Identification of Casting defects using Deep learning

The aim of the project is to identify the metal casting defects and classify them into defective and non-defective images using CNN

In [2]:
```
#Importing the libraries
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import image
from matplotlib.image import imread
%matplotlib inline
```

In [4]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import models, layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Flatten, Dense
from tensorflow.keras.layers import Conv2D, MaxPooling2D
```

Dataset:

Image size

In [5]:
```
image_size = (300,300,3) #Image size of the defective and non-defective parts
 in casting data
```

Images of few defective parts in the dataset:

Image 1

In [6]:
```
#Image of a defective part - Image1
defective_train_image1 = plt.imread('C:/Users/admin/data/train/defective/cast_def_0_5034.jpeg')
plt.imshow(defective_train_image1)
```

Out[6]: <matplotlib.image.AxesImage at 0x1dd46db5400>

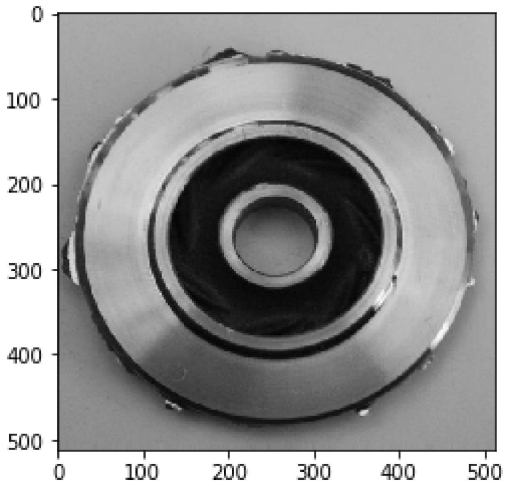


Image 2

In [7]:
```
#Image of a defective part - Image 2
defective_train_image2 = plt.imread('C:/Users/admin/data/train/defective/cast_def_0_9817.jpeg')
plt.imshow(defective_train_image2)
```

Out[7]: <matplotlib.image.AxesImage at 0x1dd46e5c470>

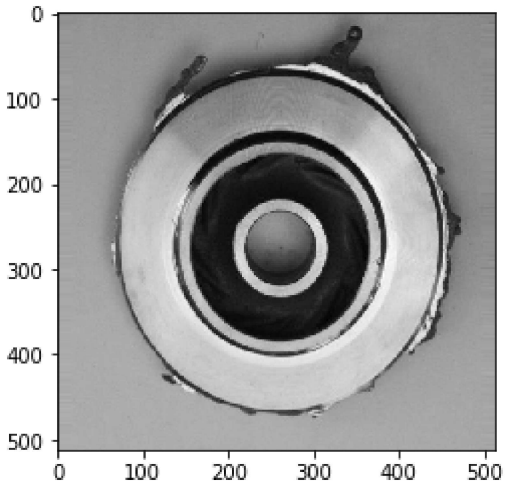


Image 3

In [8]:
```
#Image of a defective part - Image 3
defective_train_image3 = plt.imread('C:/Users/admin/data/train/defective/cast_
def_0_9328.jpeg')
plt.imshow(defective_train_image3)
```

Out[8]: <matplotlib.image.AxesImage at 0x1dd46ebae48>

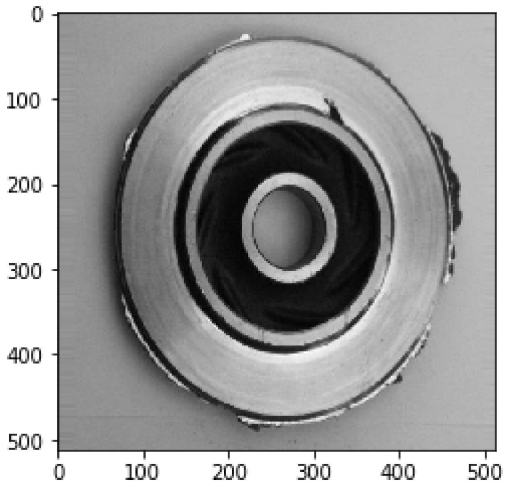


Image 4

In [9]:
```
#Image of a defective part - Image 4
defective_train_image4 = plt.imread('C:/Users/admin/data/train/defective/cast_
def_0_9051.jpeg')
plt.imshow(defective_train_image4)
```

Out[9]: <matplotlib.image.AxesImage at 0x1dd46f21390>

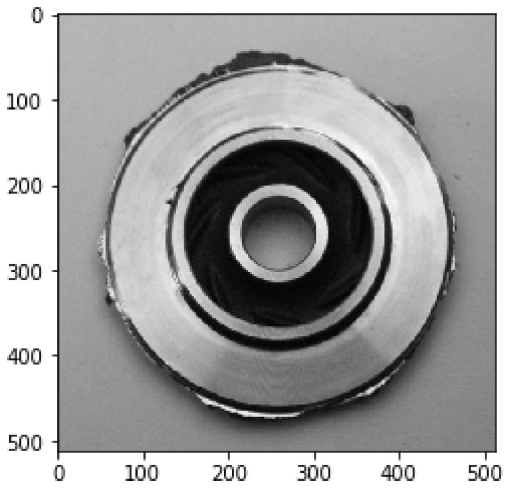
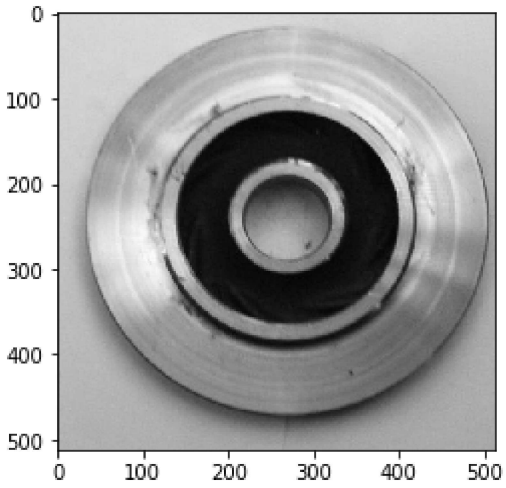


Image 5

In [10]:
```
#Image of a defective part - Image 5
defective_train_image5 = plt.imread('C:/Users/admin/data/train/defective/cast_
def_0_8467.jpeg')
plt.imshow(defective_train_image5)
```

Out[10]:
```
<matplotlib.image.AxesImage at 0x1dd46f7c828>
