The University of Azad Jammu and Kashmir, Muzaffarabad

Department of Software Engineering

Course Title: Computer Vision

Course Code: (SE-4104)

Course Instructor: Engr. Ahmed Khawaja

Semester: 7h Semester

Session: 2021 – 25

Project Report for

COMPUTER VISION-BASED AUDIO SIGNAL ANALYSIS FOR FAULT DETECTION

BY

Muneeba Shafiq	**2021-SE-28**
Anish Waseem	**2021-SE-29**
Zubia Israr	**2021-SE-38**

Table of Contents

Abstract

Detecting faults in industrial machines is essential to prevent unexpected breakdowns and costly repairs. Traditional fault detection methods rely on vibration analysis or manual inspection, which require specialized equipment and expertise. This project aims to develop an automated system that analyzes sound signals using deep learning techniques. The sound data is preprocessed, converted into images, and then classified using a convolutional neural network (CNN). By applying Short-Time Fourier Transform (STFT) and Mel-Frequency Cepstral Coefficients (MFCCs), the system transforms audio signals into meaningful visual representations. The CNN model is trained on these images to detect different types of machine faults. The final model achieves high accuracy, making it a reliable and efficient solution for automated fault detection in industrial applications. Future improvements include refining feature extraction techniques and deploying the model for real-time monitoring.

1. Introduction

In industrial settings, detecting faults early is crucial to maintaining efficiency and reducing downtime. Machines often produce distinct sound patterns when faults develop. Analyzing these sound signals can help identify faults before they lead to failures. Traditional fault detection methods, such as vibration analysis, require expensive sensors and expert analysis. This project proposes an automated solution using computer vision and deep learning to analyze sound signals. The approach involves converting audio data into images and using a convolutional neural network (CNN) to classify different fault types. The goal is to develop a system that is accurate, fast, and reliable for real-world industrial applications.

2. Methodology

This project follows a structured approach, which includes data preprocessing, feature extraction, data augmentation, model training, and evaluation.

- **Data Preprocessing**

Raw audio signals are collected from machine operations and processed to ensure consistency. Since analyzing long audio files is complex, they are split into smaller chunks of 0.2 seconds each. This segmentation ensures that each sample captures relevant sound features. A 20% overlap is applied between consecutive chunks to preserve smooth transitions in the audio data. The sampling rate is set to 44.1 kHz to maintain high-quality signal processing.

- **Feature Extraction**

Once the audio data is segmented, meaningful features are extracted to convert the sound into images. This process includes:

a) **Short-Time Fourier Transform (STFT) Magnitude Images**: The STFT magnitude images show how sound frequencies change over time. STFT divides an audio signal into small sections and analyzes how much of each frequency is present in these sections. This helps visualize the variations in sound intensity at different times, making it easier to detect abnormal patterns in a machine's operation. When a fault occurs, the frequency patterns may shift, become more intense, or show irregularities. These changes in the magnitude spectrum help identify possible faults.

b) **STFT Phase Images**: While magnitude images show how much of each frequency is present, STFT phase images capture the timing and direction of sound waves. Phase information tells us how different sound waves interact with each other over time. In a healthy machine, the phase patterns remain stable, but if a fault develops, these patterns can shift unpredictably. Capturing these variations helps in detecting machine faults that may not be evident in the magnitude images alone. Phase images provide additional insights that improve fault classification accuracy.

c) **Mel-Frequency Cepstral Coefficients (MFCCs)**: The Mel-Frequency Cepstral Coefficients (MFCCs) are a widely used feature in sound processing because they closely mimic how human ears interpret sounds. MFCCs extract the most important characteristics of a sound, filtering out unnecessary background noise. They focus on the parts of the sound that change over time, which makes them useful for identifying different fault patterns in a machine. By converting MFCCs into images, the model can analyze how these features evolve, making it easier to classify machine faults accurately.

Each extracted feature is converted into an image, ensuring that the CNN can effectively learn from the sound data.

