National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Keerthana Reddy Mettu
Student ID: x23275634

School of Computing
National College of Ireland

Supervisor: Jaswinder Singh

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Keerthana Reddy Mettu

Student ID: x23275634

Programme: MSc in Data Analytics **Year:** 2024 - 2025

Module: MSc (Research) Practicum/Internship Part 2

Lecturer: Jaswinder Singh

Submission Due Date: 15-09-2025

Project Title: Car Condition Detection Using CNN Algorithm

Word Count: 1267 **Page Count:** 7

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Keerthana Reddy Mettu

Date: 15-09-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Keerthana Reddy Mettu
Student ID: x23275634

1 Section 1: Used Software

The model developed used the PyTorch deep learning library to code and train the model, and evaluate it under Python 3.10. Other libraries, including Torchvision, NumPy, and Matplotlib, aided in preprocessing the images, loading data, and visualizing the images. Jupyter Notebook used as the development platform provided the framework of an iterative test environment. One of the transfer learning models includes VGG16 and ResNet18; the resource was retrieved from the torchvision.models and shares ImageNet weights (Johnson, 2025).

2 Section 2: System Specification

The implementation of the project was on the Dell Inspiron 15 3520 laptop powered by 12th Gen Intel(R) Core(TM) i5-1235U with a frequency of 1.3 GHz, cores 10, and logical processors 12. The system was set up to a total of 16 GB installed physical memory (15.7 GB usable) and a total virtual memory 19.5 GB. It was run through Microsoft Windows 11 Home Single Language (Version 10.0.26100 Build 26100, w/ UEFI BIOS Version 1.30.0, dated 16-12-2024). Hardware configuration of the machine determined the data-intensive procedures and training of deep learning models quite effortlessly. Secure boot and kernel DMA protection were enabled, virtualization-based security was active with hypervisor-enforced code integrity to provide the enhanced protection. The storage configuration had two filesystem with space of 3.83 gigabyte as the page file space and a system directory found in C:\WINDOWS\system32. The use of multi-core Intel processor coupled with enough RAM and the use of virtualization created a redundant environment with which computations could be leveled. This specification supported the data processing efficiently and accelerated the model training, as well as its stable performance at the inference and test periods, which meant the project worked efficiently in a secure and competent environment of computing (He *et al*. 2025).

3 Section 3: Deep Learning Image Classification Workflow

```
import os
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader, WeightedRandomSampler
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
from pathlib import Path
import random
from PIL import Image
from collections import Counter
```

Figure 1: Used libraries

(Source: Self-created)

This code prepares an environment for a deep learning image classification task of transfer learning on ResNet18. It consists of necessary imports of the libraries used to preprocess the data, define the model, train it, evaluate it, and visualize the results. More critical modules of PyTorch and torchvision are utilized in conjunction with sklearn to obtain performance measures. The arrangement permits reproducibility, data management, and initialization of essential aspects of education and prediction.

```
# Set fixed random seed
torch.manual_seed(42)

# Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Transform: Resize to 128x128
transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
```

```
# Paths
root_path = Path("C:/Users/KEERTHANA REDDY/Documents/DISSERTATION/archive (16)")
train_path = root_path / "train"
test_path = root_path / "test"
```

```
# Detect class folders
class_folders = [folder for folder in train_path.iterdir() if folder.is_dir()]
class_names = [folder.name for folder in class_folders]
print("Detected classes:", class_names)

# Gather one random image from each class
sample_images = []
for folder in class_folders:
    image_files = list(folder.glob("*"))
    if image_files:
        sample_images.append((random.choice(image_files), folder.name))

# Display sample images with their class labels
plt.figure(figsize=(15, 5))
for i, (img_path, label) in enumerate(sample_images[:6]):
    img = Image.open(img_path)
    plt.subplot(1, 6, i + 1)
    plt.imshow(img)
    plt.title(label)
    plt.axis('off')
plt.tight_layout()
plt.show()
```

Figure 2: Dataset Setup and Visualization of Sample Images with Class Labels

(Source: Self-created)

This code comes in handy to read the dataset to be used in a deep learning image classification problem. It starts out by establishing a random seed to make reproductions possible and characterizing the computation device (GPU or CPU). An image resizing and normalization transform pipeline downsizes the images into a 128 x 128 pixel grid, converts to tensors, and normalizes. It assigns the paths of files used in training and testing, identifies the class folders in the training set, and reads the names of classes. It then chooses randomly each of the images in each class, and saves the path of the image along with the label. At last, it shows the chosen images with accompanying class labels by using Matplotlib.

