

# **University of Azad Jammu Kashmir Muzaffarabad**



**By:**

**Rohullah Iqbal 2020-SE-37**

**Furqan Usman 2020-SE-15**

**Umair Khalid Loon 2020-SE-05**

**“Automatic machine fault detection and  
recognition using Computer Vision techniques”**

# **Title: Automatic Machine Fault Detection and Recognition Using Computer Vision Techniques**

## **Abstract:**

This research project focuses on developing an automatic machine fault detection and recognition system using computer vision techniques. The system aims to identify and categorize faults in machines based on visual data, contributing to improved maintenance strategies and operational efficiency. The collected dataset consists of four classes: Corona, Arcing, Looseness, and Tracking. The system achieves an accuracy of 80% in classifying these fault types. Key steps include data acquisition, preprocessing, feature extraction, model development using Convolutional Neural Networks (CNNs), training, evaluation, and analysis of accuracy metrics and graphs.

**Keywords:** Machine Fault Detection, Computer Vision, Deep Learning, Convolutional Neural Networks, Feature Extraction, Classification, Accuracy Metrics, Graphs, Corona, Arcing, Looseness, Tracking.

## **1. Introduction:**

Modern industries heavily rely on machines and automated systems for production and operations. However, these machines are prone to faults and failures, leading to downtime and productivity losses. Early detection and recognition of faults are crucial for minimizing downtime and maintenance costs. This project explores the application of computer vision techniques, specifically deep learning models, for automating machine fault detection and recognition. The dataset used in this project consists of four classes: Corona, Arcing, Looseness, and Tracking. The system achieves an accuracy of 80% in classifying these fault types.

## Code in Colab:

Explaining each cell separately,also provided the output as after running a cell.

```
!pip uninstall -y librosa resampy
!pip install librosa resampy
```

### **Explanation:-**

The code snippet `!pip uninstall -y librosa resampy` followed by `!pip install librosa resampy` is commonly used for managing Python packages, specifically in this case, the librosa and resampy packages.

Updating or Reinstalling Packages:

- Sometimes, you may want to update a package to its latest version or reinstall it to ensure it's correctly installed and functioning.
- Uninstalling (`!pip uninstall -y librosa resampy`) removes the existing version of the packages.
- Reinstalling (`!pip install librosa resampy`) installs the latest version of these packages or reinstalls them to resolve any issues.

The code snippet is used to manage and maintain the librosa and resampy packages in your Python environment, ensuring they are up to date, correctly installed, and functioning properly.