- **Data Augmentation**

To improve the model's ability to recognize faults in different conditions, several data augmentation techniques are applied. These techniques include:

a) **Time Stretching**: Time stretching is a technique where the speed of an audio signal is slightly increased or decreased without changing its pitch. This helps the model learn how the same machine fault might sound at different operational speeds. By training on both faster and slower versions of the same audio, the model becomes better at generalizing across real-world machine conditions.

b) **Pitch Shifting**: Pitch shifting involves altering the pitch of an audio signal while keeping its speed constant. This simulates slight differences in machine noise that may occur due to changes in load, temperature, or machine wear. By training the model on pitch-shifted audio samples, it learns to recognize faults even when external conditions cause slight variations in the sound.

c) **Noise Addition**: In real industrial environments, background noise is common. Noise addition involves introducing small amounts of random noise into the audio samples to make the model more resilient. This ensures that the model can correctly classify faults even when there is interference from other machines, environmental sounds, or recording imperfections.

Applying these transformations increases the dataset's diversity, helping the CNN generalize better to new data.

3. Model Training

A Convolutional Neural Network (CNN) is used to classify the generated spectrogram images. The dataset is divided into 80% training data and 20% testing data to ensure proper model evaluation. The CNN consists of multiple layers:

- **Convolutional Layers:** These layers extract important features from the images by applying filters that detect edges, textures, and patterns. By scanning different parts of the image, the network learns to identify machine faults based on sound characteristics. The deeper the network, the more complex patterns it can recognize, improving classification accuracy.

- **Max Pooling Layers:** Max pooling layers help in reducing the image size while keeping important information intact. They work by selecting the most significant values from small regions of the image, making the model more efficient. This also helps in reducing computational requirements, allowing faster processing without losing critical details.

- **Fully Connected Layers:** After extracting key features, the fully connected layers process this information to make final predictions. These layers combine patterns learned from previous layers and assign probabilities to different fault categories. This step ensures that the model accurately classifies the given sound data.

- **Dropout Layers:** To prevent overfitting, dropout layers randomly disable certain neurons during training. This forces the model to learn more general patterns rather than memorizing the training data. As a result, the model performs better when predicting faults on new, unseen data.
- **Softmax Output Layer:** The final layer of the model is the softmax output layer, which classifies the image into one of the four fault categories. It assigns a probability to each class and selects the one with the highest confidence. This ensures that the model provides accurate and meaningful predictions for fault detection.

4. Evaluation Metrics

After training, the model's performance is evaluated using different metrics:

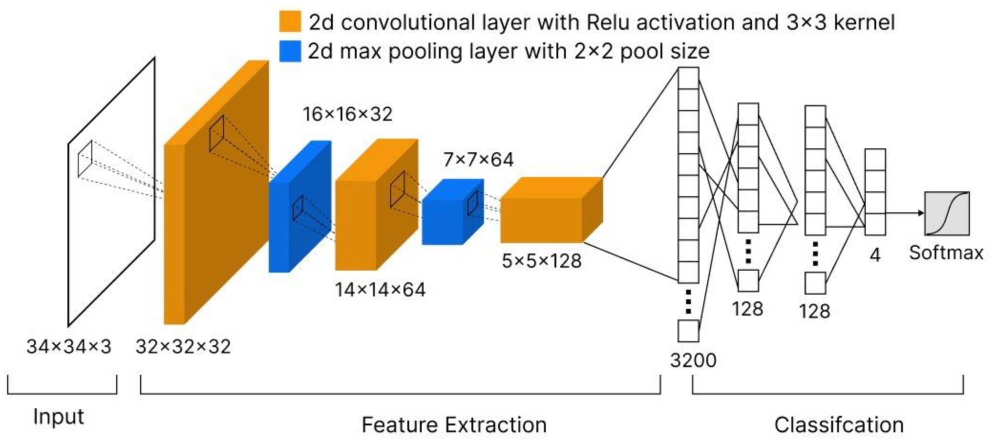
- **Accuracy**: Accuracy measures how many faults the model correctly identifies out of all predictions. A high accuracy score indicates a well-trained and effective model.
- **Loss Graphs**: Loss graphs show how well the model improves over each training epoch. A decreasing loss value means the model is learning effectively.
- **Confusion Matrix**: The confusion matrix provides a breakdown of correct and incorrect classifications. It helps identify which fault types the model struggles with.
- **Comparison with Other Models**: The CNN's performance is compared with other deep learning models to determine the best approach. Alternative architectures may provide better accuracy for fault detection.