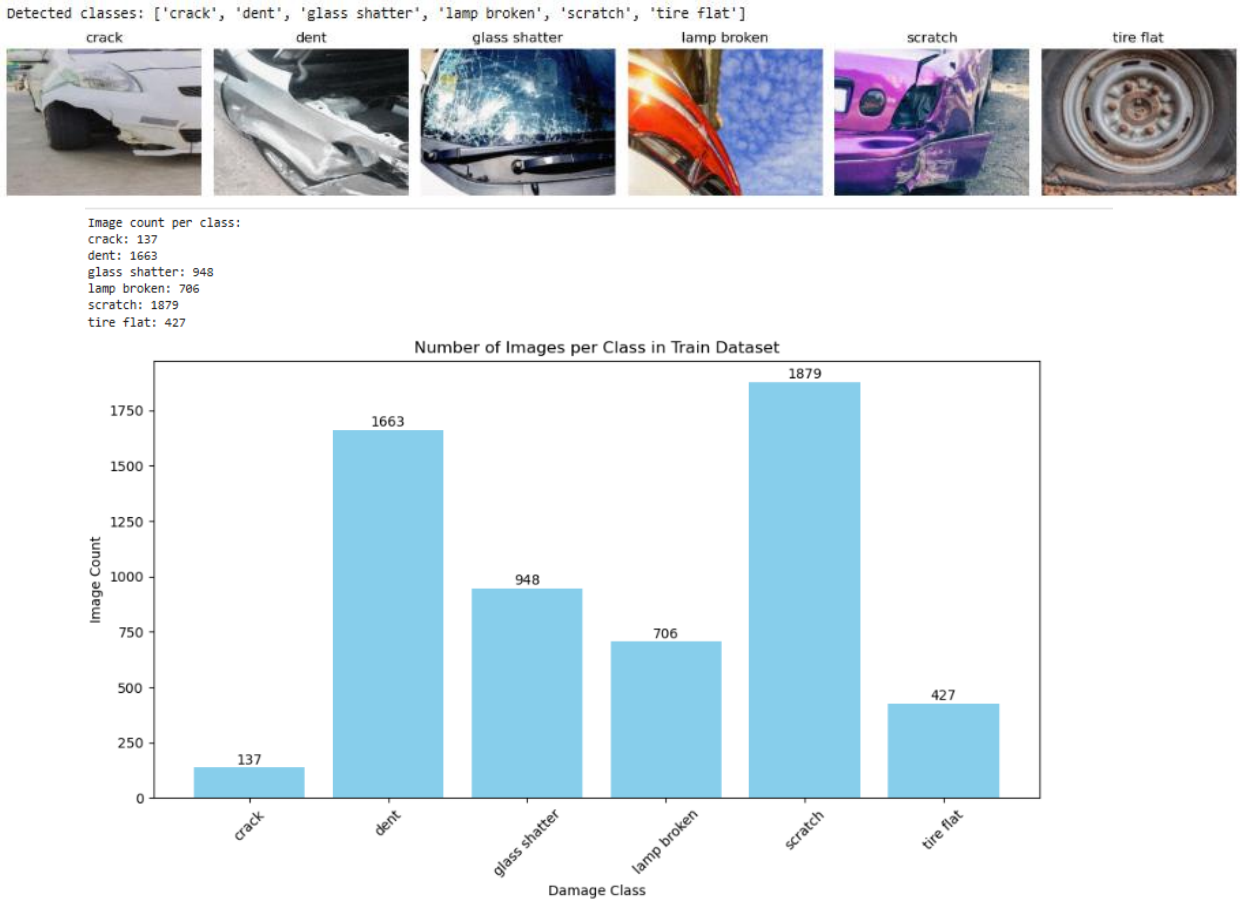

Figure 3: Class-wise Distribution of Vehicle Damage Types in the Training Dataset
(Source: Self-created)

