Course/Programme:	MSc. Artificial Intelligence
Module Name and Code:	Applied Artificial Intelligence (AIN7301)
Student ID:	2011184
Student Names	Bogomil Iliev
Tutor:	Dr. Naveed Islam
Assessment Number:	2 of 2
Assessment Type:	Coursework
Assessment Title:	Portfolio
Indicative Word Count:	3000 words.
Weighting:	60% of overall module grade.
Submission Deadline:	28.05.2025 (no later than 23:59)
Submission Date:	28/05/2025 at 06:00 am
Learning Outcomes Assessed:	**LO2:** Apply appropriate AI algorithms in real-world contexts, demonstrating originality, creativity and independence. **LO3:** Critically evaluate the output of the innovative AI algorithms used to address complex problems.

Quality-Controlled EffientNet for Multi-Class Gallbladder Disease Detection on Ultrasound Input

Abstract

Timely diagnosis of gallbladder diseases is limited by the operator's workload and experience. A lightweight deep learning architecture is presented that automatically classifies nine gallbladder pathologies from ultrasound frames in real time. A data-centric audit of 10 692 images introduced objective blur, high-frequence noise and contrast metrics. Frames failing thresholds have been rejected or selectively enhanced with CLAHE and NL-means denoising, returning 8 792 quality-controlled input samples. Transfer learning has been applied to a single-channel tf-EffieciNet-V2-S backbone using class-weighted cross-entropy, mixed precision and early stopping. The model converged in seven epochs and achieved 0.998 macro-recall on the 10% test set. Separate classes returned overall recall of 0.97 and above.

Keywords: EfficientNet-B0, Gallbladder Disease, EfficientNet-V2-S

Contents

Table of Figures

Table of Tables

1. Introduction

Artificial Intelligence (AI) systems that absorb knowledge and insights straight from data, nowadays are gathering pace fast and are getting implemented in various industries and for different tasks like fraud detection in banking, demand foretelling in supply-chain logistics, and independent quality assurance on factory floors. In healthcare, deep learning (DL) architectures have proven equally effective or even surpassing human expertise in tasks such as diabetic-retinopathy grading and skin-cancer triage. Such advancements are possible because convolutional neural networks (CNNs) can automatically draw out picture features that conventional rule-established models would not spot (Kuzior *et al.*, 2023; Gulshan *et al.*, 2016; Esteva *et al.*, 2017; Goodfellow *et al.*, 2016).

1.1. Problem Statement

Every day, Gallbladder diseases strike 1 in 10 grown-ups and ultrasound (US) is the most common approach utilised to diagnose the conditions, due to its non-invasive nature. Still, the minute analysis of the imagery depends on the skilfulness of the operator and the load that the professional is under. In smaller hospitals a single expert may deliver and asses more that 150 patients per day, which can cause delays and inaccurate diagnosis between different variations of gallbladder diseases. Thus, with the current developments in AI, such an implementation can greatly support professionals by flagging up possible gallbladder diagnoses, hence reducing waiting times and improving accuracy to prevent mistakes and uplift proper treatment prescription (Blaivas *et al.*, 1999; Okaniwa, 2021).

1.2. Aims and Objectives

The current project aims to develop a lightweight, transferable AI model that automatically classifies nine gallbladder pathologies from grey-scale ultrasound frames with high macro-recall. The particular objectives are:

- Utilise and curate a publicly available gallbladder dataset.
- Establish a baseline for comparison purposes.
- Develop and deploy a lightweight model for the classification task.
- Test and explain predictions.

1.3. Proposed AI Solution

A **transfer learning** approach is adopted, rather than training an architecture from scratch. This is chosen in order to save on computing resources and the limited dataset availability. Pretrained EfficientNet encoders are fine-tuned on the curated dataset, where a comprehensive pre-processing plan is taking action to remove blurred/noisy frames and normalise contrast. The chosen EfficientNet-V2-S architecture delivers ImageNet accuracy with 24 million parameters. Thus, highlighting as a good fit for deployment on a GPU-equipped ultrasound or ARM device. The model is supposed to detect critical gallbladder conditions with high recall values, thus uplifting the professionals' ability to diagnose different disease variations, speed up diagnosing times, reduce costs, and improve proper treatment prescriptions (Tan and Le, 2021).

