Metal-Casting-linearregression

June 1, 2023

1. Importing the necessary libraries

[1]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
%matplotlib inline
import cv2
```

2. Define the Logistic Regression model

[2]:
```
model = keras.Sequential([
    layers.Flatten(input_shape=(64, 64, 3)),
    layers.Dense(1, activation='sigmoid')
])
```

3. Compile the model

[3]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 12288)             0


 dense (Dense)               (None, 1)                 12289


=================================================================
Total params: 12,289
Trainable params: 12,289
Non-trainable params: 0

_________________________________________________________________
```

4. Load and preprocess the data

[4]:
```
train_datagen = keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
test_datagen = keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
```

```
[5]: training_set = train_datagen.flow_from_directory(
         r"D:\Casting\train",
         target_size=(64, 64),
         batch_size=32,
         class_mode='binary'
     )
```

Found 6633 images belonging to 2 classes.

```
[6]: test_set = test_datagen.flow_from_directory(
         r"D:\Casting\test",
         target_size=(64, 64),
         batch_size=32,
         class_mode='binary'
     )
```

Found 715 images belonging to 2 classes.

5. Train the model

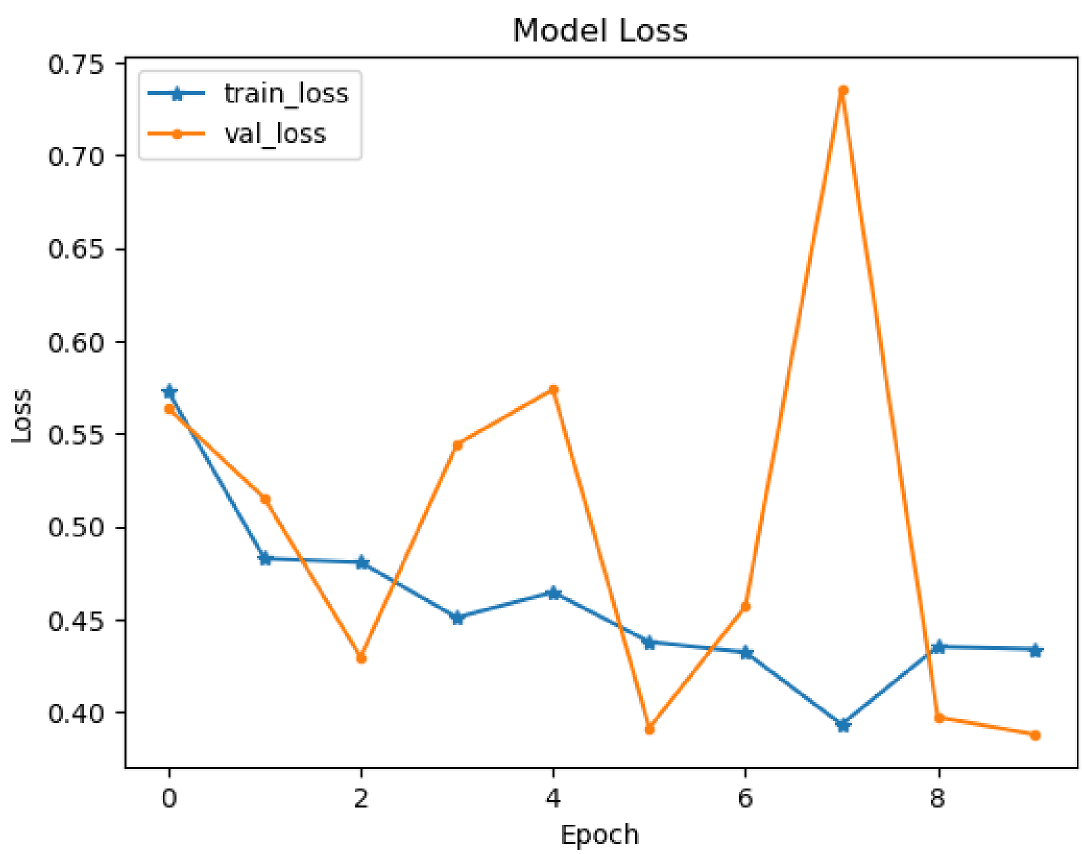
```
[7]: result = model.fit(
         training_set,
         epochs=10,
         validation_data=test_set

     )
```

```
Epoch 1/10
208/208 [==============================] - 28s 132ms/step - loss: 0.5727 -
accuracy: 0.7066 - val_loss: 0.5636 - val_accuracy: 0.7077
Epoch 2/10
208/208 [==============================] - 26s 125ms/step - loss: 0.4829 -
accuracy: 0.7742 - val_loss: 0.5154 - val_accuracy: 0.7301
Epoch 3/10
208/208 [==============================] - 26s 123ms/step - loss: 0.4808 -
accuracy: 0.7707 - val_loss: 0.4297 - val_accuracy: 0.7888
Epoch 4/10
208/208 [==============================] - 27s 128ms/step - loss: 0.4511 -
accuracy: 0.7891 - val_loss: 0.5443 - val_accuracy: 0.7077
Epoch 5/10
208/208 [==============================] - 27s 129ms/step - loss: 0.4647 -
accuracy: 0.7846 - val_loss: 0.5738 - val_accuracy: 0.7217
Epoch 6/10
208/208 [==============================] - 26s 127ms/step - loss: 0.4380 -
accuracy: 0.8016 - val_loss: 0.3916 - val_accuracy: 0.7888
Epoch 7/10
208/208 [==============================] - 26s 127ms/step - loss: 0.4324 -
accuracy: 0.8054 - val_loss: 0.4571 - val_accuracy: 0.7902
Epoch 8/10
```

```
208/208 [==============================] - 26s 126ms/step - loss: 0.3936 -
accuracy: 0.8189 - val_loss: 0.7355 - val_accuracy: 0.6601
Epoch 9/10
208/208 [==============================] - 26s 126ms/step - loss: 0.4354 -
accuracy: 0.7954 - val_loss: 0.3975 - val_accuracy: 0.7790
Epoch 10/10
208/208 [==============================] - 12s 58ms/step - loss: 0.4341 -
accuracy: 0.8042 - val_loss: 0.3882 - val_accuracy: 0.8294
```

