1)
batch_size = 32

train_set = image_gen.flow_from_directory(train_path,
                                          target_size=image_shape[:2],
                                          color_mode="grayscale",
                                          classes={'def_front': 0, 'ok_front': 1},
                                          batch_size=batch_size,
                                          class_mode='binary',
                                          shuffle=True,
                                          seed=0)

test_set = image_gen.flow_from_directory(test_path,
```

```
                                        target_size=image_shape[:2],
                                        color_mode="grayscale",
                                        classes={'def_front': 0, 'ok_front': 1},
                                        batch_size=batch_size,
                                        class_mode='binary',
                                        shuffle=False,
                                        seed=0)
```

```
Found 6633 images belonging to 2 classes.
Found 715 images belonging to 2 classes.
```

In [ ]:
```
train_set.class_indices
```

Out[ ]:
```
{'def_front': 0, 'ok_front': 1}
```

- class 0 : defect
- class 1 : ok

Table of Contents

Modeling

Model-Settings

The elements of the model are listed below.

- **Sequential** : model container
- **Conv2D** : convolutional layer for 2D images
  - **filters** : number of filters
    - numbers such as 16, 32, 64, 128, 256 and 512 tend to be used, and there is a technique to increase the number of filter for the complicated task and decrease it for simple one.
  - **kernel_size** : filter size (width * height)
    - combinations of odd numbers such as 3x3, 5x5, 7x7 tend to be used.
  - **strides** : window size used for convolution
  - **input_shape** : size of input images (width/height, color channel)
    - if you input color images as it is, the model will need convolutions for 3 RGB channels, which will increase the amount of calculation(graysclaed images need less calculations).
  - **activation** : activation function
  - **padding** : adjust the size of the layer output. When set to 'same', the pixels are filled with 0 so that the input and output sizes are the same.
- **MaxPooling2D** : pooling layer for 2D images
  - **pool_size** : specify width/height range and extract the largest pixel in this range to downscale the input
  - **strides** : window size used for pooling
- **Flatten** : convert input to linear vector

- **Dropout** : apply dropout and randomly set the input to the unit to 0 to prevent overfitting when updating weights
  - **rate** : ratio of dropping the input to the unit
- **Dense** : fully connected layer
  - **units** : number of dimensions of output
  - **activation** : activation function (binary classification : sigmoid, other objectives : softmax)

In [ ]:
```
backend.clear_session()
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=(7,7), strides=2, input_shape=image_shape, activati
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(Conv2D(filters=32, kernel_size=(3,3), strides=1, input_shape=image_shape, activati
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(Conv2D(filters=64, kernel_size=(3,3), strides=1, input_shape=image_shape, activati
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))
model.add(Flatten())
model.add(Dense(units=224, activation='relu'))
model.add(Dropout(rate=0.2))
model.add(Dense(units=1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()
```

```
WARNING:tensorflow:From C:\Users\usaid\AppData\Roaming\Python\Python311\site-packages\keras\src\backend.py:277: The name tf.reset_default_graph is deprecated. Please use tf.compat.v1.reset_default_graph instead.

WARNING:tensorflow:From C:\Users\usaid\AppData\Roaming\Python\Python311\site-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

WARNING:tensorflow:From C:\Users\usaid\AppData\Roaming\Python\Python311\site-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 16)      800

 max_pooling2d (MaxPooling2  (None, 75, 75, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 75, 75, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 37, 37, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 37, 37, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 18, 18, 64)        0
 g2D)

 flatten (Flatten)           (None, 20736)             0

 dense (Dense)               (None, 224)               4645088

 dropout (Dropout)           (None, 224)               0

 dense_1 (Dense)             (None, 1)                 225

=================================================================
Total params: 4669249 (17.81 MB)
Trainable params: 4669249 (17.81 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The figure below shows the model architecture.