2. Literature Review

2.1. Gallbladder Ultrasound And AI

US is the first technology that is employed when there is suspicion of a gallbladder disease (GBD). Nowadays, it is cost-effective, non-invasive, and free from radiation. Also, equipment is easily movable and compact. However, the accuracy of the approach is highly dependant on the skill level of the operator. A recent scoping review identified only 24 peer-reviewed studies applying AI to GBD US, when compared to similar studies assessing different organs like thyroid or liver. This is due to the fact that there are just a handful of large enough and publicly available GBD datasets. Most works target a single pathology, making the multiclass diagnosis field largely unexplored (Blaivas *et al.*, 1999; Wang *et al.*, 2025; Takahashi *et al.*, 2024; Hasan *et al.*, 2025; Wang *et al.*, 2025).

2.2. DL classifiers for GBD

Early attempts relied on handcrafted texture features combined with support-vector machines, achieving less than 80% accuracy on small data frames. With the advancements in CNNs the first nine-class classifier, trained on a dataset consisting of ten thousand images emerged. This work benchmarked four DL classifiers for nine gallbladder pathologies. Through contrast enhancement and segmentation, MobileNet achieved the best performance of 98.35% accuracy, showcasing that lightweight CNNs can deliver output with human expertise skilfulness (Vanitha *et al.*, 2022; Obaid *et al.*, 2023).

Another paper approached the same nine-class task with a feature engineering approach. They fused descriptors extracted from three pretrained CNN backbones into a content-based image-retrieval pipeline and ranked candidates via cosine similarity. The resulting model reached a mean average precision of 0.94, outperforming six standalone CNNs and two classical texture methods, and offering intuitive visual retrieval for clinician review.

Nonetheless, this algorithm has 138 million parameters and is quite computationally expensive, which makes its real-world implementation questionable (Bozdag *et al.*, 2025).

2.3. Efficient Architectures and Transfer Learning

In medical imaging AI applications, the trend has shifted from heavyweight ResNet/VGG models towards ones that have more efficient parameters like MobileNet and EfficientNet. EfficientNet-V2 demonstrates faster convergence and higher accuracy on ImageNet. Later research showcased that the EfficientNet architecture performs outstandingly on limited medical datasets. The EfficientNet-B7 variation was utilised through targeted augmentation and hit 97% AUC. No published research has been discovered while researching for the current work that employs EfficientNet-V2 for multi-class classification of GBD via US images. Also, it has not yet been compared to a conventional EfficientNet-B baseline under identical augmentation and loss conditions (Tan and Le, 2021; Shim *et al.*, 2024; Latha *et al.*, 2024).

2.4. Quality Control of Data

Well-known issues of US imagery are speckle noise, motion blur and low contrast. Still, a high number of works either do not disclose the specific preprocessing steps or apply only generic resizing and normalisation techniques. Recent data-oriented research show that targeted frame rejection and image-specific corrections like adaptive histogram equalisation (CLAHE) prior to training can actually raise sensitivity. During research for the current work, no scientific paper was identified to focus on GBD that tackles blur, high-frequency noise and contrast (Takahashi *et al.*, 2024; Yoshimi *et al.*, 2024).

3. Methodology

3.1. AI Techniques and Algorithms

As a **baseline** for the current project a pretrained version of the EfficientNet-B0 on the ImagNet dataset was selected. It boasts 5.3 M parameters and is widely accepted as a strong and compact CNN that sets a reproducible reference point. The proposed model of choice is a pretrained tf-EfficientNet-V2-S with 24M parameters, a dropout of 0.25 and drop-connect of 0.2. This is a newer architecture version with speedier convergence and higher accuracy per FLOP. The model is still light enough for the specified task and has the potential to be easily transformed and utilised on medical and ARM devices(Tan and Le, 2021; Shim *et al.*, 2024).

