# Project Report

## 1. Road Condition Classification Using CNN

**Introduction**:

The rapid increase in road usage has resulted in significant wear and tear, necessitating timely and effective maintenance strategies. This project focuses on classifying road conditions into four categories: Good, Poor, Satisfactory, and Very Poor, leveraging a Convolutional Neural Network (CNN)-based deep learning model for accurate assessment.

**Objective**:

- The primary goal of this project is to develop a CNN-based classification model that can accurately categorize road damage from images.
- This classification helps authorities prioritize road maintenance and ensure road safety.

**Dataset:**

- It contains images categorized into four classes: Good, Poor, Satisfactory, and Very Poor
- Training and testing sets were stored in separate directories: Training and Testing

Dataset Source: https://www.kaggle.com/datasets/prudhvignv/road-damage-classification-and-assessment

**Implementation Steps:**

**Step 1: Data Preprocessing and Augmentation**:

To enhance the robustness of the model, data augmentation techniques were applied using TensorFlow's ImageDataGenerator. Augmentations included:

- Rescaling images to a range of [0,1].
- Random shear transformation.
- Zoom transformations.
- Horizontal flipping.
- Resizing all images to **64x64 pixels**.

**Step 2: Model Architecture** The CNN model was built using TensorFlow and Keras with the following architecture:

- **Input Layer:** Image of size (64, 64, 3)
- **Convolutional Layers:**
    - 32 filters, kernel size (3,3), ReLU activation, followed by MaxPooling (2,2)
    - 64 filters, kernel size (3,3), ReLU activation, followed by MaxPooling (2,2)
    - 128 filters, kernel size (3,3), ReLU activation, followed by MaxPooling (2,2)
- **Fully Connected Layers:**

- Dense layer with 64 neurons (ReLU activation)
- Dense layer with 32 neurons (ReLU activation)
- Output layer with 4 neurons (Softmax activation)

**Step 3: Model Compilation and Training**

- The model was compiled using:
    - **Optimizer:** Adam
    - **Loss Function:** Sparse Categorical Crossentropy
    - **Metric:** Accuracy
- **Training Parameters:**
    - Batch size: 32
    - Number of epochs: 30
    - Validation on the test dataset

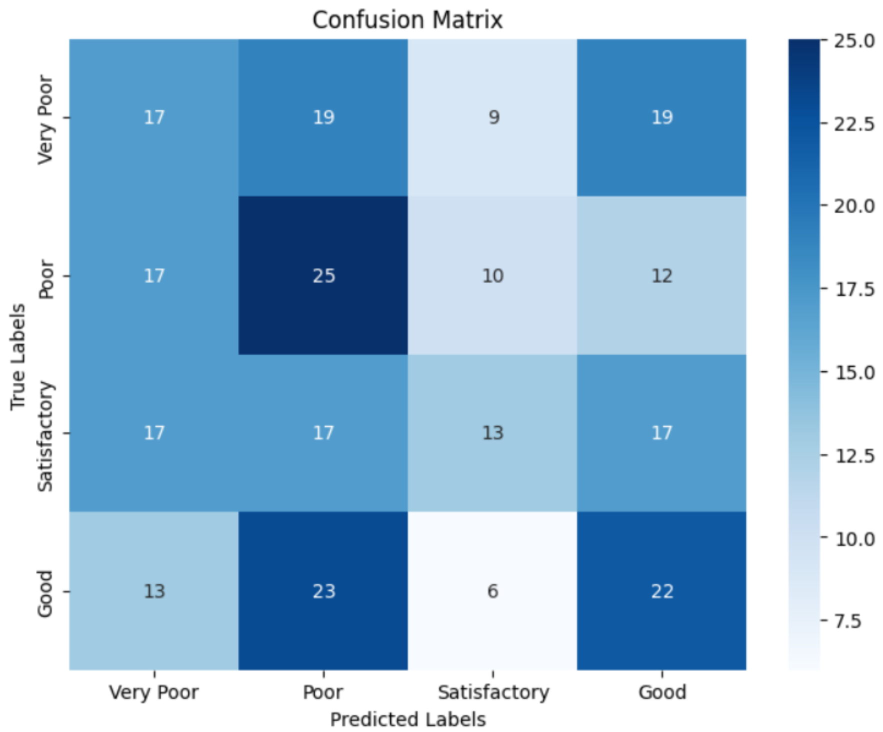
**Results:**

```
model = load_model(r"D:\svasti vector solution\CNN_RoadDamageClassification\road_damage_model.h5")

img_path = r"D:\svasti vector solution\CNN_RoadDamageClassification\sih_road_dataset\test_img_VP.jpg"

test_image = image.load_img(img_path, target_size=(64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)
test_image = test_image / 255.0


prediction = model.predict(test_image)

predicted_class = np.argmax(prediction, axis=1)

class_labels = {0: 'Good', 1: 'Poor', 2: 'Satisfactory', 3: 'Very Poor'}

predicted_label = class_labels[predicted_class[0]]

print(f"The predicted class is: {predicted_label}")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step
The predicted class is: Very Poor
```

```
model = load_model(r"D:\svasti vector solution\CNN_RoadDamageClassification\road_damage_model.h5")

img_path = r"D:\svasti vector solution\CNN_RoadDamageClassification\sih_road_dataset\test_img_Go.jpg"

test_image = image.load_img(img_path, target_size=(64, 64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)
test_image = test_image / 255.0

prediction = model.predict(test_image)

predicted_class = np.argmax(prediction, axis=1)

class_labels = {0: 'Good', 1: 'Poor', 2: 'Satisfactory', 3: 'Very Poor'}

predicted_label = class_labels[predicted_class[0]]

print(f"The predicted class is: {predicted_label}")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 136ms/step
The predicted class is: Good
```