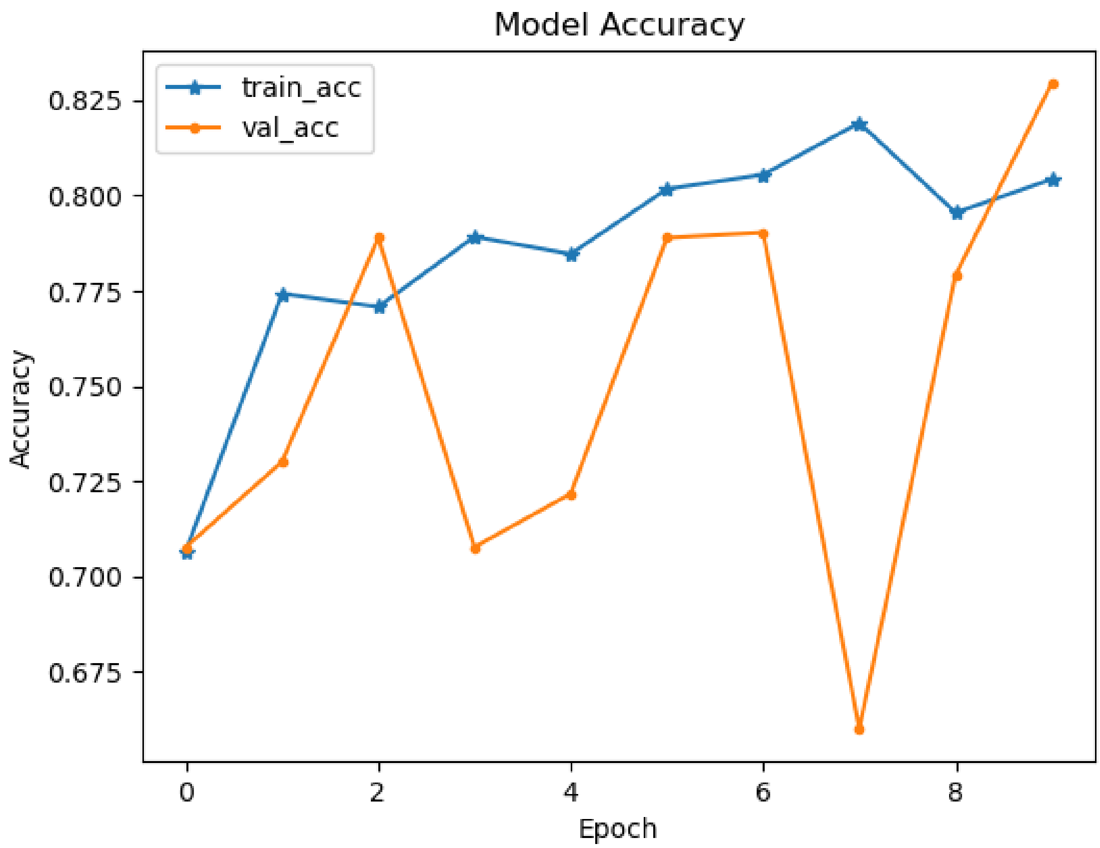
6. Plotting the training and validation loss

[8]:
```
plt.plot(result.history['loss'], label='train_loss',marker = '*')
plt.plot(result.history['val_loss'], label='val_loss',marker = '.')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



7. Plotting the training and validation accuracy

[9]:
```
plt.plot(result.history['accuracy'], label='train_acc',marker = '*')
plt.plot(result.history['val_accuracy'], label='val_acc',marker = '.')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