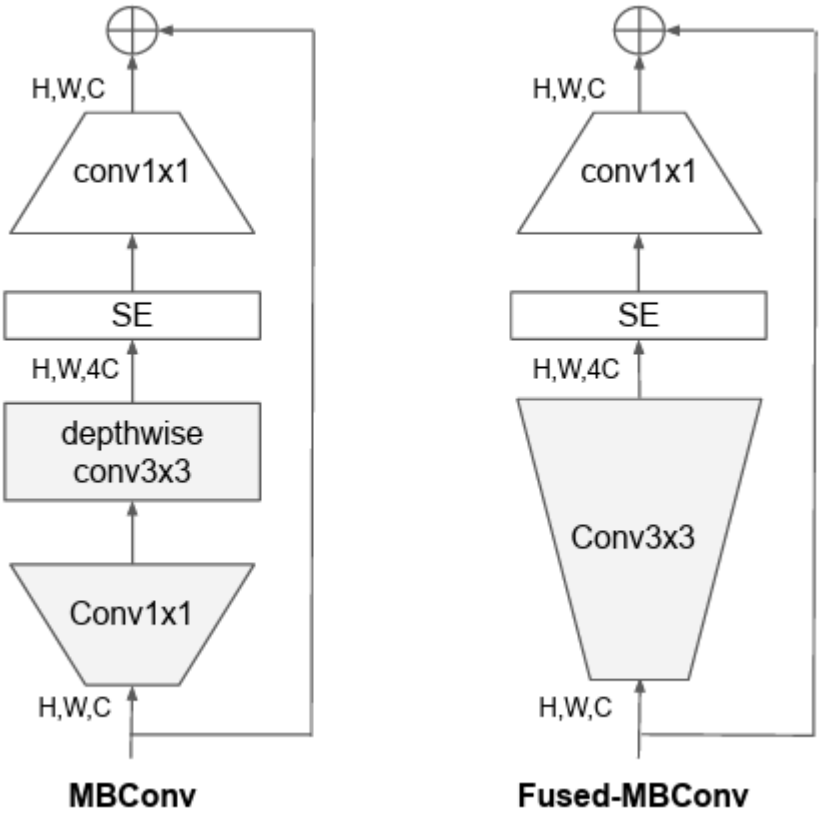

Figure 1 - EfficientNet-V2 Architecture (Tan and Le, 2021)

The models were trained on the new data through **transfer learning** by applying a two-stage approach. The first step is initialised with the ImageNet-21k weights, where the backbone is

frozen for two epochs, then fine-tuning is applied in epochs 8 to 30. In such a manner the generic edge and texture filters are maintained, while allowing adaptation to the new domain and cutting training time by nearly 60%. This was specifically aimed at due to time and resource constraints. Thus, the application of the latter approach (Brownlee, 2024).

AdamW optimiser has been utilised because it is stable on small batches and diminishes manual LR decay. For the **loss** class-weighted cross-entropy is employed to mitigate the imbalance. As for the **regularisation** the model has a built-in stochastic depth and classifier dropout, which reduces overfitting, thus no additional drop out has been applied. Also, some data augmentation techniques have been used. **Early stopping** has also been used to reduce the training time and save the best model state, where the patience has been set to 3 epochs on validation macro-recall (Brownlee, 2024).

3.2. Data Collection and Preprocessing

For this project, a publicly available **dataset** has been used, which is the **Gallbladder Diseases Dataset**, that is available at: https://data.mendeley.com/datasets/r6h24d2d3y/1

It contains 10 692 JPG images, segregated into nine classes according to the diagnose they depict (Turki *et al.*, 2024).

As **preprocessing** an **initial sanity check** has been performed, where the resolution histogram exemplifies a majority of 512x512 images, that are in the RGB format, but 60.8 % are pseudo-grayscale. Some **Quality Control (QC) measures** have been applied where the Blur is a variance of Laplacian with a threshold of less than 5. High-frequency (HF) energy is equal to percentage of FFT power outside central low-frequency band and the threshold is set to more than 22%. Contrast threshold is set to less than 25 and the frames failing any rule have been discarded (Approximately 19%). Batch size is set to 32.

Some **image-specific** corrections have also been used. The CLAHE approach is utilised if the contrast is less than 30 and Median denoise if HF is more than 18%. Standardisation techniques employed include the conversion of the remaining images to a single-channel, which have then been resized to 224x224, augmentation only on the training set via horizontal and vertical flipping, +/- 15% brightness/contrast jitter, Gaussian blur, and random rotation of +/- 10 degrees.

The dataset has then been split into three subsets for training, validation and testing in ration of 75/15/10. This resulted into train(6852), validation(1370) and test(913) sets.