**Confusion Matrix Analysis:**

- A heatmap was generated to visualize classification performance.
- Some misclassifications were observed, indicating potential improvement areas.

**Model Deployment and Testing:**

- The trained model was saved as road_damage_model.h5.
- The model was tested on a sample image using the following steps:

  - Image was preprocessed and converted into an array.
  - Model prediction was obtained using predict().
  - The predicted class label was mapped using a dictionary.
  - The result was displayed with Matplotlib.

**Code:**
https://drive.google.com/file/d/1jyVxMHCFwm2GDjPIGnLOSVXXi1bNqWwV/view?usp=drive_link

**Conclusion and Future Work**:

This project successfully implemented a CNN-based road damage classification system. The model demonstrated promising accuracy and can be further enhanced by:

- Using a larger dataset to improve generalization.
- Fine-tuning hyperparameters to optimize performance.
- Implementing transfer learning techniques for better feature extraction.
- Deploying the model in real-time applications for automated road condition monitoring.

## 2.Road Condition Classification using CNN and MobileNetV2

**Introduction:**

Road damage and poor road conditions pose significant challenges for transportation safety and maintenance planning. This project aims to classify road conditions into four categories: Good, Poor, Satisfactory, and Very Poor using deep learning techniques. By leveraging MobileNetV2 with transfer learning, we aim to build an efficient and accurate model for road condition classification.

**Objectives:**

- To develop an image classification model that categorizes road conditions into four classes.
- To utilize transfer learning with MobileNetV2 for improved accuracy and computational efficiency.
- To apply image augmentation techniques to enhance model generalization.
- To evaluate model performance and improve accuracy for real-world deployment.

**Step 1: Data Collection**

The dataset consists of images categorized into four classes:

- **Good**: Roads in excellent condition with minimal or no visible damage.
- **Poor**: Roads with significant cracks or wear but still usable.
- **Satisfactory**: Roads with minor cracks or irregularities.
- **Very Poor**: Roads with severe damage, potholes, or hazardous conditions.

Dataset Source: https://www.kaggle.com/datasets/prudhvignv/road-damage-classification-and-assessment

**Step 2: Data Preprocessing**

- Images were resized to **224x224** pixels to match MobileNetV2's input dimensions.
- Data augmentation was applied to enhance model robustness:
  - **Rotation (15 degrees)**
  - **Width and height shift (10%)**
  - **Shear transformation (10%)**
  - **Zoom range (10%)**
  - **Horizontal flipping**
- The dataset was split into **training** and **testing** sets.