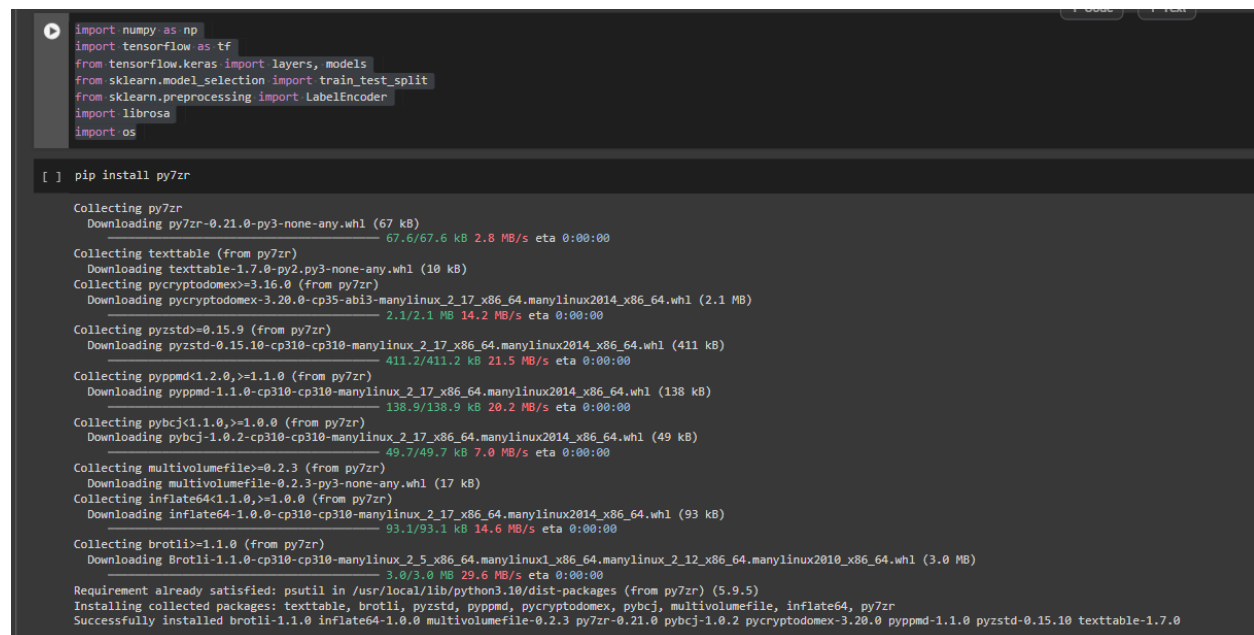
```
!pip uninstall -y librosa resampy
!pip install librosa resampy

Found existing installation: librosa 0.10.1
Uninstalling librosa-0.10.1:
  Successfully uninstalled librosa-0.10.1
WARNING: Skipping resampy as it is not installed.
collecting librosa
  Downloading librosa-0.10.1-py3-none-any.whl (253 kB)
    253.7/253.7 kB 4.8 MB/s eta 0:00:00
collecting resampy
  Downloading resampy-0.4.3-py3-none-any.whl (3.1 MB)
    3.1/3.1 MB 18.1 MB/s eta 0:00:00
Requirement already satisfied: audioread>=2.1.9 in /usr/local/lib/python3.10/dist-packages (from librosa) (3.0.1)
Requirement already satisfied: numpy<1.22.0,!=1.22.1,!=1.22.2,!=1.20.3 in /usr/local/lib/python3.10/dist-packages (from librosa) (1.25.2)
Requirement already satisfied: scipy>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from librosa) (1.11.4)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from librosa) (1.2.2)
Requirement already satisfied: joblib>=0.14 in /usr/local/lib/python3.10/dist-packages (from librosa) (1.3.2)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.10/dist-packages (from librosa) (4.4.2)
Requirement already satisfied: numba>=0.51.0 in /usr/local/lib/python3.10/dist-packages (from librosa) (0.58.1)
Requirement already satisfied: soundfile>=0.12.1 in /usr/local/lib/python3.10/dist-packages (from librosa) (0.12.1)
Requirement already satisfied: pooch>=1.0 in /usr/local/lib/python3.10/dist-packages (from librosa) (1.8.1)
Requirement already satisfied: soxr>=0.3.2 in /usr/local/lib/python3.10/dist-packages (from librosa) (0.3.7)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/dist-packages (from librosa) (4.10.0)
Requirement already satisfied: lazy-loader>=0.1 in /usr/local/lib/python3.10/dist-packages (from librosa) (0.3)
Requirement already satisfied: msgpack>=1.0 in /usr/local/lib/python3.10/dist-packages (from librosa) (1.0.8)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba>=0.51.0->librosa) (0.41.1)
Requirement already satisfied: platformdirs>=2.5.0 in /usr/local/lib/python3.10/dist-packages (from pooch>=1.0->librosa) (4.2.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from pooch>=1.0->librosa) (24.0)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/dist-packages (from pooch>=1.0->librosa) (2.31.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->librosa) (3.3.0)
Requirement already satisfied: cffi>=1.0 in /usr/local/lib/python3.10/dist-packages (from soundfile>=0.12.1->librosa) (1.16.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.0->soundfile>=0.12.1->librosa) (2.21)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->pooch>=1.0->librosa) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->pooch>=1.0->librosa) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->pooch>=1.0->librosa) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->pooch>=1.0->librosa) (2024.2.2)
Installing collected packages: resampy, librosa
Successfully installed librosa-0.10.1 resampy-0.4.3
```

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import librosa
import os
```

## Explanation

### Importing necessary libraries