3.3. Evaluation and Analysis Approach

To evaluate and analyse the results the current work employed **macro recall** on every training epoch, because it treats each class equally and is recommended for imbalanced medical detasets, due to the fact that a high overall accuracy can hide missed rare conditions. **Early stoppings** save the best state whenever whenever validation macro recall improves, if it does not it stops within three epochs. This prevents overfitting. A **confusion matrix** is also generated to expose specific error modes. A **per class recall** (TP/(TP+FN) is also calculated to highlight if some of the diagnoses lag behind others. **Loss** development is also observed during training and validation. As a final experiment the proposed model is manually tested by providing it unfamiliar input of three pathologies randomly sourced and downloaded from Google Images (Sokolova and Lapalme, 2009; Caruana *et al.*, 2000; Chicco and Jurman, 2020; Bhati *et al.*, 2024).

4. Implementation

For the implementation of this project Google Colab Environment was used and the code was written in Python. The baseline and proposed model have been trained on a Nvidia A100 GPU. Each one of them has initially been set to train loop in 30 epochs over the data.

The source code for this project can be found through the Google Colab Notebook located at: https://colab.research.google.com/drive/1-53YzvzmT8wGZK4mkkQ8UuRsW-wYAxyp?usp=sharing

4.1. Exploratory Data Analysis (EDA)

In Figure 2, 3, and 4 is the starting point for the EDA. A table of the top resolutions is produced with the number of associated pictures and a plot of 1 random image from each class. Thus, in Figures 5, 6, 7 and 8 can be observed that the majority of 10639 pictures have dimensions of 1200x900 pixels and the remaining are spread around 3 more resolutions and the respective code to produce these plots.

```python
# EDA
# Importing of Libraries needed
import matplotlib.pyplot as plt
from PIL import Image
import random, pandas as pd
from collections import Counter
from IPython.display import display

# Quick visual sanity-check by randomly plotting images from each class.

def show_random_grid(n: int = 12):
    """Display an n-image grid of randomly sampled ultrasound frames."""
    sample = random.sample(image_paths, n)
    plt.figure(figsize=(12, 6))
    for i, p in enumerate(sample):
        plt.subplot(3, 4, i + 1)
        plt.imshow(Image.open(p).convert("L"), cmap="gray")
        plt.title(p.split(os.sep)[-2], fontsize=8)
        plt.axis("off")
    plt.tight_layout()

show_random_grid()

# Review of Image resolution distribution
# Collect (width, height) for every image in the dataset
sizes = [Image.open(p).size for p in image_paths]  # (w, h)
size_counts = Counter(sizes)

# Tabulate the most frequent resolutions
resolution_df = pd.DataFrame(
    [{"width": w, "height": h, "count": c} for (w, h), c in size_counts.items()]
).sort_values("count", ascending=False)

print("\nTop 10 image resolutions (w × h):")
display(resolution_df.head(10))

```

Figure 2 - EDA Code 1

Top 10 image resolutions (w × h):

