MAY 2024

Quality inspection

Adithyan .J.R

Casting product image data for quality inspection

By using Deep learning

About Dataset

Context:

This dataset is of casting manufacturing product.
Casting is a manufacturing process in which a liquid material is usually poured into a mould, which contains a hollow cavity of the desired shape, and then allowed to solidify.
Reason for collect this data is casting defects!!
Casting defect is an undesired irregularity in a metal casting process.
There are many types of defect in casting like blow holes, pinholes, burr, shrinkage defects, mould material defects, pouring metal defects, metallurgical defects, etc.

Defects are an unwanted thing in casting industry. For removing this defective product all industry have their quality inspection department. But the main problem is this inspection process is carried out manually. It is a very time-consuming process and due to human accuracy, this is not 100% accurate. This can because of the rejection of the whole order. So it creates a big loss in the company.

We decided to make the inspection process automatic and for this, we need to make deep learning classification model for this problem.

Contain:

These all photos are top view of submersible pump impeller(google search for better understanding).
The dataset contains total 7348 image data. These all are the size of (300*300) pixels grey-scaled images. In all images, augmentation already applied.
Also uploaded images size of 512x512 grayscale. This data set is without Augmentation. This contains 519 ok_front and 781 def_front impeller images.
For capturing these images requires stable lighting, for this we made a special arrangement.
there are mainly two categories:-
1) Defective 2)Ok

Making classification model we already split data for training and testing into two folders.
Both train and test folder contains def_front and ok_front subfolders.
train:- def_front have 3758 and ok_front have 2875 images
test:- def_front have:- def_front have 453 and ok_front have 262 images

Augmentation:

```
img_size = (300,300)
rand_seed = 555
batch_size = 32
epochs = 15

train_gen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    vertical_flip=True,
    rotation_range=40,
    brightness_range=[0.2, 1.5],
    validation_split=0.4,
)
```

```python
test_gen = ImageDataGenerator(rescale=1./255)

arg_train = {'target_size': img_size,
             'color_mode': 'rgb',
             'classes': {'ok_front': 0,
                         'def_front': 1},
             'class_mode': 'binary',
             'batch_size': batch_size,
             'seed': rand_seed}

dir_train='casting/casting_data/casting_data/train'
dir_test='casting/casting_data/casting_data/test'

# 80%
train_set = train_gen.flow_from_directory(directory=dir_train,
                                          subset='training',
                                          **arg_train)
#20%
valid_set = train_gen.flow_from_directory(directory=dir_train,
                                          subset='validation',
                                          **arg_train)
```

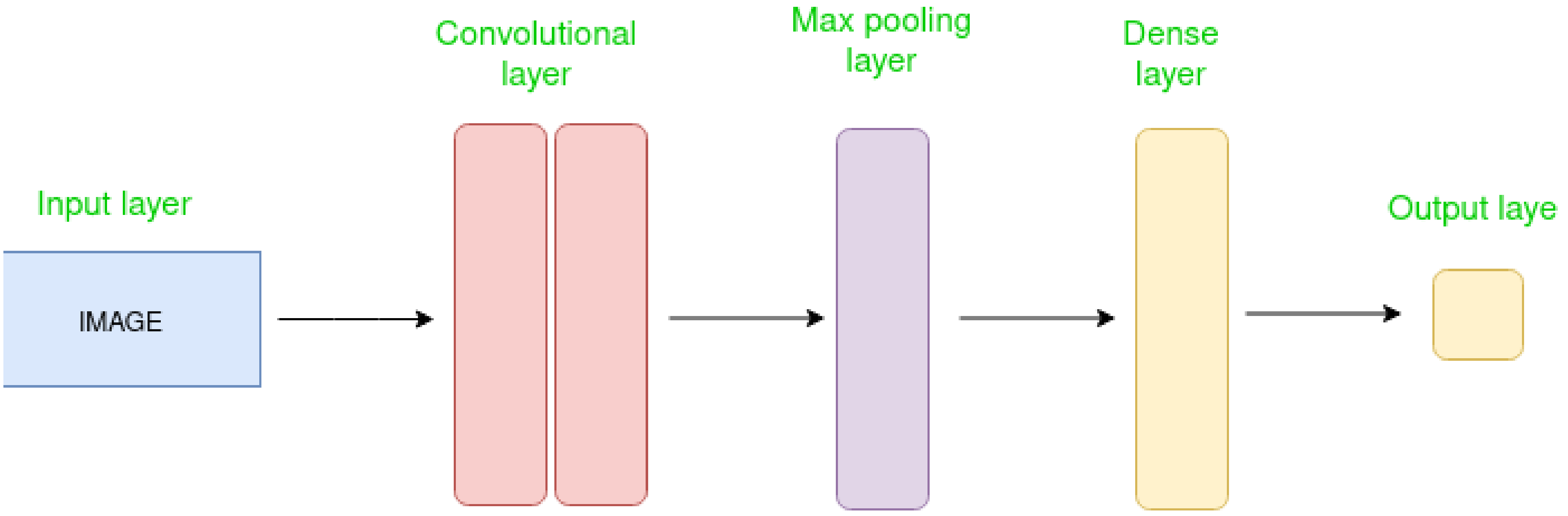
```
arg_test = {'target_size': img_size,
            'color_mode': 'rgb',
            'classes': {'ok_front': 0,
                        'def_front': 1},
            'class_mode': 'binary',
            'batch_size': batch_size,
            'seed': rand_seed,
            'shuffle': False}


# for the 0 and 1 ...etc
test_set = test_gen.flow_from_directory(directory=dir_test,
                                        **arg_test)
```