```



Images of few non-defective parts in the dataset:

Image 1

In [11]:
```
#Image of a non-defective part - Image 1
non_defective_train_image1 = plt.imread('C:/Users/admin/data/train/non_defecti
ve/cast_ok_0_9193.jpeg')
plt.imshow(non_defective_train_image1)
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x1dd46fde0f0>
```

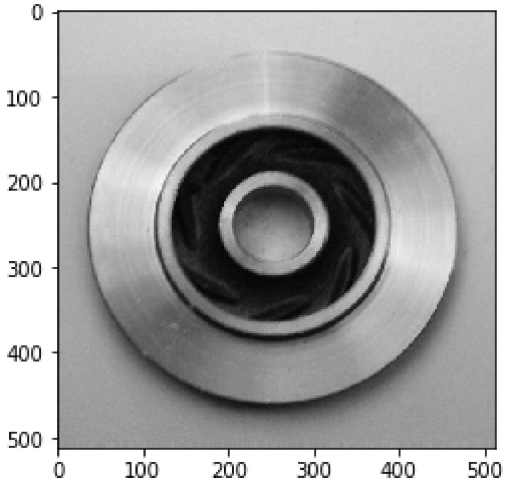


Image 2

In [12]:
```
#Image of a non-defective part - Image 2
non_defective_train_image2 = plt.imread('C:/Users/admin/data/train/non_defecti
ve/cast_ok_0_9734.jpeg')
plt.imshow(non_defective_train_image2)
```

Out[12]: <matplotlib.image.AxesImage at 0x1dd4800c240>

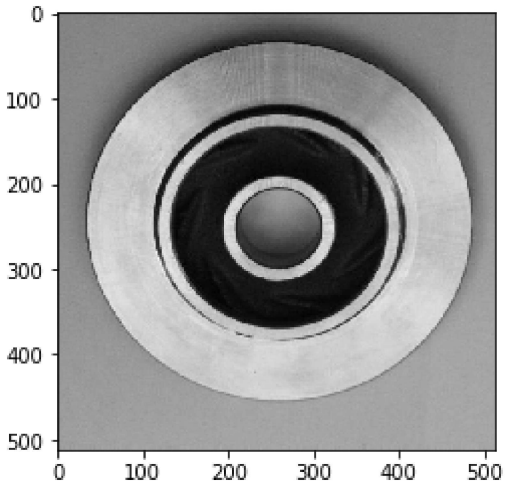


Image 3

In [13]:
```
#Image of a non-defective part - Image 3
non_defective_train_image3 = plt.imread('C:/Users/admin/data/train/non_defecti
ve/cast_ok_0_9658.jpeg')
plt.imshow(non_defective_train_image3)
```

Out[13]: <matplotlib.image.AxesImage at 0x1dd48068390>

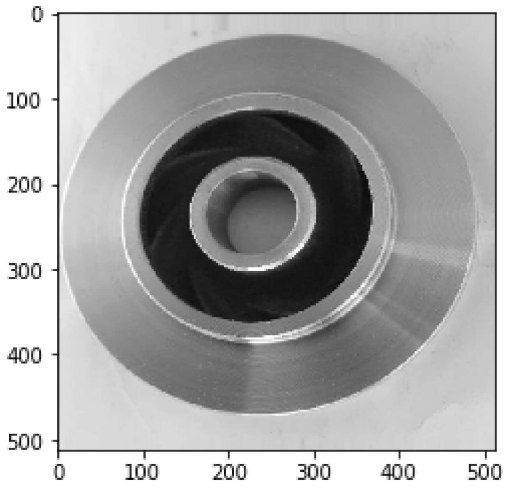


Image 4

In [14]:
```
#Image of a non-defective part - Image 4
non_defective_train_image4 = plt.imread('C:/Users/admin/data/train/non_defecti
ve/cast_ok_0_7673.jpeg')
plt.imshow(non_defective_train_image4)
```

Out[14]: <matplotlib.image.AxesImage at 0x1dd480c54e0>

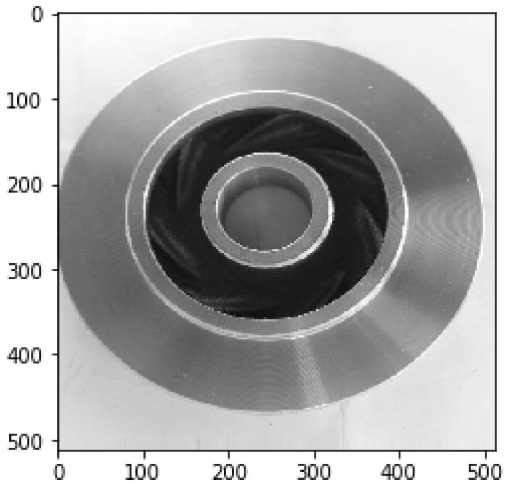
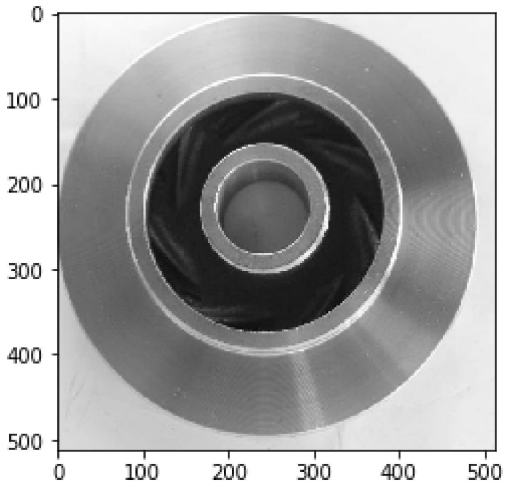


Image 5

In [15]:
```
#Image of a non-defective part - Image 5
non_defective_train_image5 = plt.imread('C:/Users/admin/data/train/non_defecti
ve/cast_ok_0_8723.jpeg')
plt.imshow(non_defective_train_image5)
```

Out[15]: <matplotlib.image.AxesImage at 0x1dd48123630>



Training data and testing data

In [16]:
```
#Function for train and test data generator
train_data_generator = ImageDataGenerator(rescale=1/255)
test_data_generator = ImageDataGenerator(rescale=1/255)
#Function for batch training for training and testing data
train_data = train_data_generator.flow_from_directory('C:/Users/admin/data/tra
in', target_size=(300,300), batch_size=2, class_mode='binary')
test_data = test_data_generator.flow_from_directory('C:/Users/admin/data/test'
, target_size=(300,300), batch_size=2, class_mode='binary')
```