	width	height	count
0	1200	900	10639
3	900	1200	46
1	1170	876	5
2	2400	1800	2

Figure 3 - Top 10 Image Resolutions

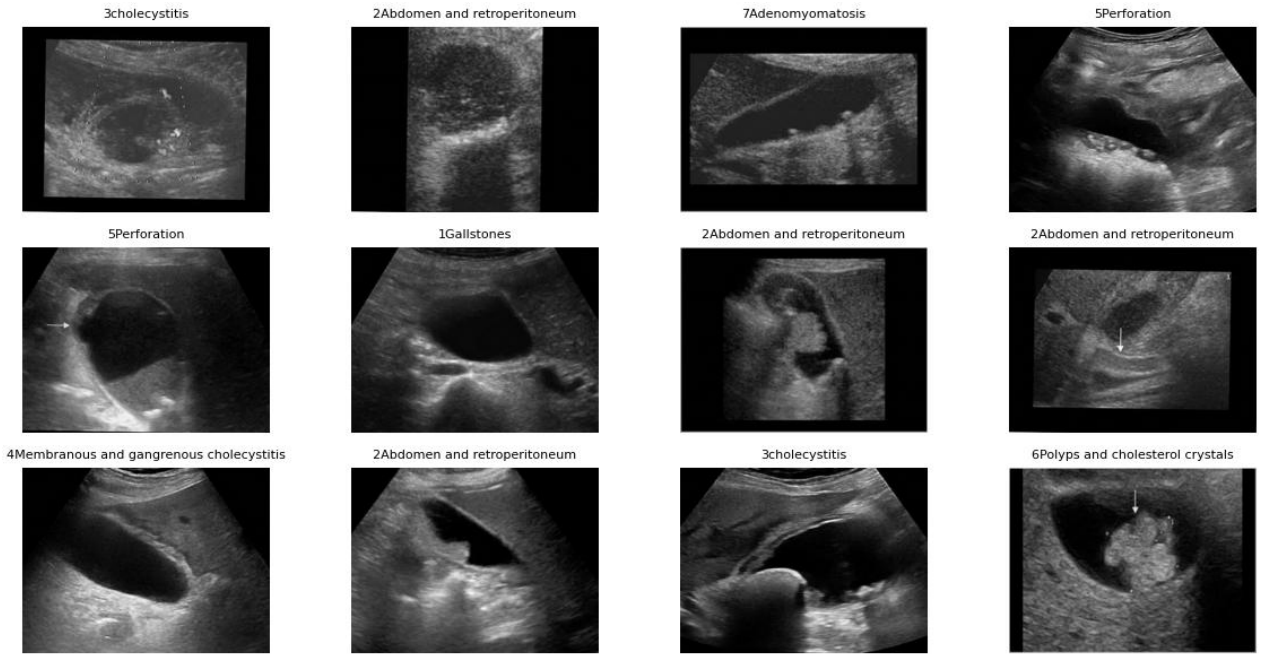

Figure 4 - One Example Image per Class

```
# Creating a 2-D histogram (heat-map) of all widths vs heights
widths  = [w for (w, _) in sizes]
heights = [h for (_, h) in sizes]

plt.figure(figsize=(6, 5))
plt.hist2d(widths, heights, bins=[30, 30])
plt.xlabel("Width (px)")
plt.ylabel("Height (px)")
plt.title("Resolution distribution across dataset")
plt.colorbar(label="Number of images")
plt.tight_layout()
plt.show()
```

Figure 5 - Code for two-dimensional histogram (heatmap)