The final model achieves an accuracy of **87%**, demonstrating its effectiveness in detecting machine faults based on sound signals.

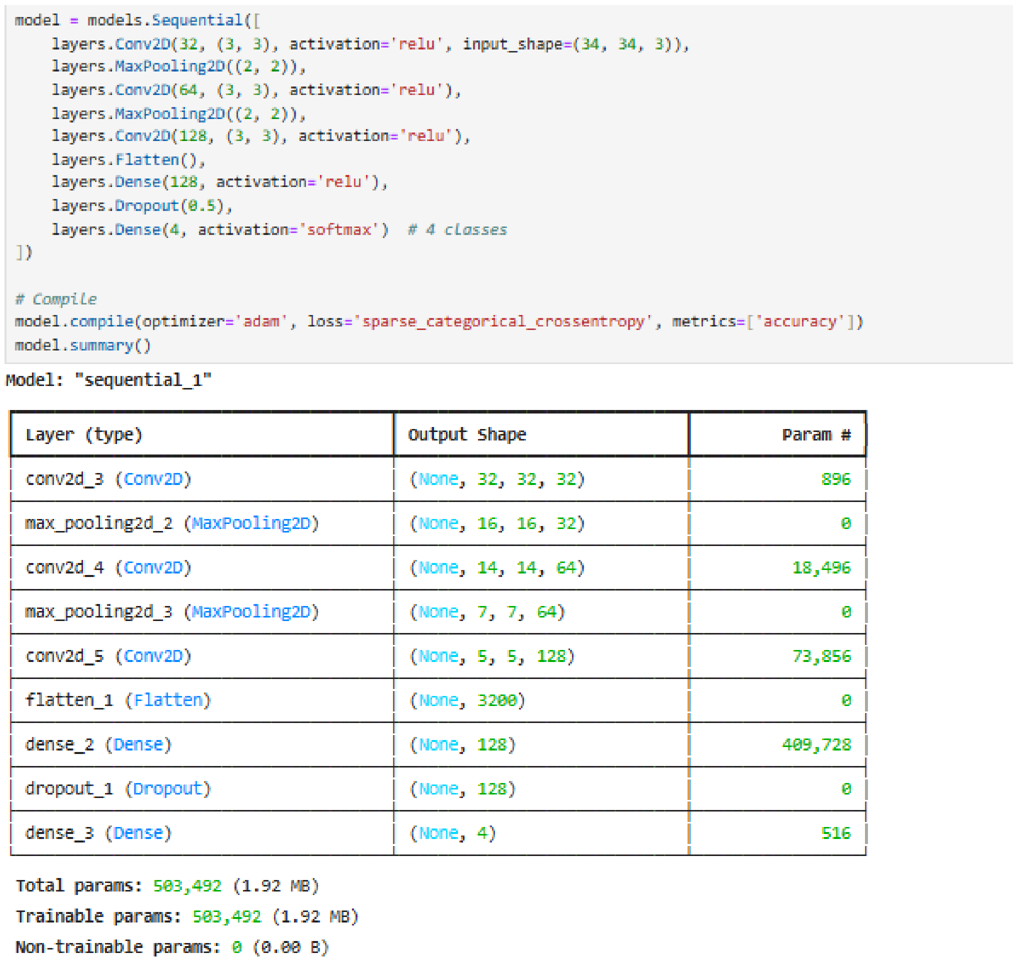
5. Results

This project successfully develops an automated fault detection system using sound analysis and deep learning. The system transforms audio signals into images and classifies faults using a CNN.