8. Save the model

[10]:
```
import os
os.chdir('D:/Casting')
```

[11]:
```
model.save('logistic_regression_model.h5')
```

9. Define a function to make predictions using the saved model

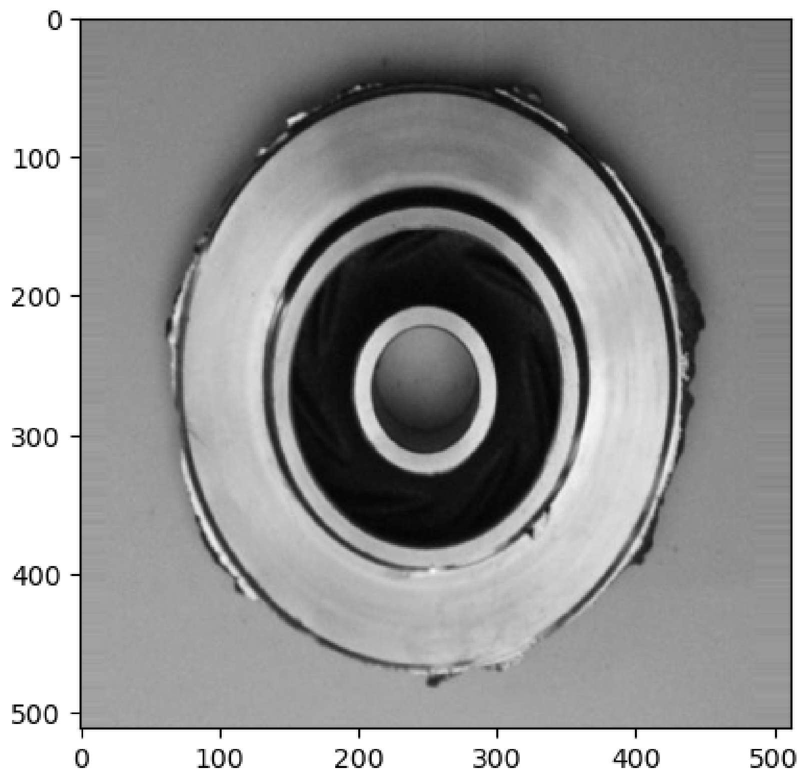
[12]:
```
def model_output(path):
    model = keras.models.load_model('logistic_regression_model.h5')
    img = keras.preprocessing.image.load_img(path, target_size=(64, 64))
    img_array = keras.preprocessing.image.img_to_array(img)
```

```
    img_array = tf.expand_dims(img_array, 0) / 255.
    prob = model.predict(img_array)[0][0]
    plt.imshow(plt.imread(path))
    print('Probability:', prob)
    if prob > 0.5:
        print("Casting is ok ")
    else:
        print("Casting is defective")
```

10. Use the function to make a prediction

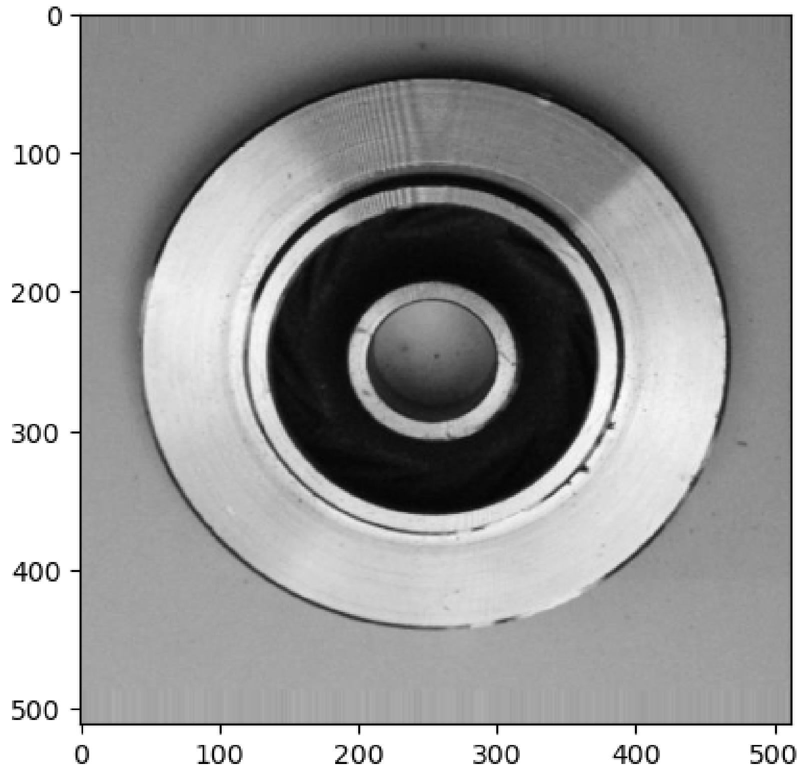
[13]:
```
model_output("D:\Casting\casting_image\def_front\cast_def_0_0.jpeg")
```

```
1/1 [==============================] - 0s 78ms/step
Probability: 0.009571619
Casting is defective
```