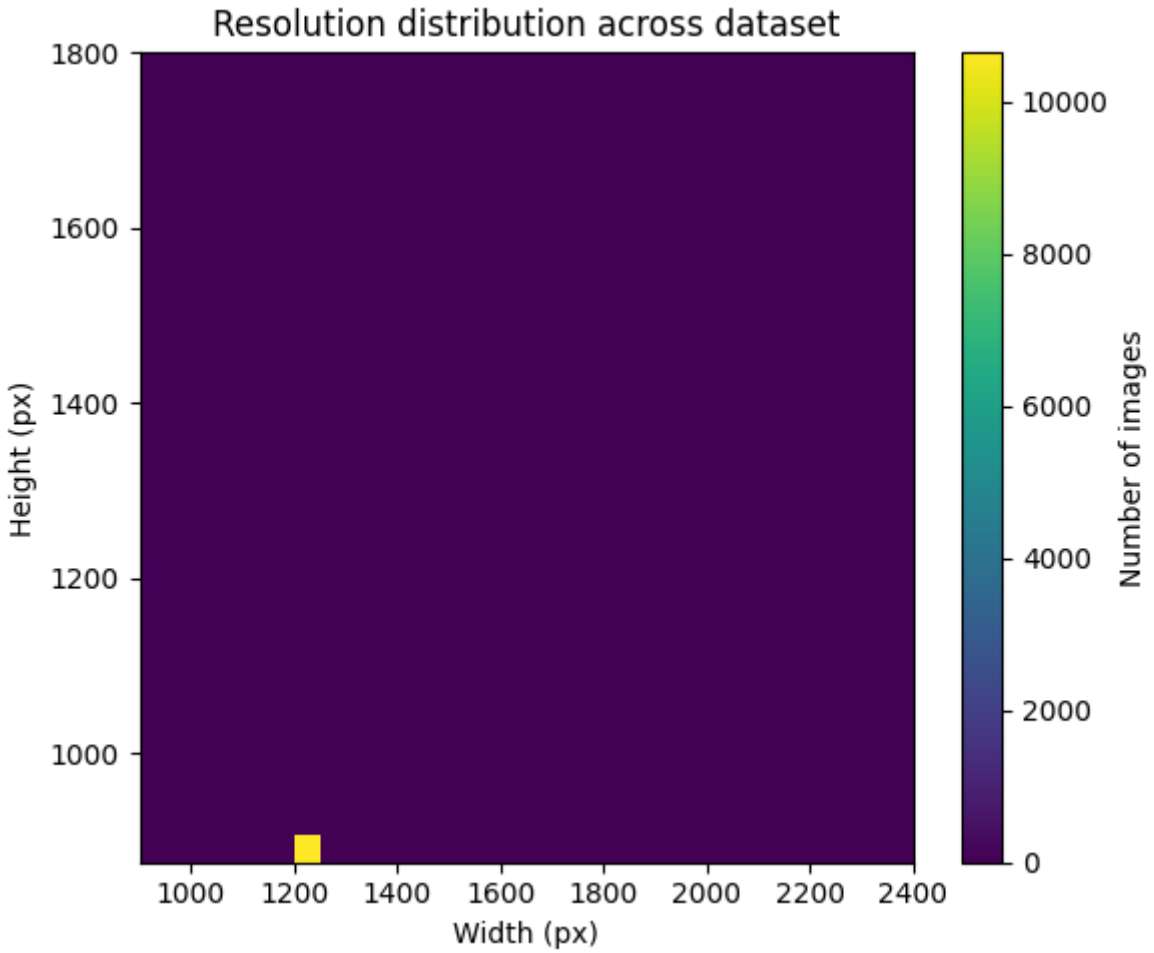

Figure 6 - Two-dimensional Histogram (Heatmap) of Image Size Distributions.

```python
# Creating a Bubble plot
top_n = 10

# Count (width, height) pairs and take the most common
pairs = Counter(sizes).most_common(top_n)
w_top  = [wh[0][0] for wh in pairs]    # widths
h_top  = [wh[0][1] for wh in pairs]    # heights
counts = [wh[1]     for wh in pairs]   # frequencies

plt.figure(figsize=(8,6))
plt.scatter(w_top, h_top, s=[c*6 for c in counts], alpha=0.6)  # bubble size / count
for w, h, c in zip(w_top, h_top, counts):
    plt.text(w, h, str(c), ha="center", va="center", fontsize=8)

plt.xlabel("Width (px)")
plt.ylabel("Height (px)")
plt.title(f"Top-{top_n} resolutions (bubble size = #images)")
plt.grid(True, linestyle="--", linewidth=0.3)
plt.show()
```