- **Model Architecture Overview**



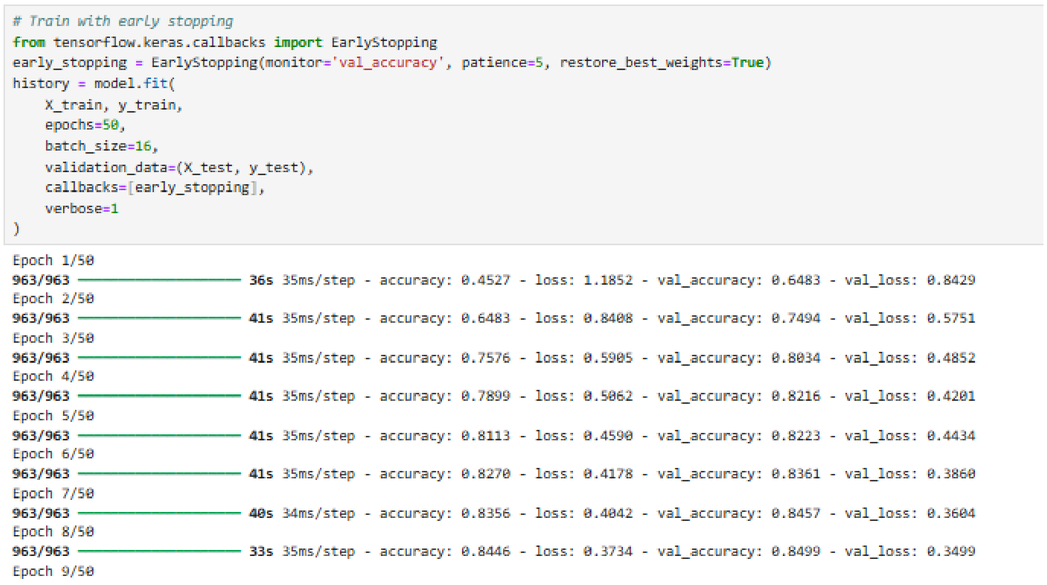
🟧 2d convolutional layer with Relu activation and 3×3 kernel
🟦 2d max pooling layer with 2×2 pool size
34×34×3
32×32×32
16×16×32
14×14×64
7×7×64
5×5×128
3200
128
128
4
Softmax
Input
Feature Extraction
Classifcation



```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(34, 34, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(4, activation='softmax')  # 4 classes
])

# Compile
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_1"

┃ Layer (type)                    ┃ Output Shape          ┃       Param # ┃
│ conv2d_3 (Conv2D)               │ (None, 32, 32, 32)    │           896 │
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 16, 16, 32)    │             0 │
│ conv2d_4 (Conv2D)               │ (None, 14, 14, 64)    │        18,496 │
│ max_pooling2d_3 (MaxPooling2D)  │ (None, 7, 7, 64)      │             0 │
│ conv2d_5 (Conv2D)               │ (None, 5, 5, 128)     │        73,856 │
│ flatten_1 (Flatten)             │ (None, 3200)          │             0 │
│ dense_2 (Dense)                 │ (None, 128)           │       409,728 │
│ dropout_1 (Dropout)             │ (None, 128)           │             0 │
│ dense_3 (Dense)                 │ (None, 4)             │           516 │

 Total params: 503,492 (1.92 MB)
 Trainable params: 503,492 (1.92 MB)
 Non-trainable params: 0 (0.00 B)
```