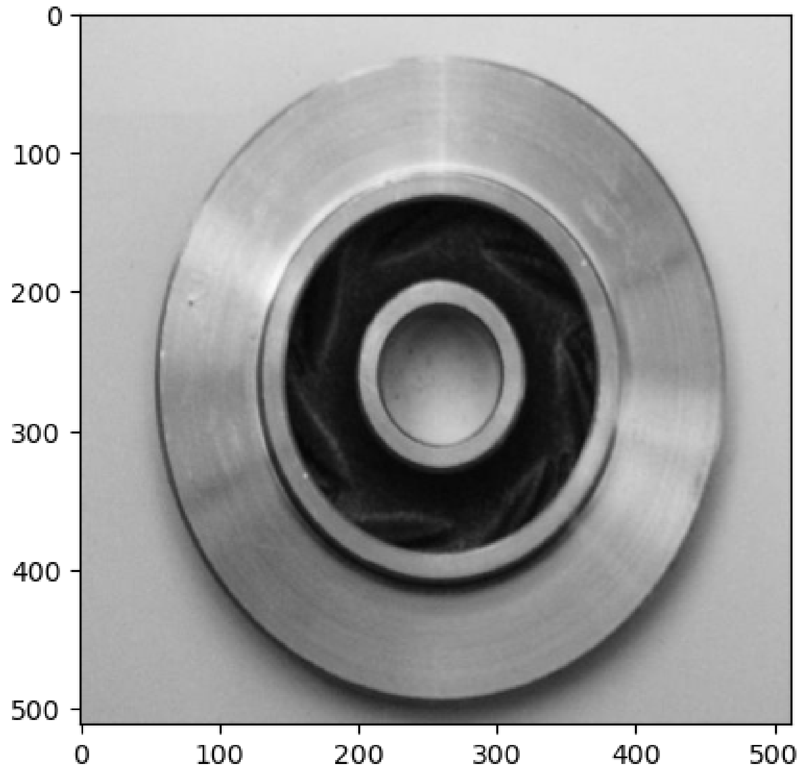
[14]:
```
model_output("D:\Casting\casting_512x512\def_front\cast_def_0_240.jpeg")
```

```
1/1 [==============================] - 0s 34ms/step
Probability: 0.14801562
Casting is defective
```



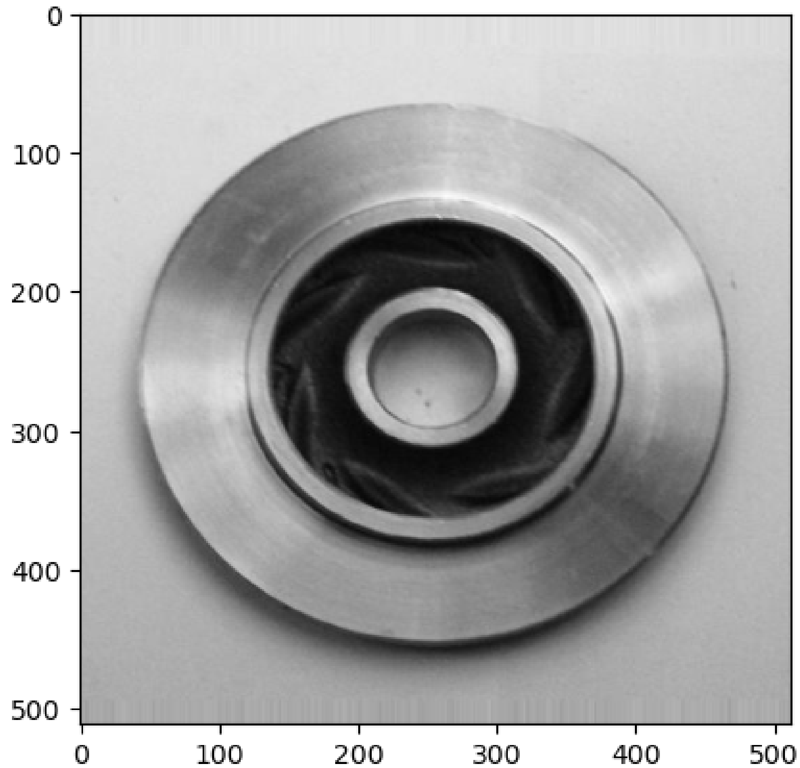
[15]:
```
model_output("D:\Casting\casting_512x512\ok_front\cast_ok_0_35.jpeg")
```

```
1/1 [==============================] - 0s 31ms/step
Probability: 0.9460394
Casting is ok
```



[16]:
```
model_output("D:\Casting\casting_512x512\ok_front\cast_ok_0_601.jpeg")
```

```
1/1 [==============================] - 0s 29ms/step
Probability: 0.9907937
Casting is ok
```