```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import librosa
import os

[ ] pip install py7zr

Collecting py7zr
  Downloading py7zr-0.21.0-py3-none-any.whl (67 kB)
    67.6/67.6 kB 2.8 MB/s eta 0:00:00
Collecting texttable (from py7zr)
  Downloading texttable-1.7.0-py3-none-any.whl (10 kB)
Collecting pycryptodomex>=3.16.0 (from py7zr)
  Downloading pycryptodomex-3.20.0-cp35-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
    2.1/2.1 MB 14.2 MB/s eta 0:00:00
Collecting pyzstd>=0.15.9 (from py7zr)
  Downloading pyzstd-0.15.10-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (411 kB)
    411.2/411.2 kB 21.5 MB/s eta 0:00:00
Collecting pyppmd<1.2.0,>=1.1.0 (from py7zr)
  Downloading pyppmd-1.1.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (138 kB)
    138.9/138.9 kB 20.2 MB/s eta 0:00:00
Collecting pybcj<1.1.0,>=1.0.0 (from py7zr)
  Downloading pybcj-1.0.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (49 kB)
    49.7/49.7 kB 7.0 MB/s eta 0:00:00
Collecting multivolumefile>=0.2.3 (from py7zr)
  Downloading multivolumefile-0.2.3-py3-none-any.whl (17 kB)
Collecting inflate64<1.1.0,>=1.0.0 (from py7zr)
  Downloading inflate64-1.0.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (93 kB)
    93.1/93.1 kB 14.6 MB/s eta 0:00:00
Collecting brotli>=1.1.0 (from py7zr)
  Downloading Brotli-1.1.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (3.0 MB)
    3.0/3.0 MB 29.6 MB/s eta 0:00:00
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from py7zr) (5.9.5)
Installing collected packages: texttable, brotli, pyzstd, pyppmd, pycryptodomex, pybcj, multivolumefile, inflate64, py7zr
Successfully installed brotli-1.1.0 inflate64-1.0.0 multivolumefile-0.2.3 py7zr-0.21.0 pybcj-1.0.2 pycryptodomex-3.20.0 pyppmd-1.1.0 pyzstd-0.15.10 texttable-1.7.0
```

```
import py7zr
import os
# Path to your .7z file
file_path = '/content/samples.7z'
extracted_dir = '/content'
os.makedirs(extracted_dir, exist_ok=True)
with py7zr.SevenZipFile(file_path, mode='r') as z:
    z.extractall(path=extracted_dir)
print("Extraction completed.")
```

## Explanation

The code is used to extract files and data from a compressed .7z file.

- `extracted_dir`: This variable specifies the directory where the extracted contents will be saved. In your case, it's set to `/content`, which is a common directory in platforms like Google Colab.

- `os.makedirs(extracted_dir, exist_ok=True)`: This line of code creates the extraction directory (`/content` in your case) if it doesn't already exist (`exist_ok=True` ensures it doesn't throw an error if the directory already exists).
- `with py7zr.SevenZipFile(file_path, mode='r') as z::` This line opens the `.7z` file in read mode using `py7zr` and assigns it to the variable `z`.
- `z.extractall(path=extracted_dir)`: This command extracts all contents from the `.7z` file (`file_path`) and saves them in the specified extraction directory (`extracted_dir`).
- The `with` statement ensures that the file is closed properly after extraction.

### Completion Message:

- `print("Extraction completed.")`: This line prints a message to indicate that the extraction process has been successfully completed.

```
Extraction completed.
```

## Code

```
def load_data(test_size=0.2, chunk_duration=1):
    x, y = [], []
    for file in os.listdir('/content/samples'):
        audio, sample_rate = librosa.load('/content/samples/' + file,
res_type='kaiser_fast')
        num_chunks = int(np.ceil(len(audio) / (sample_rate *
chunk_duration)))
        for i in range(num_chunks):
            start = int(i * sample_rate * chunk_duration)
            end = min(len(audio), int((i + 1) * sample_rate *
chunk_duration))
            chunk_audio = audio[start:end]
            feature = extract_features_from_audio(chunk_audio,
sample_rate)
            x.append(feature)
            class_label = file.split('(')[0]
            y.append(class_label)
        encoder = LabelEncoder()
        y = encoder.fit_transform(y)
    return train_test_split(np.array(x), y, test_size=test_size,
random_state=42)
```

## Explanation