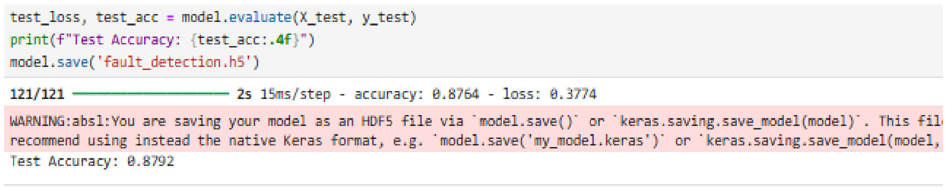
- **Model Training Progress**



```
# Train with early stopping
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights=True)
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=16,
    validation_data=(X_test, y_test),
    callbacks=[early_stopping],
    verbose=1
)
```

```
Epoch 1/50
963/963 ━━━━━━━━━━━━━━━━━━━━ 36s 35ms/step - accuracy: 0.4527 - loss: 1.1852 - val_accuracy: 0.6483 - val_loss: 0.8429
Epoch 2/50
963/963 ━━━━━━━━━━━━━━━━━━━━ 41s 35ms/step - accuracy: 0.6483 - loss: 0.8408 - val_accuracy: 0.7494 - val_loss: 0.5751
Epoch 3/50
963/963 ━━━━━━━━━━━━━━━━━━━━ 41s 35ms/step - accuracy: 0.7576 - loss: 0.5905 - val_accuracy: 0.8034 - val_loss: 0.4852
Epoch 4/50
963/963 ━━━━━━━━━━━━━━━━━━━━ 41s 35ms/step - accuracy: 0.7899 - loss: 0.5062 - val_accuracy: 0.8216 - val_loss: 0.4201
Epoch 5/50
963/963 ━━━━━━━━━━━━━━━━━━━━ 41s 35ms/step - accuracy: 0.8113 - loss: 0.4590 - val_accuracy: 0.8223 - val_loss: 0.4434
Epoch 6/50
963/963 ━━━━━━━━━━━━━━━━━━━━ 41s 35ms/step - accuracy: 0.8270 - loss: 0.4178 - val_accuracy: 0.8361 - val_loss: 0.3860
Epoch 7/50
963/963 ━━━━━━━━━━━━━━━━━━━━ 40s 34ms/step - accuracy: 0.8356 - loss: 0.4042 - val_accuracy: 0.8457 - val_loss: 0.3604
Epoch 8/50
963/963 ━━━━━━━━━━━━━━━━━━━━ 33s 35ms/step - accuracy: 0.8446 - loss: 0.3734 - val_accuracy: 0.8499 - val_loss: 0.3499
Epoch 9/50
```

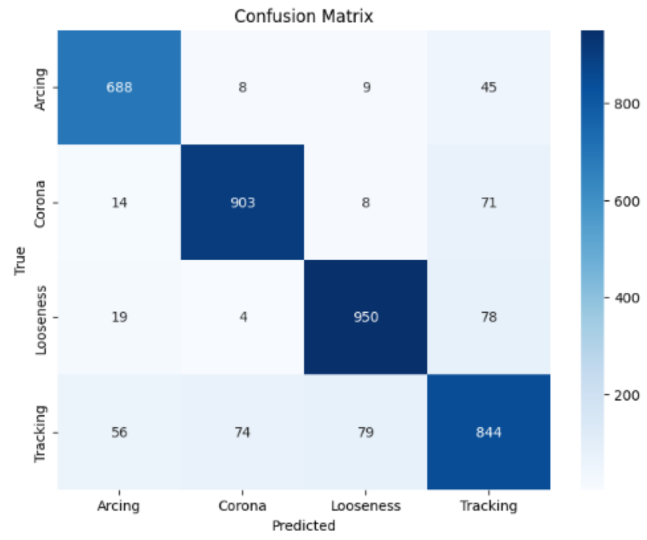
- **Model Evaluation Results**



```
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc:.4f}")
model.save('fault_detection.h5')
```

```
121/121 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.8764 - loss: 0.3774
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This fil
recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model,
Test Accuracy: 0.8792
```