0
100
200
300
400
500
0
100
200
300
400
500

```
[ ]:
```

```
[ ]:
```

Metal-Casting-CNN

June 1, 2023

1. Importing the necessary libraries

[1]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras import Sequential
import matplotlib.pyplot as plt
%matplotlib inline
import cv2
```

2. Define the CNN model

[2]:
```
model = keras.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(64,64,3)),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Conv2D(32, (3,3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

3. Compile the model

[3]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 29, 29, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 32)       0
```

```
 2D)

 flatten (Flatten)           (None, 6272)             0

 dense (Dense)               (None, 128)              802944

 dense_1 (Dense)             (None, 1)                129

=================================================================
Total params: 813,217
Trainable params: 813,217
Non-trainable params: 0
_________________________________________________________________
```

4. Load and preprocess the data

[4]:
```python
train_datagen = keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
test_datagen = keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
```

[5]:
```python
training_set = train_datagen.flow_from_directory(
    r"D:\Casting\train",
    target_size=(64,64),
    batch_size=32,
    class_mode='binary'
)
```

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

[6]:
```python
test_set = test_datagen.flow_from_directory(
    r"D:\Casting\test",
    target_size=(64,64),
    batch_size=32,
    class_mode='binary'
)
```

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

5. Train the model

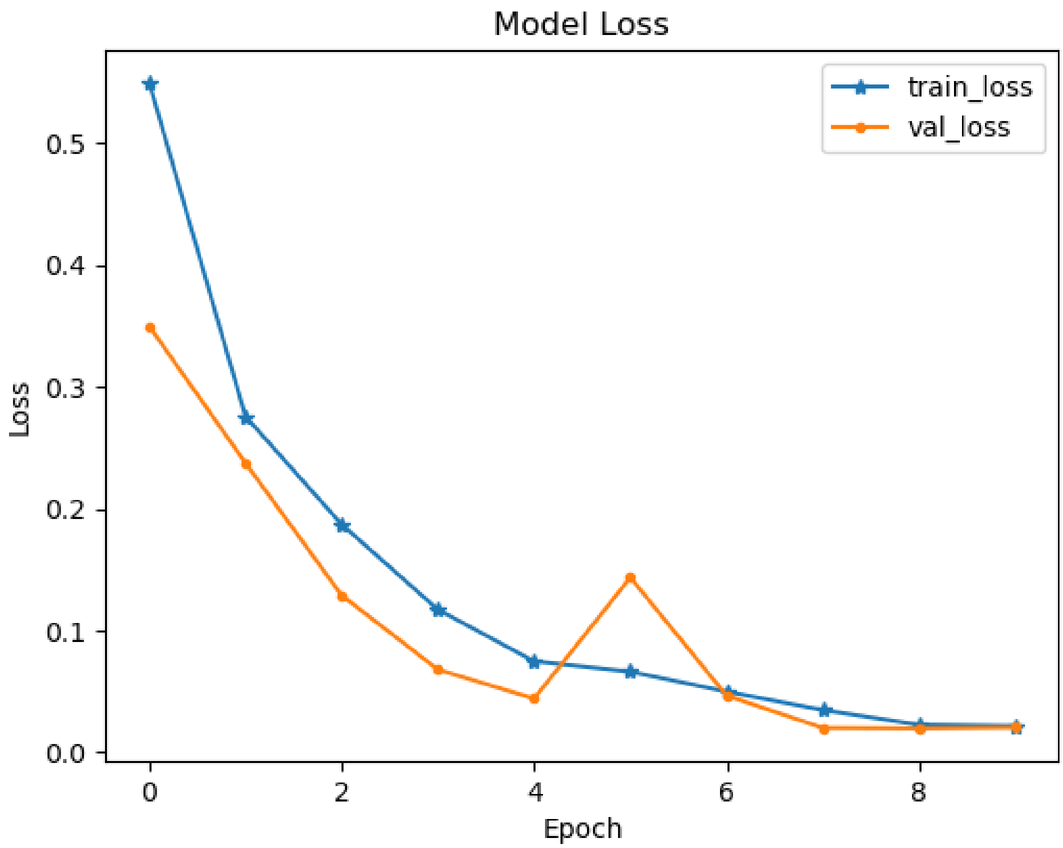
[7]:
```python
result = model.fit(
    training_set,
    epochs=10,
    validation_data=test_set
)
```

```
Epoch 1/10
208/208 [==============================] - 25s 117ms/step - loss: 0.5496 -
accuracy: 0.7021 - val_loss: 0.3498 - val_accuracy: 0.8378
Epoch 2/10
208/208 [==============================] - 24s 115ms/step - loss: 0.2758 -
```