```
Found 1120 images belonging to 2 classes.
Found 180 images belonging to 2 classes.
```

Training data

In [17]:
```
#Number of images in training data after batch training
train_data_len = len(train_data)
train_data_len
```

Out[17]:
```
560
```

Categories of training data

In [18]:
```
#Categories of images in training data
train_data.class_indices
```

Out[18]:
```
{'defective': 0, 'non_defective': 1}
```

Testing data

In [19]:
```
#Number of images in testing data
test_data_len = len(test_data)
test_data_len
```

Out[19]:
```
90
```

Categories of testing data

In [20]:
```
#Categories of images in testing data
test_data.class_indices
```

Out[20]:
```
{'defective': 0, 'non_defective': 1}
```

Deep learning method:

Convolutional Neural Network(CNN) algorithm

In [21]:
```
#CNN network
model = Sequential()
#Convolutional 2D Layer
model.add(Conv2D(10, (3,3),input_shape=image_size, activation='relu', name="Co
nvolutional_2D_Layer1"))
#Max Pooling Layer
model.add(layers.MaxPooling2D((2, 2), name="Max_Pooling_Layer1"))
#Convolutional 2D Layer
model.add(Conv2D(10, (3,3),input_shape=image_size, activation='relu', name="Co
nvolutional_2D_Layer2"))
#Max Pooling Layer
model.add(layers.MaxPooling2D((2, 2), name="Max_Pooling_Layer2"))
#Convolutional 2D Layer
model.add(Conv2D(10, (3,3),input_shape=image_size, activation='relu', name="Co
nvolutional_2D_Layer3"))
#Max Pooling Layer
model.add(layers.MaxPooling2D((2, 2), name="Max_Pooling_Layer3"))
#Convolutional 2D Layer
model.add(Conv2D(10, (3,3),input_shape=image_size, activation='relu', name="Co
nvolutional_2D_Layer4"))
#Max Pooling Layer
model.add(layers.MaxPooling2D((2, 2), name="Max_Pooling_Layer4"))
#Flatten Layer
model.add(Flatten(name="Flatten_Layer"))
#Dense Layer
model.add(Dense(128, activation='relu', name="Dense_Layer1"))
model.add(Dense(1, activation='sigmoid',name="Dense_Layer2"))
```