**Step 3: Model Architecture**

**Pre-trained Model: MobileNetV2**

- MobileNetV2 was selected due to its efficiency in handling image classification tasks with minimal computational overhead.
- The model was used without its top layers to allow for custom classification layers.
- The convolutional layers were **frozen** to retain pre-trained features.

**Custom Layers**

- **Global Average Pooling** to reduce feature dimensions.
- **Fully Connected Layer (128 neurons, ReLU activation)** for learning complex patterns.
- **Dropout Layer (50%)** to prevent overfitting.
- **Output Layer (4 neurons, Softmax activation)** to classify images into four categories.

**Step 4: Model Training**

**Training Configuration**

- **Optimizer**: Adam with a learning rate of **0.001**
- **Loss Function**: Categorical Crossentropy
- **Batch Size**: 16
- **Number of Epochs**: 10

**Training Process**

- ImageDataGenerator was used for real-time image augmentation.
- Training images were fed in batches to improve computational efficiency.
- The model was validated using the test dataset after each epoch.

**Step 5: Model Evaluation**

**Performance Metrics**

The model was evaluated using:

- **Categorical Accuracy**: Measures the percentage of correctly classified images.
- **Loss Function**: Evaluates how well the model is minimizing errors.

**Step 6: Results**

```python
img_path = 'test_img_Go.jpg'
img = image.load_img(img_path, target_size=(224, 224))  # Use the same target size as used in training
img_array = image.img_to_array(img)  # Convert the image to a NumPy array
img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
img_array = img_array / 255.0  # Normalize if required (based on your training preprocessing)

# Use the loaded model to predict the class of the new image
prediction = model.predict(img_array)

# If you have multiple classes, you can find the class with the highest probability
predicted_class = np.argmax(prediction, axis=1)

if predicted_class[0] == 0:
    m = "Good"
elif predicted_class[0] == 1:
    m = "Poor"
elif predicted_class[0] == 2:
    m = "Moderate"
elif predicted_class[0] == 3:
    m = "Very Poor"
# Print the predicted class
print("Predicted Class:", predicted_class," which is",m)
```

```
1/1 [==============================] - 0s 59ms/step
Predicted Class: [0]  which is Good
```

```python
img_path = 'test_img_P.jpg'
img = image.load_img(img_path, target_size=(224, 224))  # Use the same target size as used in training
img_array = image.img_to_array(img)  # Convert the image to a NumPy array
img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
img_array = img_array / 255.0  # Normalize if required (based on your training preprocessing)

# Use the loaded model to predict the class of the new image
prediction = model.predict(img_array)

# If you have multiple classes, you can find the class with the highest probability
predicted_class = np.argmax(prediction, axis=1)

if predicted_class[0] == 0:
    m = "Good"
elif predicted_class[0] == 1:
    m = "Poor"
elif predicted_class[0] == 2:
    m = "Moderate"
elif predicted_class[0] == 3:
    m = "Very Poor"
# Print the predicted class
print("Predicted Class:", predicted_class," which is",m)
```

```
1/1 [==============================] - 0s 81ms/step
Predicted Class: [3]  which is Very Poor
```

**Step 7: Model Deployment**

- The trained model was saved in HDF5 format (road_condition_model2.h5).
- It can be deployed for real-world applications such as automated road condition monitoring systems.

**Code:**

https://drive.google.com/file/d/1QCOxkLtac6hE4-oQYTvhl5Rg1j81nyoS/view?usp=drive_link

**Load Model:**
https://drive.google.com/file/d/1dxRBiOY_uyltQZ3dBQvYXYvGjUc5cSSg/view?usp=drive_link

**H5 File:**

https://drive.google.com/file/d/1R6UF_QEaeUrb2ML64k7za99C-If54Mt8/view?usp=drive_link

**Future Enhancements:**

- **Fine-tuning MobileNetV2 layers** to improve feature extraction.
- **Increasing dataset size** with real-world road images to improve generalization.
- **Experimenting with alternative architectures** like ResNet50 or EfficientNet.
- **Deploying the model as a web application** for real-time road condition analysis.

**Conclusion:**

This project successfully developed a road condition classification model using MobileNetV2 with transfer learning. The model demonstrated promising results, making it a viable tool for automated road condition monitoring. Future work will focus on enhancing accuracy and deploying the model in practical applications.