```
accuracy: 0.8807 - val_loss: 0.2376 - val_accuracy: 0.8937
Epoch 3/10
208/208 [==============================] - 24s 116ms/step - loss: 0.1871 -
accuracy: 0.9276 - val_loss: 0.1289 - val_accuracy: 0.9524
Epoch 4/10
208/208 [==============================] - 23s 108ms/step - loss: 0.1172 -
accuracy: 0.9569 - val_loss: 0.0681 - val_accuracy: 0.9790
Epoch 5/10
208/208 [==============================] - 22s 104ms/step - loss: 0.0748 -
accuracy: 0.9768 - val_loss: 0.0442 - val_accuracy: 0.9832
Epoch 6/10
208/208 [==============================] - 22s 103ms/step - loss: 0.0663 -
accuracy: 0.9803 - val_loss: 0.1436 - val_accuracy: 0.9371
Epoch 7/10
208/208 [==============================] - 22s 104ms/step - loss: 0.0500 -
accuracy: 0.9852 - val_loss: 0.0467 - val_accuracy: 0.9804
Epoch 8/10
208/208 [==============================] - 22s 104ms/step - loss: 0.0349 -
accuracy: 0.9890 - val_loss: 0.0202 - val_accuracy: 0.9930
Epoch 9/10
208/208 [==============================] - 22s 104ms/step - loss: 0.0227 -
accuracy: 0.9944 - val_loss: 0.0198 - val_accuracy: 0.9972
Epoch 10/10
208/208 [==============================] - 22s 104ms/step - loss: 0.0217 -
accuracy: 0.9934 - val_loss: 0.0205 - val_accuracy: 0.9916
```

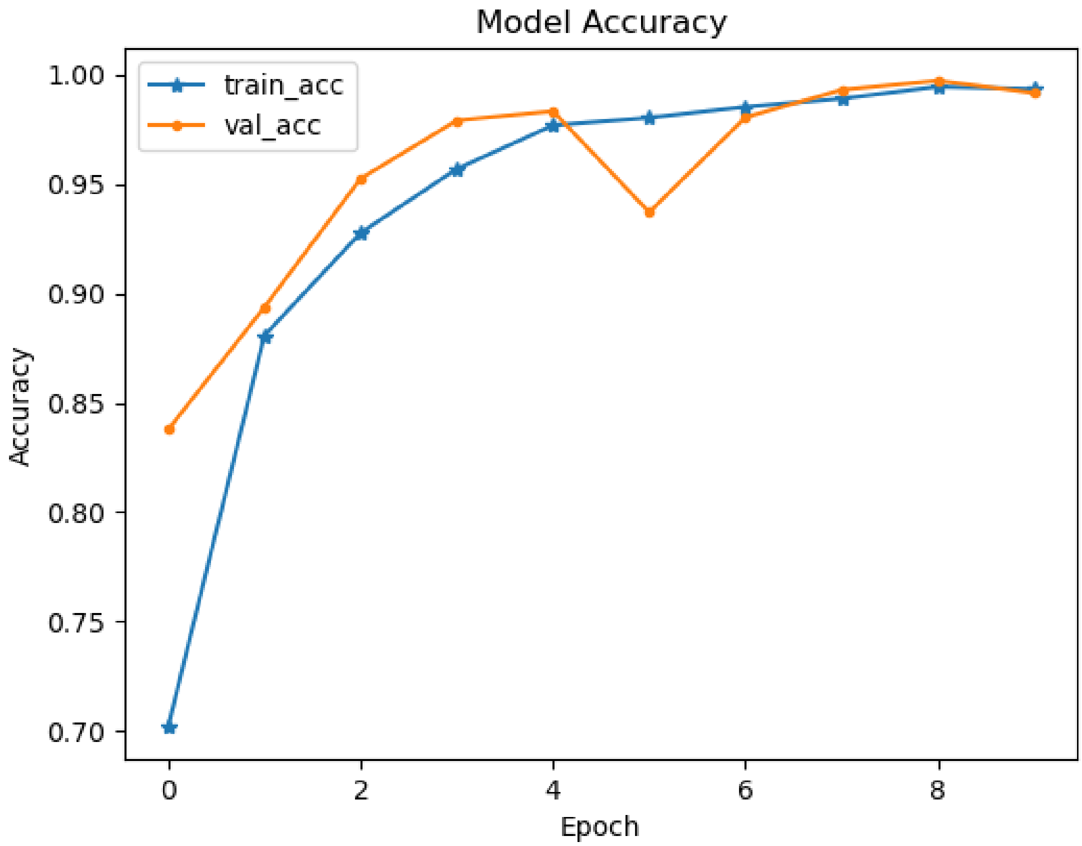
6. Plotting the training and validation loss

[8]:
```python
import matplotlib.pyplot as plt

# Plot the training and validation loss
plt.plot(result.history['loss'], label='train_loss', marker = '*')
plt.plot(result.history['val_loss'], label='val_loss',marker= '.')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