```
def load_data(test_size=0.2, chunk_duration=1):
    x, y = [], []
    for file in os.listdir('/content/samples'):
        audio, sample_rate = librosa.load('/content/samples/' + file,
res_type='kaiser_fast')
        num_chunks = int(np.ceil(len(audio) / (sample_rate *
chunk_duration)))
        for i in range(num_chunks):
            start = int(i * sample_rate * chunk_duration)
            end = min(len(audio), int((i + 1) * sample_rate *
chunk_duration))
            chunk_audio = audio[start:end]
            feature = extract_features_from_audio(chunk_audio,
sample_rate)
            x.append(feature)
            class_label = file.split('(')[0]
            y.append(class_label)
        encoder = LabelEncoder()
        y = encoder.fit_transform(y)
    return train_test_split(np.array(x), y, test_size=test_size,
random_state=42)
```

## Explanation:

In your project, the `load\_data` function is designed to prepare the audio data for training and testing your machine learning model for automatic machine fault detection and recognition using computer vision techniques. Here's an explanation of the function in easy English and its relevance to your project:

### 1. Purpose of the Function:

The `load\_data` function is crucial for loading and preprocessing audio data that contains information about machine faults such as Corona, Arcing, Looseness, and Tracking. This data is essential for training a machine learning model to detect and classify these faults based on audio characteristics.

### 2. Iterating Through Audio Files:

The function starts by iterating through the audio files located in the `/content/samples` directory. These audio files likely contain recordings or samples related to different machine faults.

### 3. Loading Audio Files:

Within the loop, each audio file is loaded using the `librosa.load` function, which reads the audio data and also provides information about the sample rate. The `res_type='kaiser_fast'` parameter indicates a specific method for resampling audio data.

#### 4. Chunking Audio Data:

After loading an audio file, the function divides it into smaller chunks based on the specified `chunk_duration`. This step is crucial for processing large audio files efficiently and extracting features from manageable segments.

#### 5. Feature Extraction:

For each chunk of audio data, features are extracted using the `extract_features_from_audio` function. These features capture important characteristics such as frequency, amplitude, and other audio properties that are relevant for fault detection.

#### 6. Creating Feature Vectors and Labels:

The extracted features are added to the feature vector `x`, while the corresponding class labels (e.g., Corona, Arcing, etc.) are added to the label vector `y`. This pairing of features and labels is fundamental for supervised machine learning tasks.

#### 7. Encoding Labels:

Before proceeding further, the class labels in `y` are encoded using `LabelEncoder` from the `sklearn.preprocessing` module. This step converts categorical labels into numerical values, which are easier for machine learning algorithms to process.

#### 8. Train-Test Split:

Finally, the function performs a train-test split on the feature matrix `x` and the encoded labels `y`. This split divides the data into training and testing sets, allowing you to evaluate the model's performance on unseen data.

The `load_data` function is preparing the audio data for machine learning tasks by extracting relevant features, encoding labels, and splitting the data for training and testing purposes.

### Code:

```
def extract_features_from_audio(audio, sample_rate, mfcc=True,
                                chroma=True, mel=True):
    result = np.array([])
    if mfcc:
        mfccs = np.mean(librosa.feature.mfcc(y=audio, sr=sample_rate,
                                                n_mfcc=40, n_fft=1024).T, axis=0) # Adjust n_fft
        result = np.hstack((result, mfccs))
    if chroma:
        stft = np.abs(librosa.stft(audio, n_fft=1024)) # Adjust n_fft
```

```

        chroma = np.mean(librosa.feature.chroma_stft(S=stft,
sr=sample_rate).T, axis=0)
        result = np.hstack((result, chroma))
    if mel:
        mel = np.mean(librosa.feature.melspectrogram(y=audio,
sr=sample_rate, n_fft=1024).T, axis=0) # Adjust n_fft
        result = np.hstack((result, mel))
    return result

```

## Explanation:

The `extract_features_from_audio` function is designed to extract important features from audio signals. These features capture essential characteristics of the audio, such as its frequency components, timbre, and spectral properties.

- The function takes parameters such as `mfcc`, `chroma`, and `mel`, which are flags indicating whether specific types of features should be extracted:
- `mfcc`: Mel-Frequency Cepstral Coefficients (MFCCs) are commonly used for speech and audio processing tasks. They represent the short-term power spectrum of sound.
- `chroma`: Chroma feature extraction focuses on the harmonic content of audio signals, capturing information about pitch and tonality.
- `mel`: Mel spectrogram features provide a representation of the audio signal in the mel-scale, which is more aligned with human perception of sound.

## Feature Extraction Process:

- The function initializes an empty array `result` to store the extracted features.
- For each specified feature type (e.g., MFCC, chroma, mel), the function calculates the corresponding features using the `librosa` library's feature extraction functions.
- The `librosa.feature.mfcc`, `librosa.feature.chroma_stft`, and `librosa.feature.melspectrogram` functions are used to compute the desired features from the audio signal.

## Adjusting Parameters:

- Parameters such as `n_mfcc`, `n_fft`, and `sr` (sample rate) are adjusted within the function to customize the feature extraction process. For example, `n_mfcc` controls the number of MFCC coefficients to compute, and `n_fft` sets the number of samples used in each FFT window.

## Combining Extracted Features:

- The extracted features are concatenated and stored in the `result` array using `np.hstack`. This step combines the different feature types into a single feature vector for each audio segment.



## Returning the Feature Vector:

- Finally, the function returns the concatenated feature vector `result`, which contains the extracted features based on the specified feature types.

## Code:

```
from tensorflow.keras import layers, models
from tensorflow.keras.layers import Dropout
def create_model_with_dropout(input_shape, num_classes, dropout_rate=0.5):
    model = models.Sequential()
    model.add(layers.Conv1D(32, 3, activation='relu',
input_shape=input_shape))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Conv1D(64, 3, activation='relu'))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Conv1D(128, 3, activation='relu'))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Conv1D(128, 3, activation='relu'))
    model.add(layers.MaxPooling1D(2))
    model.add(layers.Flatten())
    model.add(layers.Dense(512, activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(layers.Dense(num_classes, activation='softmax'))
    return model
X_train, X_test, y_train, y_test = load_data(test_size=0.2)
input_shape = (X_train.shape[1], 1)
num_classes = 4
model_with_dropout =
create_model_with_dropout(input_shape=(X_train.shape[1], 1),
num_classes=num_classes)
model_with_dropout.compile(optimizer='adam',
                           loss='sparse_categorical_crossentropy',
                           metrics=['accuracy'])
model_with_dropout.summary()
model_with_dropout.fit(X_train, y_train, epochs=50, batch_size=32,
validation_data=(X_test, y_test))

test_loss, test_acc = model_with_dropout.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```

## Output

```

dense (dense)                (None, 512)                508336
dropout (Dropout)            (None, 512)                0
dense_1 (Dense)              (None, 4)                  2052
-----
Total params: 672788 (2.57 MB)
Trainable params: 672788 (2.57 MB)
Non-trainable params: 0 (0.00 Byte)
-----
Epoch 1/50
16/16 [=====] - 7s 51ms/step - loss: 1.3290 - accuracy: 0.3780 - val_loss: 1.1453 - val_accuracy: 0.4127
Epoch 2/50
16/16 [=====] - 0s 7ms/step - loss: 1.0913 - accuracy: 0.5140 - val_loss: 1.0175 - val_accuracy: 0.5714
Epoch 3/50
16/16 [=====] - 0s 7ms/step - loss: 0.9540 - accuracy: 0.6160 - val_loss: 0.9033 - val_accuracy: 0.5635
Epoch 4/50
16/16 [=====] - 0s 7ms/step - loss: 0.7874 - accuracy: 0.6840 - val_loss: 0.8326 - val_accuracy: 0.5952
Epoch 5/50
16/16 [=====] - 0s 7ms/step - loss: 0.6994 - accuracy: 0.6760 - val_loss: 0.8002 - val_accuracy: 0.6746
Epoch 6/50
16/16 [=====] - 0s 7ms/step - loss: 0.6389 - accuracy: 0.7360 - val_loss: 0.6751 - val_accuracy: 0.7143
Epoch 7/50
16/16 [=====] - 0s 7ms/step - loss: 0.5637 - accuracy: 0.7800 - val_loss: 0.6442 - val_accuracy: 0.7460
Epoch 8/50
16/16 [=====] - 0s 7ms/step - loss: 0.5243 - accuracy: 0.7620 - val_loss: 0.6707 - val_accuracy: 0.6904
Epoch 9/50
16/16 [=====] - 0s 7ms/step - loss: 0.4947 - accuracy: 0.8020 - val_loss: 0.6675 - val_accuracy: 0.7460
Epoch 10/50
16/16 [=====] - 0s 7ms/step - loss: 0.4326 - accuracy: 0.8460 - val_loss: 0.5245 - val_accuracy: 0.7937
Epoch 11/50
16/16 [=====] - 0s 8ms/step - loss: 0.4075 - accuracy: 0.8320 - val_loss: 0.5555 - val_accuracy: 0.7143
Epoch 12/50
16/16 [=====] - 0s 6ms/step - loss: 0.3050 - accuracy: 0.8800 - val_loss: 0.5430 - val_accuracy: 0.8492
Epoch 13/50
16/16 [=====] - 0s 7ms/step - loss: 0.2708 - accuracy: 0.9000 - val_loss: 0.5448 - val_accuracy: 0.7540
Epoch 14/50
16/16 [=====] - 0s 7ms/step - loss: 0.2666 - accuracy: 0.8940 - val_loss: 0.5301 - val_accuracy: 0.7698