In [ ]:
```
plot_model(model, show_shapes=True, expand_nested=True, dpi=60)
```

```
You must install pydot (`pip install pydot`) and install graphviz (see instructions at https://graphviz.gitlab.io/download/) for plot_model to work.
```

Table of Contents

Build-Model

Build settings:

- **EarlyStopping** : Conditions to stop training early
  - ex. validation loss not improved continuously in 2 epochs
- **ModelCheckpoint** : Model saving settings for each epoch

In [ ]:
```
model_save_path = 'casting_product_detection.hdf5'
early_stop = EarlyStopping(monitor='val_loss',patience=2)
checkpoint = ModelCheckpoint(filepath=model_save_path, verbose=1, save_best_only=True, monit
```

In [ ]:
```
n_epochs = 20
results = model.fit_generator(train_set, epochs=n_epochs, validation_data=test_set, callback
```

```
Epoch 1/20
WARNING:tensorflow:From C:\Users\usaid\AppData\Roaming\Python\Python311\site-packages\keras\s
rc\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.c
ompat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\usaid\AppData\Roaming\Python\Python311\site-packages\keras\s
rc\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is depreca
ted. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

208/208 [==============================] - ETA: 0s - loss: 0.5324 - accuracy: 0.7215
Epoch 1: val_loss improved from inf to 0.25832, saving model to casting_product_detection.hdf
5
208/208 [==============================] - 206s 986ms/step - loss: 0.5324 - accuracy: 0.7215
- val_loss: 0.2583 - val_accuracy: 0.9161
Epoch 2/20
208/208 [==============================] - ETA: 0s - loss: 0.2530 - accuracy: 0.8960
Epoch 2: val_loss improved from 0.25832 to 0.14177, saving model to casting_product_detectio
n.hdf5
208/208 [==============================] - 70s 338ms/step - loss: 0.2530 - accuracy: 0.8960 -
val_loss: 0.1418 - val_accuracy: 0.9538
Epoch 3/20
208/208 [==============================] - ETA: 0s - loss: 0.1493 - accuracy: 0.9441
Epoch 3: val_loss improved from 0.14177 to 0.10607, saving model to casting_product_detectio
n.hdf5
208/208 [==============================] - 102s 490ms/step - loss: 0.1493 - accuracy: 0.9441
- val_loss: 0.1061 - val_accuracy: 0.9636
Epoch 4/20
208/208 [==============================] - ETA: 0s - loss: 0.1032 - accuracy: 0.9652
Epoch 4: val_loss improved from 0.10607 to 0.05649, saving model to casting_product_detectio
n.hdf5
208/208 [==============================] - 55s 265ms/step - loss: 0.1032 - accuracy: 0.9652 -
val_loss: 0.0565 - val_accuracy: 0.9804
Epoch 5/20
208/208 [==============================] - ETA: 0s - loss: 0.0747 - accuracy: 0.9733
Epoch 5: val_loss improved from 0.05649 to 0.04179, saving model to casting_product_detectio
n.hdf5
208/208 [==============================] - 48s 228ms/step - loss: 0.0747 - accuracy: 0.9733 -
val_loss: 0.0418 - val_accuracy: 0.9874
Epoch 6/20
208/208 [==============================] - ETA: 0s - loss: 0.0800 - accuracy: 0.9718
Epoch 6: val_loss improved from 0.04179 to 0.04171, saving model to casting_product_detectio
n.hdf5
208/208 [==============================] - 45s 217ms/step - loss: 0.0800 - accuracy: 0.9718 -
val_loss: 0.0417 - val_accuracy: 0.9804
Epoch 7/20
208/208 [==============================] - ETA: 0s - loss: 0.0468 - accuracy: 0.9849
Epoch 7: val_loss improved from 0.04171 to 0.04033, saving model to casting_product_detectio
n.hdf5
208/208 [==============================] - 49s 237ms/step - loss: 0.0468 - accuracy: 0.9849 -
val_loss: 0.0403 - val_accuracy: 0.9790
Epoch 8/20
208/208 [==============================] - ETA: 0s - loss: 0.0396 - accuracy: 0.9884
Epoch 8: val_loss improved from 0.04033 to 0.03620, saving model to casting_product_detectio
n.hdf5
208/208 [==============================] - 49s 237ms/step - loss: 0.0396 - accuracy: 0.9884 -
val_loss: 0.0362 - val_accuracy: 0.9902
Epoch 9/20
208/208 [==============================] - ETA: 0s - loss: 0.0444 - accuracy: 0.9846
Epoch 9: val_loss improved from 0.03620 to 0.02198, saving model to casting_product_detectio
n.hdf5
208/208 [==============================] - 51s 245ms/step - loss: 0.0444 - accuracy: 0.9846 -
```

```
val_loss: 0.0220 - val_accuracy: 0.9958
Epoch 10/20
208/208 [==============================] - ETA: 0s - loss: 0.0388 - accuracy: 0.9876
Epoch 10: val_loss did not improve from 0.02198
208/208 [==============================] - 62s 298ms/step - loss: 0.0388 - accuracy: 0.9876 -
val_loss: 0.0329 - val_accuracy: 0.9874
Epoch 11/20
208/208 [==============================] - ETA: 0s - loss: 0.0323 - accuracy: 0.9899
Epoch 11: val_loss did not improve from 0.02198
208/208 [==============================] - 55s 262ms/step - loss: 0.0323 - accuracy: 0.9899 -
val_loss: 0.0310 - val_accuracy: 0.9860
```