Feature selection:

```
model=Sequential()

model.add( Conv2D(filters=128,
                  kernel_size=(3,3),
                  strides=(2,2),
                  padding='valid',
                  kernel_initializer='he_uniform',
                  activation='relu',
                  input_shape=(300,300,3)
                  )
          )
model.add(MaxPooling2D(strides=(2,2),pool_size=(2,2),padding='valid'))

model.add(Conv2D(filters=128,
                  kernel_size=(3,3),
                  strides=(2,2),
                  padding='valid',
                  kernel_initializer='he_uniform',
                  activation='relu',
                  input_shape=(300,300,3)
                  )
)

model.add(MaxPooling2D(strides=(2,2),pool_size=(2,2),padding='valid'))
```

Fully Connected layer:

```
model.add(Flatten())
model.add(Dense(units=128,activation='relu',kernel_initializer='he_uniform'))
model.add(Dropout(0.2))
model.add(Dense(units=64,activation='relu',kernel_initializer='he_uniform'))
model.add(Dense(units=1,activation='sigmoid'))
```



Compile:

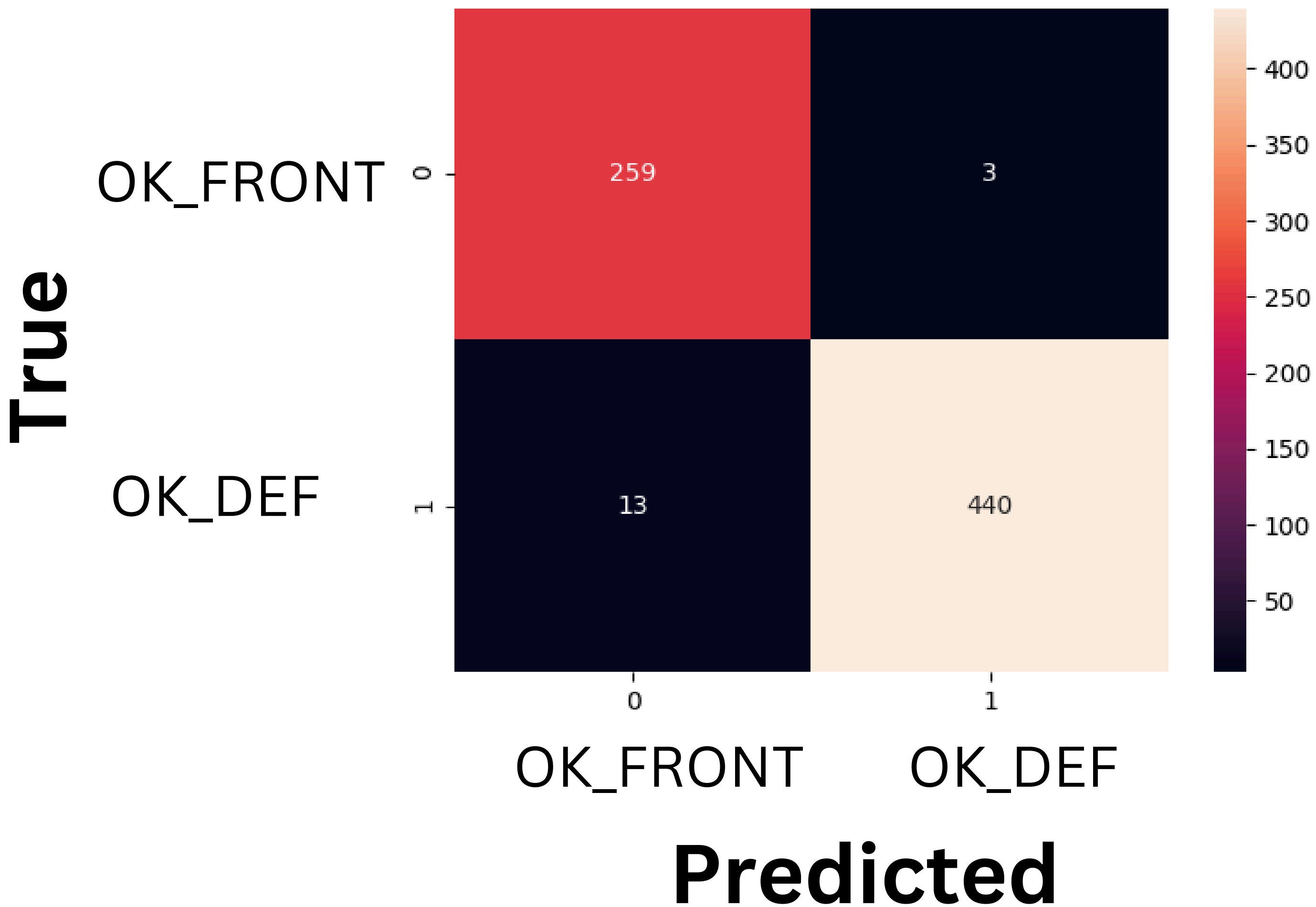
```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
model.fit(train_set,validation_data=valid_set,epochs=10,verbose=1)
```

```
the number of all the images in the training set is 6633
number of def imgs is 3758
number of ok imgs is 2875
the ratio between ok and def imgs is 0.7650345928685471
(300, 300) RGB
Found 3980 images belonging to 2 classes.
Found 2653 images belonging to 2 classes.
Found 715 images belonging to 2 classes.
```

```
Epoch 9/10
125/125 [==============================] - 315s 3s/step - loss: 0.1854 -
accuracy: 0.9269 - val_loss: 0.1369 - val_accuracy: 0.9514
Epoch 10/10
125/125 [==============================] - 319s 3s/step - loss: 0.1326 -
accuracy: 0.9505 - val_loss: 0.0861 - val_accuracy: 0.9736
```

Train Evaluation

Confusion Matrix:

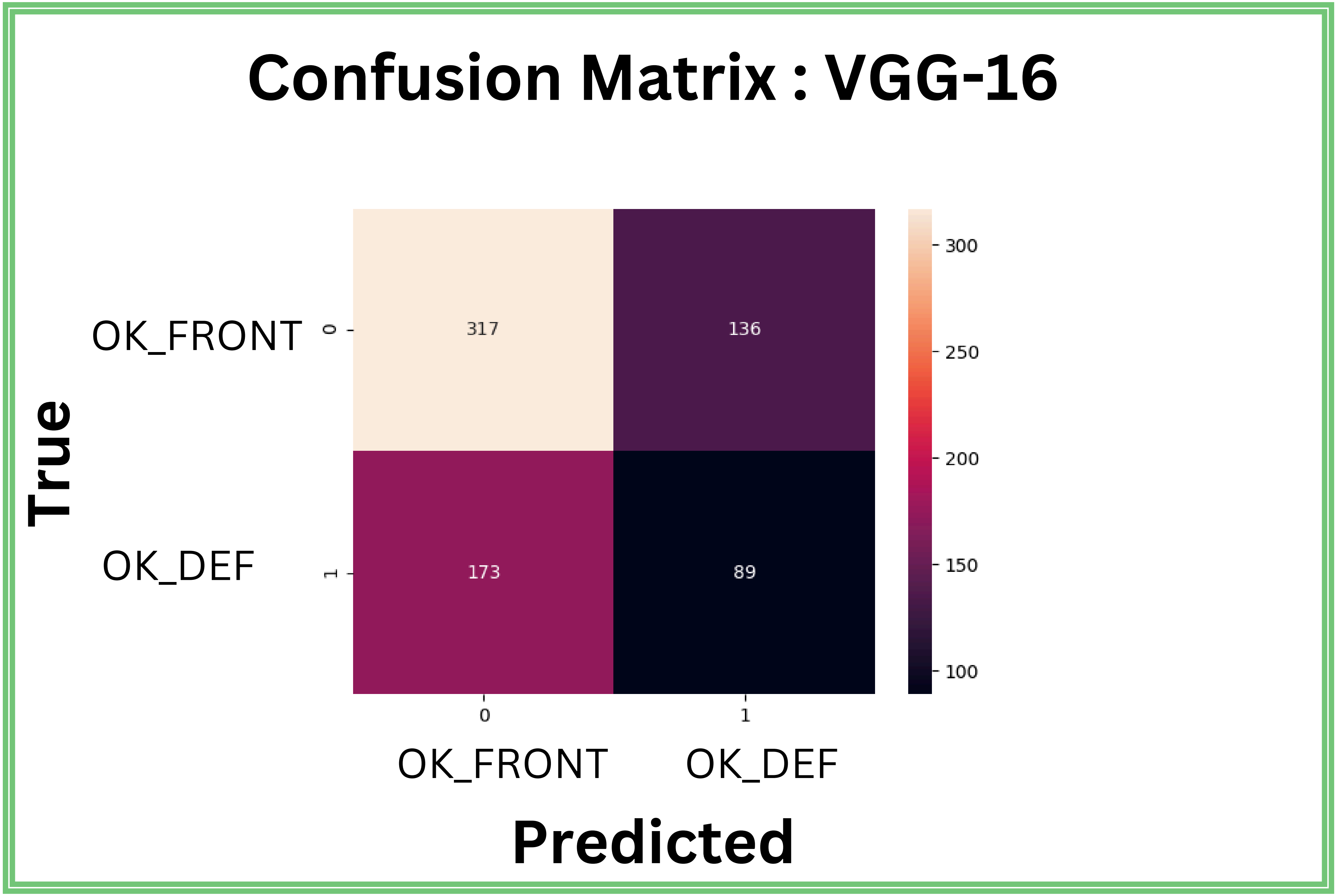


VGG-16

VGG-16 is a renowned CNN architecture by the Visual Geometry Group at Oxford. It's known for its depth, with 16 layers, including 13 convolutional and 3 fully connected layers. Its simplicity and effectiveness make it a popular choice for tasks like image classification and object recognition

```
(100, 100, 3)
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 3, 3, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              4719616
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 129
=================================================================
Total params: 20,123,457
Trainable params: 5,408,769
Non-trainable params: 14,714,688
_________________________________________________________________

Epoch 4/5
166/166 [==============================] - 48s 287ms/step - loss: 0.1074 -
accuracy: 0.9629 - val_loss: 0.0736 - val_accuracy: 0.9706
Epoch 5/5
166/166 [==============================] - 48s 287ms/step - loss: 0.1075 -
accuracy: 0.9627 - val_loss: 0.2255 - val_accuracy: 0.9155
```

Train Evaluation


Confusion Matrix : VGG-16
True
OK_FRONT
OK_DEF
0
1
317
136
173
89
300
250
200
150
100
0
1
OK_FRONT
OK_DEF
Predicted

Conclusion:

After rigorously evaluating my model against the VGG-16 model, I observed that my model exhibited a slightly higher accuracy rate. Moreover, its confusion matrix demonstrated a significant improvement, primarily due to its enhanced capability in accurately predicting outcomes across various classes. This comparative analysis underscores the superior predictive performance and efficacy of my model over the VGG-16 architecture.

Thank you