7. Plotting the training and validation accuracy

[9]:
```
# Plot the training and validation accuracy
plt.plot(result.history['accuracy'], label='train_acc' , marker = '*')
plt.plot(result.history['val_accuracy'], label='val_acc' , marker = '.')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



8. Save the model

[10]:
```
import os
os.chdir('D:/Casting')
```

[11]:
```
model.save('casting_classifiers.h5')
```

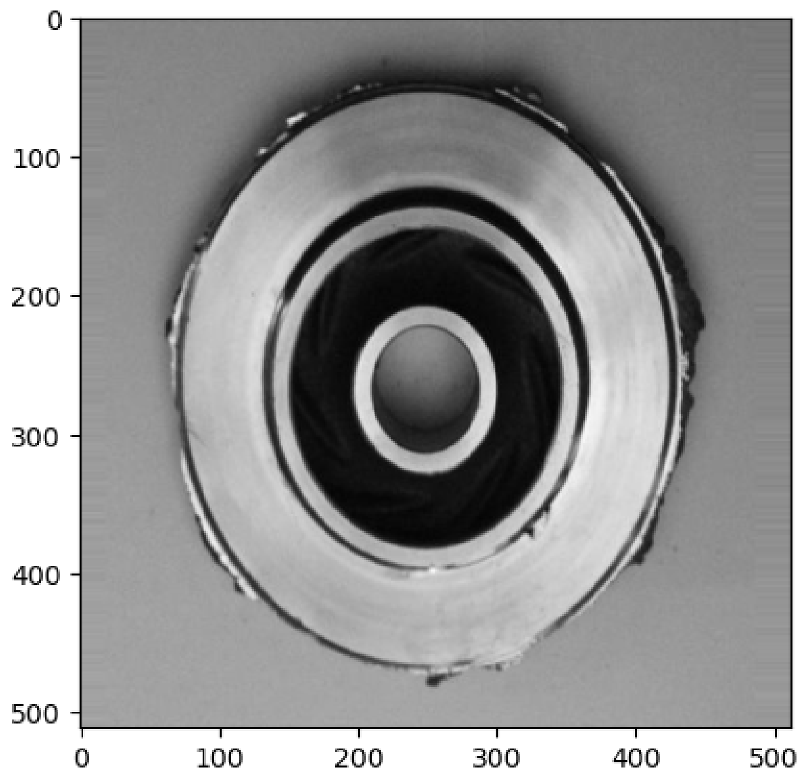
9. Define a function to make predictions using the saved model

[12]:
```
def model_output(path):
    model = keras.models.load_model('casting_classifiers.h5')
    img = keras.preprocessing.image.load_img(path, target_size=(64,64))
    img_array = keras.preprocessing.image.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0) / 255.
    prob = model.predict(img_array)[0][0]
    plt.imshow(cv2.imread(path))
    print('Probability:', prob)
    if prob > 0.5:
        print("Casting is ok")
    else:
        print("Casting is defective")
```

10. Use the function to make a prediction

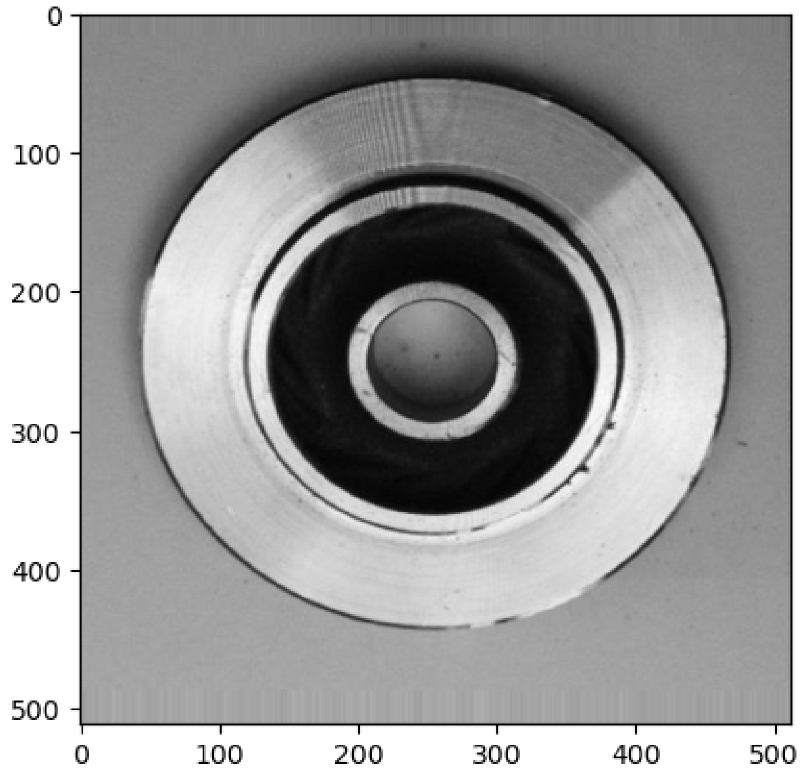
[13]:
```
model_output("D:\Casting\casting_image\def_front\cast_def_0_0.jpeg")
```