In [ ]:
```
model_history = { i:list(map(lambda x: float(x), j)) for i,j in results.history.items() }
with open('model_history.json', 'w') as f:
    json.dump(model_history, f, indent=4)
```

Table of Contents

Model Performance

In [ ]:
```
losses = pd.DataFrame(model_history)
losses.index = map(lambda x : x+1, losses.index)
losses.head(3)
```

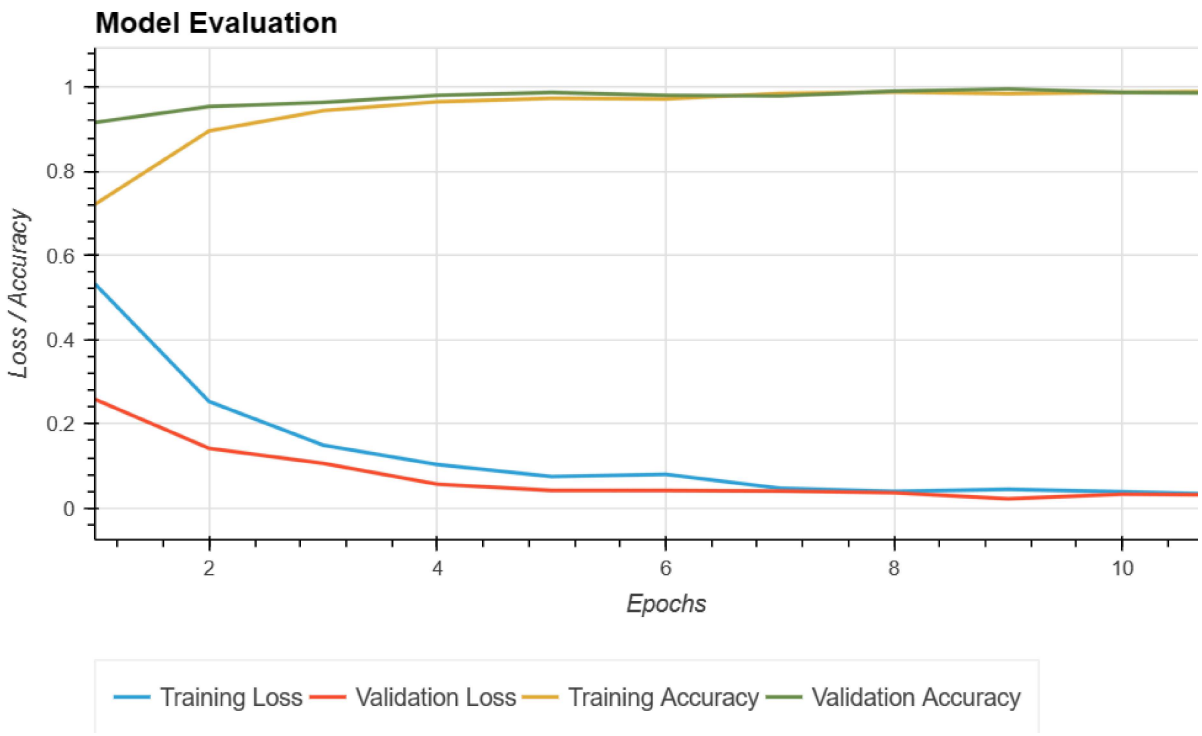
Out[ ]:

	loss	accuracy	val_loss	val_accuracy
1	0.532436	0.721544	0.258316	0.916084
2	0.253024	0.895975	0.141770	0.953846
3	0.149291	0.944068	0.106066	0.963636

Since the loss at the time of training and validation are steadily decreasing for each epoch, and the accuracy at the time of training and the validation are steadily increasing, it can be said that the training is generally successful.