- **Confusion Matrix for Model Performance**

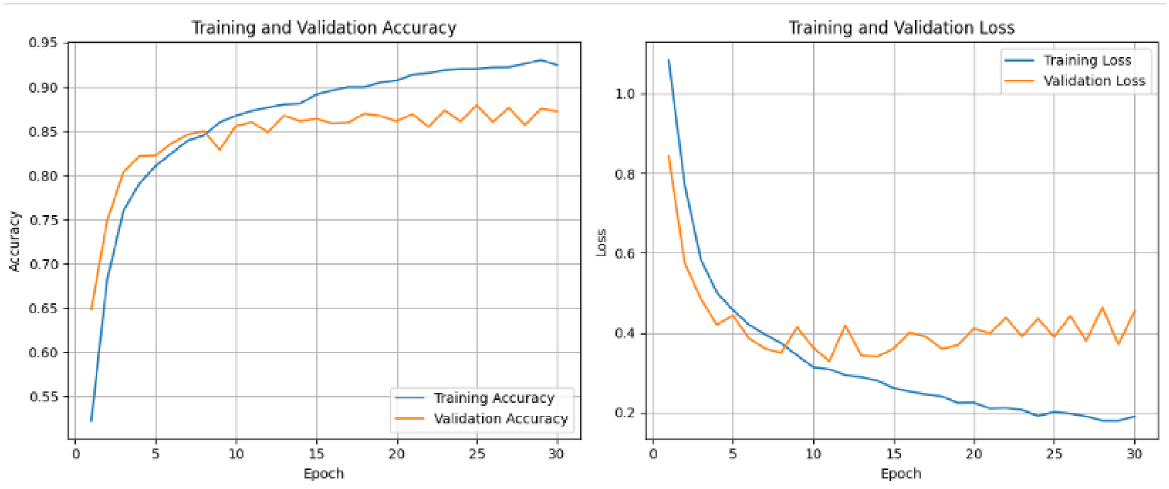


- ## Classification Report

```
# Compute and display other evaluation metrics
print("\nClassification Report:")
print(classification_report(y_test, y_pred_classes, target_names=fault_types_reverse.values()))

# Compute overall metrics
val_loss, val_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"\nValidation Loss: {val_loss:.4f}")
print(f"Validation Accuracy: {val_accuracy:.4f}")
```

```
Classification Report:
              precision    recall  f1-score   support

      Arcing       0.89      0.92      0.90       750
      Corona       0.91      0.91      0.91       996
   Looseness       0.91      0.90      0.91      1051
    Tracking       0.81      0.80      0.81      1053

    accuracy                           0.88      3850
   macro avg       0.88      0.88      0.88      3850
weighted avg       0.88      0.88      0.88      3850


Validation Loss: 0.3894
Validation Accuracy: 0.8792
```

- ## Loss and Accuracy Curves



6. Future Improvements

There are opportunities for further improvements:

- **Fine-Tuning Hyper-parameters:** Adjusting learning rates, batch sizes, and network layers can optimize performance. This ensures better training efficiency and improves fault detection accuracy.
- **Enhancing Feature Extraction Methods:** Exploring advanced sound processing techniques can help capture more meaningful features. This may improve classification accuracy and make the model more robust.
- **Deploying in Real-Time Applications:** Integrating the model into an industrial system for continuous monitoring is essential. This will allow real-time fault detection, preventing potential failures before they occur.

By implementing these improvements, the system can become even more robust and practical for industrial use.

7. Conclusion

This project presents a computer vision-based approach for detecting machine faults using sound signals. By converting audio data into spectrogram images and using a CNN model, an efficient and automated fault detection system is developed. The model achieves high accuracy, demonstrating its effectiveness in identifying different fault types. Future work will focus on refining the model and deploying it in real-world industrial applications, making fault detection faster, more reliable, and accessible to industries worldwide.