```
WARNING:tensorflow:From C:\Users\Janani SR\Anaconda3\lib\site-packages\tensor
flow\python\ops\resource_variable_ops.py:435: colocate_with (from tensorflow.
python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

Compiling using Stochastic Gradient Descent(SGD) optimizer

In [22]:
```
#Compiling the model using the parameters using loss, optimizer and metrics pa
rameters
model.compile(loss='binary_crossentropy', optimizer='SGD', metrics=['accuracy'
])
```

Summary with CNN parameters

In [23]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Convolutional_2D_Layer1 (Con (None, 298, 298, 10)      280
_________________________________________________________________
Max_Pooling_Layer1 (MaxPooli (None, 149, 149, 10)      0
_________________________________________________________________
Convolutional_2D_Layer2 (Con (None, 147, 147, 10)      910
_________________________________________________________________
Max_Pooling_Layer2 (MaxPooli (None, 73, 73, 10)        0
_________________________________________________________________
Convolutional_2D_Layer3 (Con (None, 71, 71, 10)        910
_________________________________________________________________
Max_Pooling_Layer3 (MaxPooli (None, 35, 35, 10)        0
_________________________________________________________________
Convolutional_2D_Layer4 (Con (None, 33, 33, 10)        910
_________________________________________________________________
Max_Pooling_Layer4 (MaxPooli (None, 16, 16, 10)        0
_________________________________________________________________
Flatten_Layer (Flatten)      (None, 2560)              0
_________________________________________________________________
Dense_Layer1 (Dense)         (None, 128)               327808
_________________________________________________________________
Dense_Layer2 (Dense)         (None, 1)                 129
=================================================================
Total params: 330,947
Trainable params: 330,947
Non-trainable params: 0
_________________________________________________________________
```

Training for 25 epochs

In [24]:
```
#Fitting the model and training
results = model.fit_generator(train_data, epochs=25, validation_data=test_data
)
```