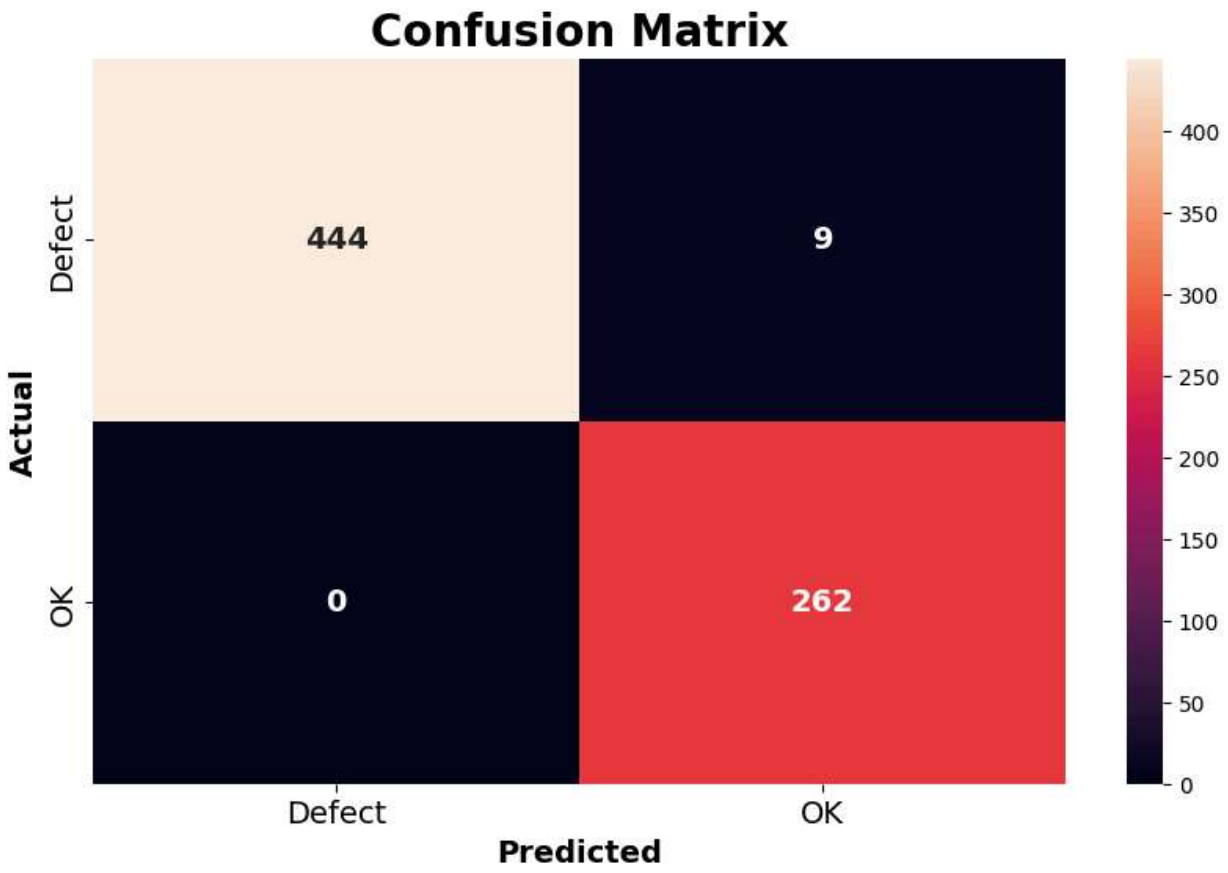
In [ ]:
```
g = hv.Curve(losses.loss, label='Training Loss') * hv.Curve(losses.val_loss, label='Validati
    * hv.Curve(losses.accuracy, label='Training Accuracy') * hv.Curve(losses.val_accuracy, l
g.opts(opts.Curve(xlabel="Epochs", ylabel="Loss / Accuracy", width=700, height=400,tools=['h
```

Out[ ]:



In [ ]:

```
pred_probability = model.predict_generator(test_set)
predictions = pred_probability > 0.5

plt.figure(figsize=(10,6))
plt.title("Confusion Matrix", size=20, weight='bold')
sns.heatmap(
    confusion_matrix(test_set.classes, predictions),
    annot=True,
    annot_kws={'size':14, 'weight':'bold'},
    fmt='d',
    xticklabels=['Defect', 'OK'],
    yticklabels=['Defect', 'OK'])
plt.tick_params(axis='both', labelsize=14)
plt.ylabel('Actual', size=14, weight='bold')
plt.xlabel('Predicted', size=14, weight='bold')
plt.show()
```



In [ ]:
```
print(classification_report(test_set.classes, predictions, digits=3))
```

```
              precision    recall  f1-score   support

           0      1.000     0.980     0.990       453
           1      0.967     1.000     0.983       262

    accuracy                          0.987       715
   macro avg      0.983     0.990     0.987       715
weighted avg      0.988     0.987     0.987       715
```