Epoch 15/50
16/16 [=====] - 0s 7ms/step - loss: 0.2889 - accuracy: 0.8800 - val_loss: 0.5668 - val_accuracy: 0.7857
Epoch 16/50
16/16 [=====] - 0s 7ms/step - loss: 0.2646 - accuracy: 0.9000 - val_loss: 0.5510 - val_accuracy: 0.7619
Epoch 17/50
16/16 [=====] - 0s 8ms/step - loss: 0.2398 - accuracy: 0.9100 - val_loss: 0.4996 - val_accuracy: 0.8095
Epoch 18/50
16/16 [=====] - 0s 7ms/step - loss: 0.1955 - accuracy: 0.9300 - val_loss: 0.5000 - val_accuracy: 0.8175
Epoch 19/50
16/16 [=====] - 0s 7ms/step - loss: 0.1447 - accuracy: 0.9560 - val_loss: 0.4726 - val_accuracy: 0.8254
Epoch 20/50
16/16 [=====] - 0s 6ms/step - loss: 0.1435 - accuracy: 0.9500 - val_loss: 0.5663 - val_accuracy: 0.8333
Epoch 21/50
16/16 [=====] - 0s 7ms/step - loss: 0.1595 - accuracy: 0.9260 - val_loss: 0.4614 - val_accuracy: 0.8492
Epoch 22/50
16/16 [=====] - 0s 7ms/step - loss: 0.1620 - accuracy: 0.9500 - val_loss: 0.6167 - val_accuracy: 0.8175
Epoch 23/50
16/16 [=====] - 0s 7ms/step - loss: 0.1405 - accuracy: 0.9380 - val_loss: 0.3703 - val_accuracy: 0.8651
Epoch 24/50
```

```
Epoch 48/50
16/16 [=====] - 0s 11ms/step - loss: 0.0397 - accuracy: 0.9840 - val_loss: 0.7391 - val_accuracy: 0.8333
Epoch 49/50
16/16 [=====] - 0s 10ms/step - loss: 0.0273 - accuracy: 0.9940 - val_loss: 0.5656 - val_accuracy: 0.8492
Epoch 50/50
16/16 [=====] - 0s 10ms/step - loss: 0.0425 - accuracy: 0.9860 - val_loss: 0.9562 - val_accuracy: 0.8016
4/4 [=====] - 0s 4ms/step - loss: 0.9562 - accuracy: 0.8016
Test accuracy: 0.8015872836112976
```

## Explanation:-

The code snippet you provided is related to creating, compiling, training, and evaluating a convolutional neural network (CNN) model with dropout regularization for your project. Here's an explanation of the code in easy English:

### 1. Importing Libraries:

- The code starts by importing necessary libraries from TensorFlow's Keras API, which is used for building deep learning models.
- `layers` and `models` from `tensorflow.keras` are imported to define layers and models for the neural network.
- `Dropout` is imported separately from `tensorflow.keras.layers` for applying dropout regularization.

### 2. Defining the Model Architecture:

- The `create\_model\_with\_dropout` function is defined to create a CNN model with dropout regularization.
- The model architecture includes several layers:
- Convolutional layers with varying filter sizes and activation functions (ReLU).

- MaxPooling layers for downsampling the feature maps.
- Flatten layer to convert the 2D feature maps into a 1D feature vector.
- Dense layers with ReLU activation for classification, and a final softmax layer for output probabilities.

### 3. Model Compilation:

After creating the model, it is compiled using the Adam optimizer, sparse categorical cross-entropy loss function, and accuracy as the metric for evaluation.

### 4. Model Training:

- The model is trained using the `fit` method with training data (`X\_train` and `y\_train`), specifying the number of epochs, batch size, and validation data (`X\_test` and `y\_test`).

### 5. Model Evaluation:

- After training, the model's performance is evaluated using the `evaluate` method on the test data (`X\_test` and `y\_test`), and the test accuracy is printed.

### Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Predict the probabilities for each class for the test data
y_pred_probabilities = model_with_dropout.predict(X_test)

# Get the predicted class labels
y_pred = np.argmax(y_pred_probabilities, axis=1)

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=np.unique(y_test))
disp.plot()
plt.show()
```

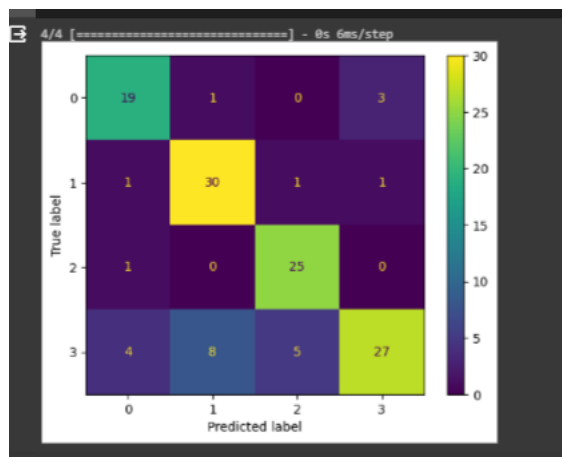
### Explanation:

The code snippet provided is used to generate and display a confusion matrix to evaluate the performance of a our model.

The code snippet provided is used to generate and display a confusion matrix to evaluate the performance of a machine learning model, specifically in your project related to automatic

machine fault detection and recognition using computer vision techniques. Here's an explanation of the code:

- The `model_with_dropout` is used to predict the probabilities for each class using the `predict` method on the test data (`X_test`). These probabilities are stored in `y_pred_probabilities`.
- The predicted class labels are obtained by finding the index of the maximum probability in each row of `y_pred_probabilities`, using `np.argmax` with `axis=1`. This gives the predicted class labels in `y_pred`.
- The confusion matrix is computed using the `confusion_matrix` function from `sklearn.metrics`. It takes the true labels (`y_test`) and predicted labels (`y_pred`) as inputs to calculate the number of true positives, true negatives, false positives, and false negatives for each class.
- The `plot` method of `ConfusionMatrixDisplay` is then called to visualize the confusion matrix using matplotlib.
- Finally, `plt.show()` is used to display the confusion matrix plot.



### Code:

```
import matplotlib.pyplot as plt

history = model_with_dropout.fit(X_train, y_train, epochs=50,
batch_size=32, validation_data=(X_test, y_test))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

### Explanation:

The code snippet provided is used to plot the training and validation accuracy over epochs during the training of a machine learning model. Here's an explanation of the code:

### Importing Libraries:

- `matplotlib.pyplot` (`plt`) is imported for data visualization.

### Training the Model:

- The `model_with_dropout` is trained using the `fit` method, specifying the training data (`X_train` and `y_train`), number of epochs (50), batch size (32), and validation data (`X_test` and `y_test`).
- The training history, which includes metrics like accuracy, loss, etc., is stored in the `history` object. This object contains information about the model's performance during training.

### Plotting Training and Validation Accuracy:

- `plt.plot(history.history['accuracy'], label='Training Accuracy')` plots the training accuracy over epochs using data from the `history` object.
- `plt.plot(history.history['val_accuracy'], label='Validation Accuracy')` plots the validation accuracy over epochs using data from the `history` object.
- `plt.xlabel('Epochs')` sets the label for the x-axis as "Epochs".
- `plt.ylabel('Accuracy')` sets the label for the y-axis as "Accuracy".
- `plt.legend()` adds a legend to the plot to distinguish between training and validation accuracy curves.
- `plt.show()` displays the plot of training and validation accuracy.

### Output



## Summary

Automatic machine fault detection and recognition play a crucial role in industrial settings to ensure smooth operation, prevent downtime, and enhance overall productivity. This project focuses on leveraging computer vision techniques to detect and classify machine faults, specifically targeting faults like Corona, Arcing, Looseness, and Tracking. The project employs deep learning models, data preprocessing, feature extraction, and performance evaluation metrics to achieve accurate fault detection and recognition.

The significance of automatic machine fault detection cannot be overstated in modern industries. Timely detection of faults such as Corona (electrical discharge), Arcing (electric sparks), Looseness (mechanical instability), and Tracking (insulation breakdown) is essential for preventing equipment damage, minimizing downtime, ensuring worker safety, and optimizing maintenance schedules. Traditional manual inspection methods are often time-consuming, prone to errors, and inefficient for detecting subtle faults. Automated fault detection using computer vision offers a reliable, efficient, and cost-effective solution to address these challenges.

The End