```
1/1 [==============================] - 0s 94ms/step
Probability: 0.002048741
Casting is defective
```



[14]:
```
model_output("D:\Casting\casting_512x512\def_front\cast_def_0_240.jpeg")
```

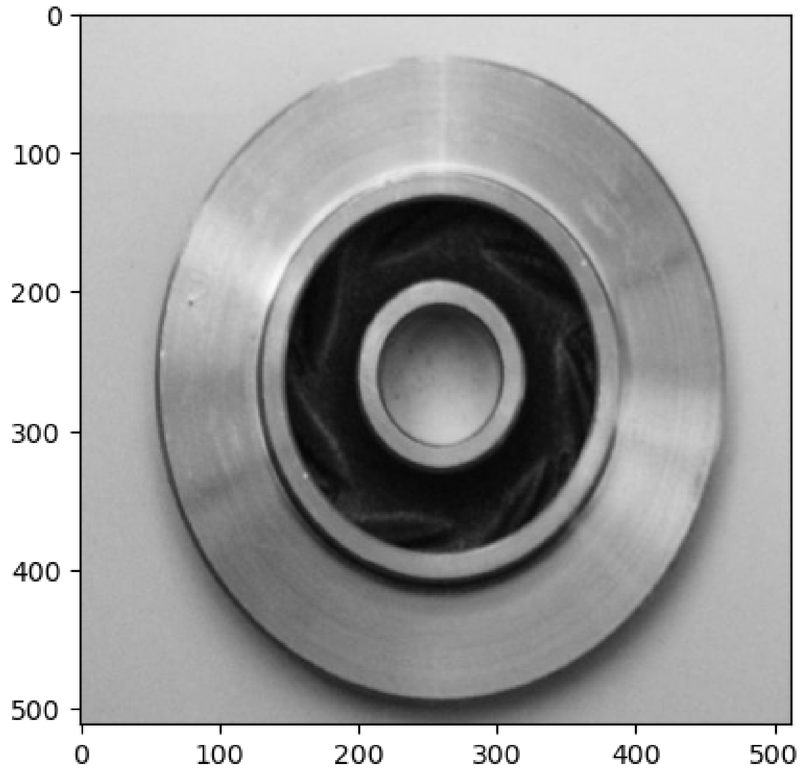
```
1/1 [==============================] - 0s 62ms/step
Probability: 0.0013744961
Casting is defective
```



[15]:

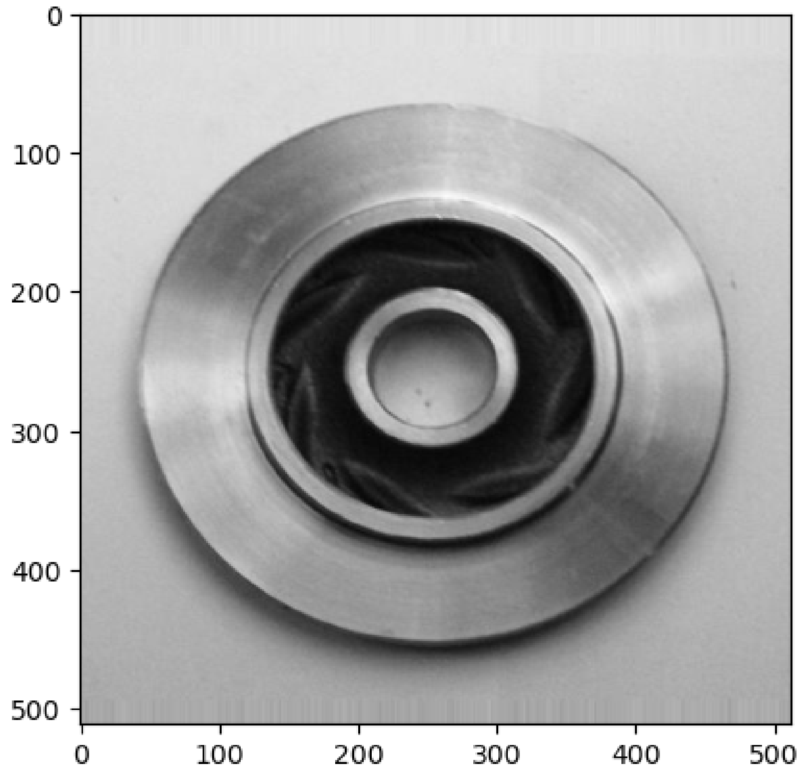
```
model_output("D:\Casting\casting_512x512\ok_front\cast_ok_0_35.jpeg")
```

```
1/1 [==============================] - 0s 62ms/step
Probability: 0.99630225
Casting is ok
```



[16]:
```
model_output("D:\Casting\casting_512x512\ok_front\cast_ok_0_601.jpeg")
```

```
1/1 [==============================] - 0s 47ms/step
Probability: 0.99971014
Casting is ok
```


0
100
200
300
400
500
0
100
200
300
400
500

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```