The visual maps six categories of vehicle damage; they are crack, dent, glass shatter, lamp broken, scratch, and tire flat, with examples of each category of damage. The figure of frequency of the image in the training dataset is shown in the accompanying bar chart. The number of samples is the highest with scratch (1879) followed by dent (1663), and crack is the least represented as it possesses only 137 images. This is a problem of the biasedness of the predictions arising in such a case because of the underrepresentation of certain categories and training such a model. Automated vehicle condition detection requires sound weak augmentation or balancing methods in order to have accurate and fair classification for all types of damage to be detected accurately.

```
# Load datasets
train_dataset = datasets.ImageFolder(train_path, transform=transform)
test_dataset = datasets.ImageFolder(test_path, transform=transform)

# Split train into train/val (80/20)
train_size = int(0.8 * len(train_dataset))
val_size = len(train_dataset) - train_size
train_subset, val_subset = torch.utils.data.random_split(train_dataset, [train_size, val_size])

# WeightedRandomSampler for class imbalance
train_targets = [train_dataset[i][1] for i in train_subset.indices]
class_count = Counter(train_targets)
class_weights = [1.0 / class_count[i] for i in range(len(class_count))]
sample_weights = [class_weights[label] for label in train_targets]
sampler = WeightedRandomSampler(sample_weights, len(sample_weights))

# DataLoaders
batch_size = 32
train_loader = DataLoader(train_subset, batch_size=batch_size, sampler=sampler)
val_loader = DataLoader(val_subset, batch_size=batch_size)
test_loader = DataLoader(test_dataset, batch_size=batch_size)

# Number of classes
num_classes = len(train_dataset.classes)
class_names = train_dataset.classes
```

Figure 4: Dataset Loading, Splitting, and Handling Class Imbalance

(Source: Self-created)

In this part, the image datasets are loaded to train and test, and appropriate transformations are performed. The training set is divided into the training and validation subsets in the proportions of 80 to 20. A weighted random sampler is applied to resolve the problem of class imbalance in training data: a low-numbered class will have a higher chance of being sampled. It computes sample weights according to the class frequencies. The training, validation, and test sets will be designed with data loaders with a batch size of 32. There is also a calculation of the unique classes and the extraction of class names to be used later when training and evaluating such models (Li *et al*. 2025).

```
# Improved CNN using AdaptiveAvgPool2d
class ImprovedCNN(nn.Module):
    def __init__(self, num_classes):
        super(ImprovedCNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(), nn.MaxPool2d(2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(), nn.MaxPool2d(2),
            nn.AdaptiveAvgPool2d((1, 1))  # Reduces output to (batch, 128, 1, 1)
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),  # Flattens to (batch, 128)
            nn.Linear(128, 256),
            nn.ReLU(),
            nn.Dropout(0.4),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x
```

Figure 5: Improved Convolutional Neural Network with Adaptive Average Pooling

(Source: Self-created)

The ImprovedCNN class sets a convolutional neural network that is optimized on the basis of image classification. It uses three convolutional layers that use a ReLU activation of the feature map and maximum pooling of the feature vectors as an increment of depth. An important improvement is to use AdaptiveAvgPool2d((1, 1)) and have a spatial output size of

1x1, irrespective of the input size, so that overfitting and the number of parameters are minimal. The classifier part consists of a Flatten layer, two fully connected layers with a dropout regularization and ReLU activation in order to produce a better generalization. This architecture enhances training stability, efficiency, and accuracy, which is appropriate in the case of differentiating data in various levels of image sizes and distribution of classes.

```python
# Training function
def train_model(model, train_loader, val_loader, device, epochs=15, lr=0.001):
    model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)

    for epoch in range(epochs):
        model.train()
        train_loss, correct = 0.0, 0
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            train_loss += loss.item() * images.size(0)
            correct += (outputs.argmax(1) == labels).sum().item()

        train_loss /= len(train_loader.dataset)
        train_acc = correct / len(train_loader.dataset)

        # Validation
        model.eval()
        val_loss, val_correct = 0.0, 0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item() * images.size(0)
                val_correct += (outputs.argmax(1) == labels).sum().item()

        val_loss /= len(val_loader.dataset)
        val_acc = val_correct / len(val_loader.dataset)

        print(f"Epoch {epoch+1}/{epochs} | "
              f"Train Loss: {train_loss:.4f} Acc: {train_acc:.4f} | "
              f"Val Loss: {val_loss:.4f} Acc: {val_acc:.4f}")

    return model
```