Table of Contents

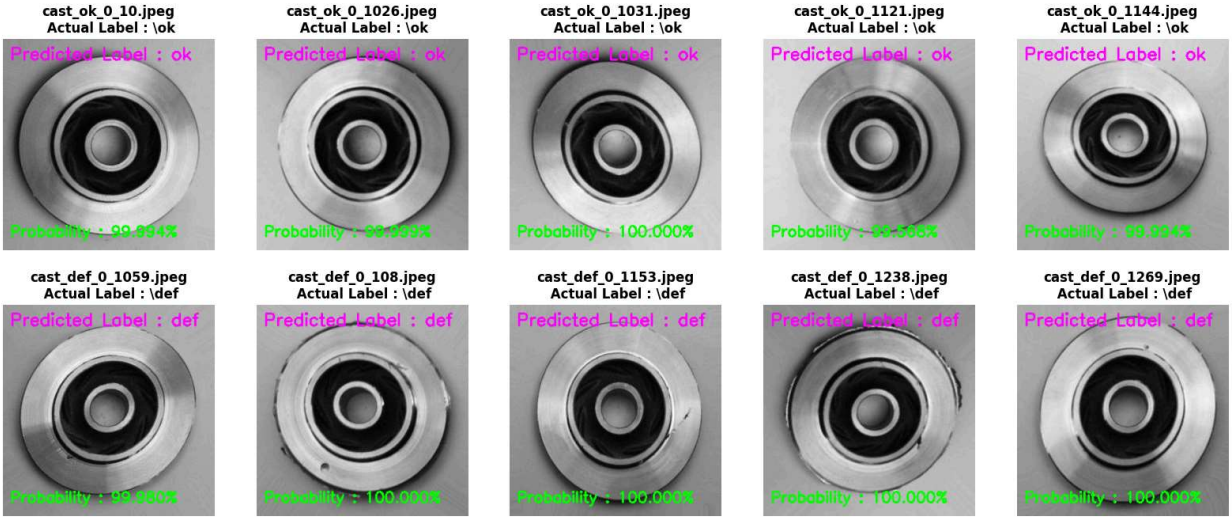
Predict on Some Images

Select images and apply it to the model.

In [ ]:
```
test_cases = ['\\ok_front\\cast_ok_0_10.jpeg', '\\ok_front\\cast_ok_0_1026.jpeg', '\\ok_fron
              '\\ok_front\\cast_ok_0_1144.jpeg','\\def_front\\cast_def_0_1059.jpeg', '\\def_
              '\\def_front\\cast_def_0_1238.jpeg', '\\def_front\\cast_def_0_1269.jpeg']
plt.figure(figsize=(20,8))
for i in range(len(test_cases)):
    img_pred = cv2.imread(test_path + test_cases[i], cv2.IMREAD_GRAYSCALE)
    img_pred = img_pred / 255 # rescale
    prediction = model.predict(img_pred.reshape(1, *image_shape))

    img = cv2.imread(test_path + test_cases[i])
    label = test_cases[i].split("_")[0]
```

```
    plt.subplot(2, 5, i+1)
    parts = test_cases[i].split('\\')
    plt.title(f"{parts[2]}\n Actual Label : {label}", weight='bold', size=12)
    # Predicted Class : defect
    if (prediction < 0.5):
        predicted_label = "def"
        prob = (1-prediction.sum()) * 100
    # Predicted Class : OK
    else:
        predicted_label = "ok"
        prob = prediction.sum() * 100

    cv2.putText(img=img, text=f"Predicted Label : {predicted_label}", org=(10, 30), fontFace
    cv2.putText(img=img, text=f"Probability : {'{:.3f}'.format(prob)}%", org=(10, 280), font
    plt.imshow(img,cmap='gray')
    plt.axis('off')

plt.show()
```

```
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 32ms/step
```



Conclusion

- **Data augmentation** proces can be easily incorporated into training process by using ImageDataGenerator.
- In data augmentation process, we should avoid excessive conversion and may require subtle adjustments depending on the dataset.
- According to the result of model interpretation, it was found that **the scratches/holes on the surface of the products and the unevenness around the products** are regarded as the

important features of defective products.
- Since we successfully built a model with relatively high accuracy, it is considered possible to incorporate the model into the camera of the inspection line and proceed with the automation of inspection.

Table of Contents