Figure 7 - Code for Bubble Plot of Image Dimensions

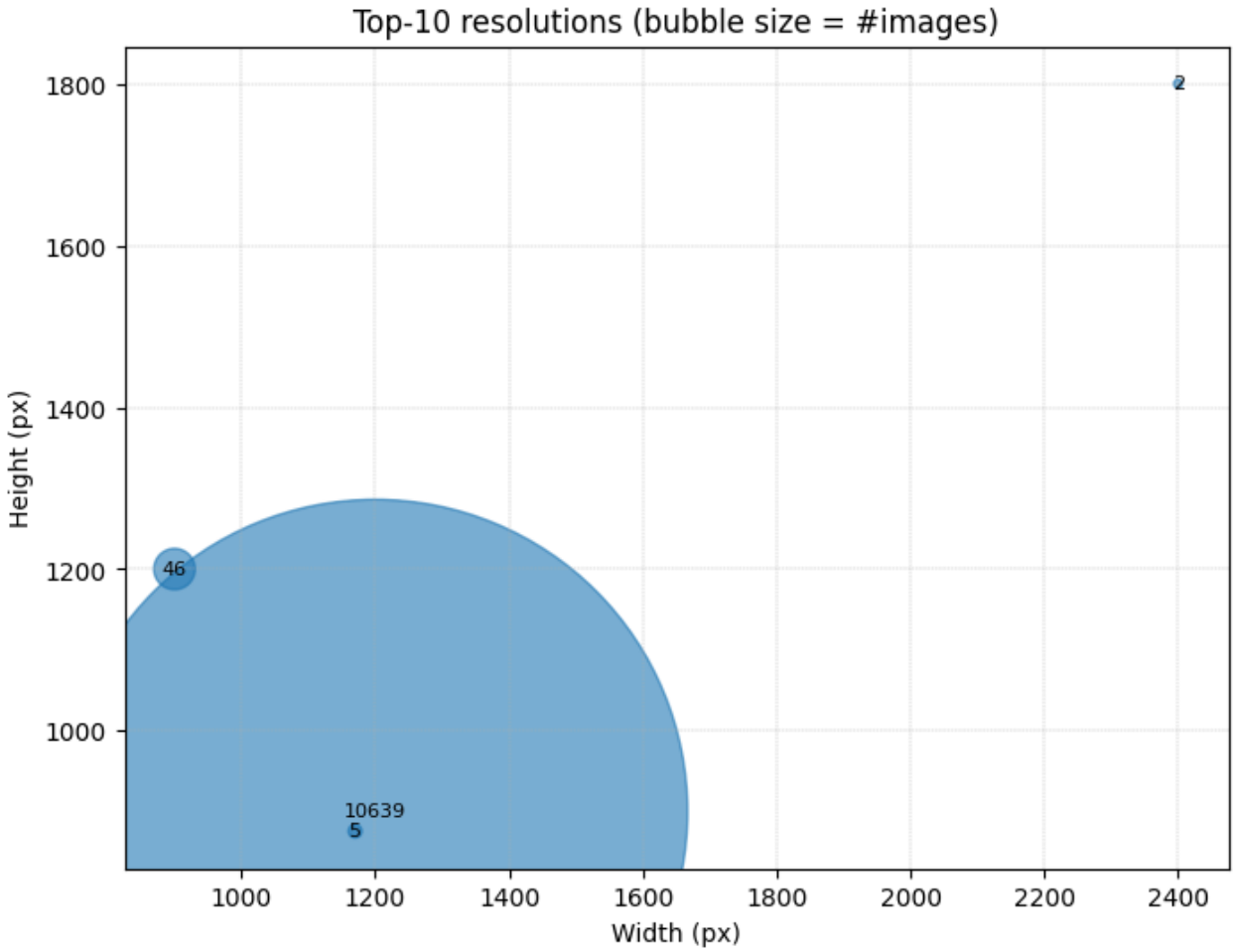

Figure 8 - Bubble Plot of Image Dimensions

Figure 9 portrays the code to perform grey scale vs. colour analysis. Here, every file is checked, and Pillow image mode is assigned to it (L, RGB, RGBA). In this manner pseudo-greyscale RGB files are identified. Figure 10 shows that 60.8 % of RGBs are actually pseudo-greyscale and 39.2% contain some non-zero colour variation. All files have been saved as RGB. Thus, majority of frames carry no colour information.

```python
# Grayscale vs. Colour analysis
import textwrap
# Scanning of every image (or sample, if ~10 k feels slow)
scan_all = True        # flip to False to sample 2 000 files for speed
sample_size = 2_000

to_scan = image_paths if scan_all else random.sample(image_paths, sample_size)

mode_counter   = Counter()   # e.g. "L", "RGB", "RGBA"
pseudo_gray    = 0           # RGB triplets where all channels are equal
failed_reads   = 0

for p in tqdm.tqdm(to_scan, desc="Scanning images"):
    try:
        with Image.open(p) as im:
            mode_counter[im.mode] += 1
            # Check for 3-channel "gray masquerading as colour"
            if im.mode == "RGB":
                arr = np.asarray(im)
                if np.all(arr[..., 0] == arr[..., 1]) and np.all(arr[..., 1] == arr[..., 2]):
                    pseudo_gray += 1
    except Exception as e:
        failed_reads += 1

# Report Creation
total = len(to_scan)
print("\nImage mode counts:")
for m, c in mode_counter.most_common():
    print(f"  {m:<4} : {c}  ({100*c/total:.1f} %)")

if pseudo_gray:
    print(f"\n  From the {mode_counter['RGB']} RGB files, {pseudo_gray} "
          f"({100*pseudo_gray/max(1, mode_counter['RGB']):.1f} %) are pseudo-grayscale "
          "(all three channels identical).")

if failed_reads:
    print(f"\n{failed_reads} files could not be read.")

# Decision helper
msg = []
if mode_counter.get("RGB", 0) - pseudo_gray == 0 and mode_counter.get("RGBA", 0) == 0:
    msg.append("All images are effectively single-channel,thus you can safely convert to 1-channel tensors.")
else:
    msg.append("There are genuine colour images,hence converting to grayscale will lose information.")
    if pseudo_gray:
        msg.append("However, many RGBs are pseudo-grayscale, meaning  you could auto-detect and drop extra channels only for those.")

print("\n" + textwrap.fill(" ".join(msg), width=100))
```