```
WARNING:tensorflow:From C:\Users\Janani SR\Anaconda3\lib\site-packages\tensor
flow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_o
ps) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/25
90/90 [==============================] - 10s 116ms/step - loss: 0.5398 - acc:
0.7333
560/560 [==============================] - 150s 269ms/step - loss: 0.6082 - a
cc: 0.6875 - val_loss: 0.5398 - val_acc: 0.7333
Epoch 2/25
90/90 [==============================] - 10s 115ms/step - loss: 0.4888 - acc:
0.7389
560/560 [==============================] - 148s 265ms/step - loss: 0.5245 - a
cc: 0.7366 - val_loss: 0.4888 - val_acc: 0.7389
Epoch 3/25
90/90 [==============================] - 11s 120ms/step - loss: 0.4048 - acc:
0.8167
560/560 [==============================] - 149s 267ms/step - loss: 0.4560 - a
cc: 0.7920 - val_loss: 0.4048 - val_acc: 0.8167
Epoch 4/25
90/90 [==============================] - 11s 117ms/step - loss: 0.4086 - acc:
0.7833
560/560 [==============================] - 150s 267ms/step - loss: 0.4210 - a
cc: 0.8027 - val_loss: 0.4086 - val_acc: 0.7833
Epoch 5/25
90/90 [==============================] - 10s 116ms/step - loss: 0.3269 - acc:
0.8222
560/560 [==============================] - 149s 266ms/step - loss: 0.3719 - a
cc: 0.8179 - val_loss: 0.3269 - val_acc: 0.8222
Epoch 6/25
90/90 [==============================] - 10s 113ms/step - loss: 0.3281 - acc:
0.8556
560/560 [==============================] - 147s 263ms/step - loss: 0.3298 - a
cc: 0.8464 - val_loss: 0.3281 - val_acc: 0.8556
Epoch 7/25
90/90 [==============================] - 10s 113ms/step - loss: 0.4774 - acc:
0.8111
560/560 [==============================] - 146s 261ms/step - loss: 0.2846 - a
cc: 0.8857 - val_loss: 0.4774 - val_acc: 0.8111
Epoch 8/25
90/90 [==============================] - 10s 112ms/step - loss: 0.2417 - acc:
0.8944
560/560 [==============================] - 146s 261ms/step - loss: 0.2352 - a
cc: 0.9045 - val_loss: 0.2417 - val_acc: 0.8944
Epoch 9/25
90/90 [==============================] - 10s 112ms/step - loss: 0.6045 - acc:
0.6222
560/560 [==============================] - 144s 258ms/step - loss: 0.6362 - a
cc: 0.6196 - val_loss: 0.6045 - val_acc: 0.6222
Epoch 10/25
90/90 [==============================] - 10s 114ms/step - loss: 0.1943 - acc:
0.9111
560/560 [==============================] - 146s 261ms/step - loss: 0.2917 - a
cc: 0.8670 - val_loss: 0.1943 - val_acc: 0.9111
Epoch 11/25
90/90 [==============================] - 10s 112ms/step - loss: 0.4272 - acc:
```