Figure 6: Model Training Function

(Source: Self-created)

The train_model is a Python function to train and validate a PyTorch neural network. It accepts the model, data loaders, device, the number of epochs, and the learning rate. Training is done with Adam optimizer and CrossEntropyLoss. It executes forward and backward sweeps through the training data of each epoch, updates the weights, and calculates training accuracy. It operates, then, by testing the model on validation data without re-training weights, calculating loss and accuracy. The metrics used to train and validate are given per epoch, and the last trained model is returned to be subject to additional evaluation or testing.

```python
# 2. VGG16 (Transfer Learning)
print("\nTraining VGG16...")
vgg_model = models.vgg16(weights='IMAGENET1K_V1')
for param in vgg_model.parameters():
    param.requires_grad = False
vgg_model.classifier[6] = nn.Linear(4096, num_classes)
vgg_model.to(device)
vgg_model = train_model(vgg_model, train_loader, val_loader, device, epochs=15)
plot_confusion(vgg_model, test_loader, device, class_names)
```

Figure 7: VGG16 Transfer Learning Implementation

(Source: Self-created)

This snippet occurs as an example of the utilization of transfer learning, containing VGG16 in image classification. It starts by loading the VGG16 model, and ImageNet weights are loaded. In order to keep learned features, all the layers are frozen by keeping requires_grad as False; thus, weights are not updated during training. The input size of the last linear layer is changed to num_classes and is used to replace the last linear layer in the final classifier layer to match num_classes. The trained model is then transferred to the desired device and is trained with the train_model function. A confusion matrix is the last thing used to check the performance of models (Memon *et al*. 2025).

```
# 3. ResNet18 (Transfer Learning)
print("\nTraining ResNet18...")
resnet_model = models.resnet18(weights='IMAGENET1K_V1')
for param in resnet_model.parameters():
    param.requires_grad = False
resnet_model.fc = nn.Linear(resnet_model.fc.in_features, num_classes)
resnet_model.to(device)
resnet_model = train_model(resnet_model, train_loader, val_loader, device, epochs=15)
plot_confusion(resnet_model, test_loader, device, class_names)
```

Figure 8: ResNet18 Transfer Learning

(Source: Self-created)

In this part, the transfer learning (ResNet18 model) is applied to classify an image. ImageNet weights are loaded in the pre-trained ResNet18 model, and the parameters are frozen in order to save the learned features. The last fully connected (fc) layer gets replaced with a new linear layer of the same number of classes as we want our model to output by resnet_model.fc = nn.Linear(...). The adapted model is transferred to the selected device and is trained on the train_model function in 15 rounds. The classification performance of the model is then scored after training by plotting a confusion matrix on the test data using confusion.

4 References

He, W., Zhou, T., Xiang, Y., Lin, Y., Hu, J. and Bao, R., 2025, January. Deep learning in image classification: Evaluating VGG19's performance on complex visual data. In 2025 5th International Conference on Neural Networks, Information and Communication Engineering (NNICE) (pp. 261-265). IEEE.

Johnson, R., 2025. PyTorch Foundations and Applications: Definitive Reference for Developers and Engineers. HiTeX Press.

Li, X., Gong, B., Chen, X., Li, H. and Yuan, G., 2025. Alzheimer's disease image classification based on enhanced residual attention network. PloS One, 20(1), p.e0317376.

Memon, S.A., Ahmed, I., Mahar, M.A., Memon, G.A.A.A.A. and Fatima, S., 2025. A COMPREHENSIVE REVIEW OF CNN ARCHITECTURES FOR IMAGE RECOGNITION: ADVANCES AND OPEN CHALLENGES. Spectrum of Engineering Sciences, 3(5), pp.121-132.