Code Explanation:

This code creates a **Streamlit web application** for detecting corrosion in steel members using a **pre-trained EfficientNetB0 model**. The application processes uploaded images of steel members, identifies the corrosion regions, and classifies the images into predefined categories, providing the user with both the classification results and visual indications of the corroded areas. Below is a detailed explanation of each section of the code.

---

1. Initial Streamlit Configuration:

```
st.set_page_config(page_title="Corrosion Detection", page_icon="🔧")
```

- This line sets up the initial configuration for the Streamlit web page. The page title is set to "Corrosion Detection" and the page icon is set to a wrench emoji ("🔧").

---

2. Loading the Pre-trained EfficientNetB0 Model:

```
model = EfficientNetB0(weights='imagenet')
```

- Here, the **EfficientNetB0** model, pre-trained on the **ImageNet** dataset, is loaded. EfficientNetB0 is a deep convolutional neural network (CNN) that has been pre-trained on a large dataset, making it suitable for general image classification tasks.

---

3. Prediction Function:

```
def predict_image(img_path):
    # Load and preprocess the image
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = tf.keras.applications.efficientnet.preprocess_input(img_array)

    # Make prediction
    prediction = model.predict(img_array)

    return prediction
```

- This function processes the uploaded image and prepares it for prediction using the pre-trained model.
- The image is loaded and resized to the required input size of the EfficientNetB0 model (224x224 pixels).
- It is then converted into an array, expanded to match the expected input shape, and preprocessed for the EfficientNet model.
- The model makes a prediction and returns the result.

---

4. Decoding the Prediction:

```
from tensorflow.keras.applications.efficientnet import decode_predictions
def get_class_prediction(prediction):
    decoded = decode_predictions(prediction, top=1)[0]
    return decoded[0]
```

- This function decodes the raw prediction output of the model into human-readable class labels and their associated probabilities.
- The `decode_predictions` function from Keras decodes the predictions and returns the top 1 prediction with its class label and probability.

---

5. Streamlit User Interface:

```
st.title('Corrosion Detection Web App’)
st.write("This web application is designed to detect and assess corrosion in
steel members. Please upload one or more images.")
```

- These lines display the title and description of the web application on the webpage.

---

6. Image Upload Widget:

```
uploaded_files = st.file_uploader("Choose steel member images", type=["jpg",
"png", "jpeg"], accept_multiple_files=True)
```

- This creates a file uploader widget that allows users to upload one or more images of steel members in **jpg**, **png**, or **jpeg** format.

---

7. Temporary Folder for Storing Uploaded Files:

```
temp_folder = './temp/'
if not os.path.exists(temp_folder):
    os.makedirs(temp_folder)

img_path = os.path.join(temp_folder, uploaded_file.name)
with open(img_path, "wb") as f:
    f.write(uploaded_file.getbuffer())
```

- This section ensures that the uploaded files are saved in a temporary folder (`temp/`). If the folder doesn't exist, it is created. The uploaded file is saved with its original name in the temporary folder.

---

8. Processing and Displaying the Results:

```
prediction = predict_image(img_path)
class_label, class_name, class_probability = get_class_prediction(prediction)
```

- This part uses the `predict_image` function to make predictions on the uploaded image.
- It then decodes the prediction to retrieve the class label (the detected object) and its probability.

---

9. Displaying the Original Image and Prediction Results:

```
st.image(uploaded_file, caption="Original Image", use_column_width=True)
st.write(f"Predicted Class: {class_name}")
st.write(f"Prediction Probability: {class_probability:.2f}")
```

- The original uploaded image is displayed on the app along with the predicted class label and its corresponding probability.

---

10. Displaying the Processed Image for Corrosion Detection:

```
img_opened = cv2.imread(img_path)
gray = cv2.cvtColor(img_opened, cv2.COLOR_BGR2GRAY)
_, thresh = cv2.threshold(gray, 100, 255, cv2.THRESH_BINARY)
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

for contour in contours:
    if cv2.contourArea(contour) > 500:
        cv2.drawContours(img_opened, [contour], -1, (0, 255, 0), 3)

st.image(img_opened, caption="Processed Image (Corrosion Areas Highlighted)",
use_column_width=True)
```

- In this section, image processing techniques like **Thresholding** and **Contour Detection** are used to identify the corrosion areas.
- The image is first converted to grayscale and thresholding is applied to highlight areas with high contrast (potential corrosion).
- Contours of the corrosion areas are detected and drawn on the image in green color, which is then displayed as the processed image.

---

11. Running the Application:

```
if __name__ == "__main__":
    st.write("The web application is running...")
```

- This block ensures that the web application is being executed when the script is run. It provides a message indicating that the app is currently running.

Conclusion:

This Streamlit web application allows users to upload images of steel members for corrosion detection. The uploaded image is processed to classify the corrosion level using a pre-trained **EfficientNetB0** model. Additionally, image processing techniques such as **Thresholding** and **Contour Detection** are applied to visually highlight the corroded regions. The user is provided with both the class prediction and visual feedback about the corrosion areas in the images.