```
0.8667
560/560 [==============================] - 146s 261ms/step - loss: 0.3297 - a
cc: 0.8554 - val_loss: 0.4272 - val_acc: 0.8667
Epoch 12/25
90/90 [==============================] - 10s 116ms/step - loss: 0.5550 - acc:
0.6944
560/560 [==============================] - 147s 262ms/step - loss: 0.1763 - a
cc: 0.9259 - val_loss: 0.5550 - val_acc: 0.6944
Epoch 13/25
90/90 [==============================] - 11s 117ms/step - loss: 0.2495 - acc:
0.8889
560/560 [==============================] - 148s 264ms/step - loss: 0.1671 - a
cc: 0.9420 - val_loss: 0.2495 - val_acc: 0.8889
Epoch 14/25
90/90 [==============================] - 10s 116ms/step - loss: 0.3032 - acc:
0.9000
560/560 [==============================] - 147s 262ms/step - loss: 0.2394 - a
cc: 0.8964 - val_loss: 0.3032 - val_acc: 0.9000
Epoch 15/25
90/90 [==============================] - 10s 114ms/step - loss: 0.2940 - acc:
0.9056
560/560 [==============================] - 147s 263ms/step - loss: 0.1126 - a
cc: 0.9473 - val_loss: 0.2940 - val_acc: 0.9056
Epoch 16/25
90/90 [==============================] - 10s 115ms/step - loss: 0.2153 - acc:
0.9222
560/560 [==============================] - 148s 265ms/step - loss: 0.1253 - a
cc: 0.9500 - val_loss: 0.2153 - val_acc: 0.9222
Epoch 17/25
90/90 [==============================] - 10s 115ms/step - loss: 0.1657 - acc:
0.9333
560/560 [==============================] - 147s 262ms/step - loss: 0.0857 - a
cc: 0.9723 - val_loss: 0.1657 - val_acc: 0.9333
Epoch 18/25
90/90 [==============================] - 10s 114ms/step - loss: 0.3293 - acc:
0.8778
560/560 [==============================] - 147s 262ms/step - loss: 0.0352 - a
cc: 0.9884 - val_loss: 0.3293 - val_acc: 0.8778
Epoch 19/25
90/90 [==============================] - 10s 116ms/step - loss: 0.1818 - acc:
0.9278
560/560 [==============================] - 147s 263ms/step - loss: 0.0839 - a
cc: 0.9714 - val_loss: 0.1818 - val_acc: 0.9278
Epoch 20/25
90/90 [==============================] - 10s 117ms/step - loss: 0.2020 - acc:
0.9278
560/560 [==============================] - 157s 280ms/step - loss: 0.0717 - a
cc: 0.9741 - val_loss: 0.2020 - val_acc: 0.9278
Epoch 21/25
90/90 [==============================] - 10s 114ms/step - loss: 0.3326 - acc:
0.9111
560/560 [==============================] - 147s 262ms/step - loss: 0.0436 - a
cc: 0.9911 - val_loss: 0.3326 - val_acc: 0.9111
Epoch 22/25
90/90 [==============================] - 10s 115ms/step - loss: 0.1641 - acc:
0.9500
560/560 [==============================] - 147s 262ms/step - loss: 0.0657 - a
```

```
cc: 0.9795 - val_loss: 0.1641 - val_acc: 0.9500
Epoch 23/25
90/90 [==============================] - 11s 117ms/step - loss: 0.1561 - acc:
0.9556
560/560 [==============================] - 148s 264ms/step - loss: 0.0092 - a
cc: 0.9964 - val_loss: 0.1561 - val_acc: 0.9556
Epoch 24/25
90/90 [==============================] - 10s 111ms/step - loss: 0.1755 - acc:
0.9556
560/560 [==============================] - 147s 262ms/step - loss: 9.0270e-04
- acc: 1.0000 - val_loss: 0.1755 - val_acc: 0.9556
Epoch 25/25
90/90 [==============================] - 10s 111ms/step - loss: 0.1908 - acc:
0.9500
560/560 [==============================] - 146s 260ms/step - loss: 4.7215e-04
- acc: 1.0000 - val_loss: 0.1908 - val_acc: 0.9500
```

Results:

Accuracy of training data and testing data

In [25]:
```
#Plot showing the accuracy of training and testing data
plt.title('Train data and test data accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
#Plot for training data accuracy
plt.plot(results.history['acc'], color="red")
#Plot for testing data accuracy
plt.plot(results.history['val_acc'], color="blue")
plt.legend(["Training Data", "Testing Data"], loc = "lower right")
plt.show()
```

Loss of training data and testing data

In [26]:
```
#Plot showing the loss of training data and testing data
plt.title('Train data and test data loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
#Plot for training data loss
plt.plot(results.history['loss'], color="red")
#Plot for testing data loss
plt.plot(results.history['val_loss'], color="blue")
plt.legend(["Training Data", "Testing Data"], loc = "upper right")
plt.show()
```