Figure 9 - Code for Grey Scale vs. Colour Analysis

```
Scanning images: 100%|██████████| 10692/10692 [01:29<00:00, 119.48it/s]
Image mode counts:
  RGB  : 10692  (100.0 %)

  From the 10692 RGB files, 6500 (60.8 %) are pseudo-grayscale (all three channels identical).

There are genuine colour images,hence converting to grayscale will lose information. However, many
RGBs are pseudo-grayscale, meaning  you could auto-detect and drop extra channels only for those.
```

Figure 10 - Colour Analysis Output

Figure 11 represents the plots for Artifact and Noise Analysis, where the variance of Laplacian is assessed to showcase the Blur in images, the HF energy to represent the noise, and Contrast to see the RMS standard pixel deviation. The Blur histogram shows a tall spike

between 4-8 and a long right tail, where the frames with blur less than 5 can be considered outliers. The noise histogram is with near normal distribution centered on 14%, hence frames with more than 22% can be dropped or denoised. The contrast histogram is showing a bell curve centring around 40 and has a left tail, meaning frames with contrast less than 25 lack diagnostic detail and can be excluded, but the ones that are less than 30 and equal or larger than 25 can be CLAHE enhanced. The quality landscape exemplifies two clear clusters – low blur (mid noise) and high blur (low noise). Thus, the threshold can be set to disqualify the poor images, without fear that the majority of pictures will be affected. Last are six examples of low-quality images.

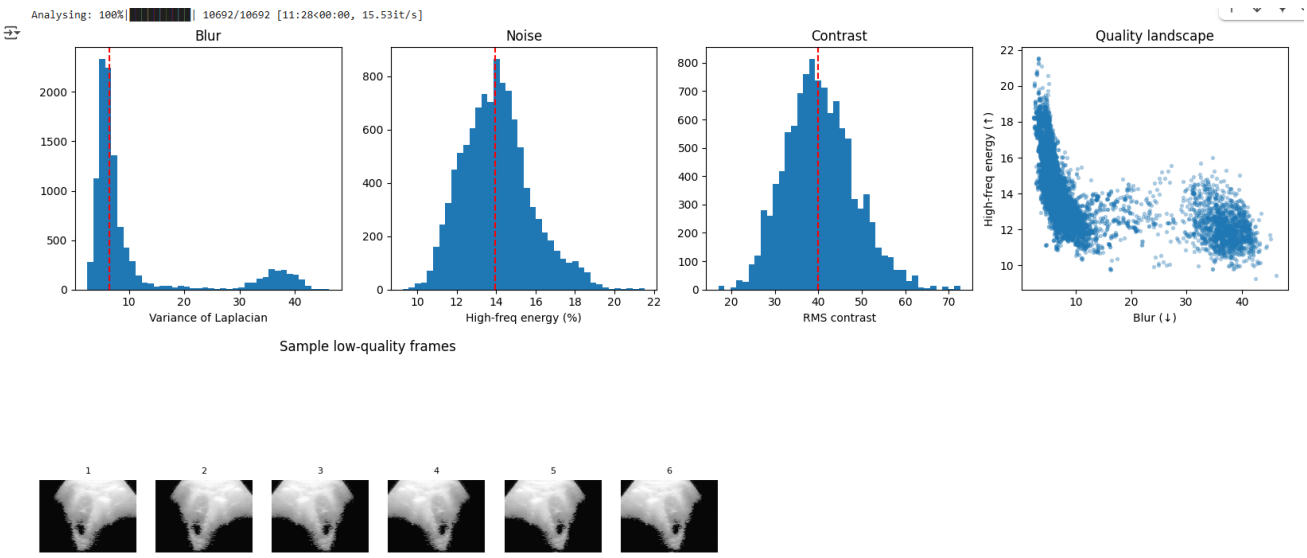

Figure 11 - Artifact / Noise Analysis Plots.

4.2. Data Preprocessing and Preparation

In Figure 12, a QC threshold is set to filter out the inappropriate images where the images blurrier than the variance of 5 Laplacian are going to be dropped, ones with HF energy exceeding 22%, and RMS contrast below 25. Ban_mask creates a Boolean mask that flags any picture failing one or more of these conditions. Thus, only 8548 from 10692 shots are of acceptable quality (Figure 13).

```
# QC thresholds
MIN_BLUR, MAX_NOISE, MIN_CONTR = 5, 22, 25

ban_mask = (
    (df_qc.blur_var < MIN_BLUR) |
    (df_qc.hp_energy > MAX_NOISE) |
    (df_qc.contrast  < MIN_CONTR)
)
approved = df_qc[~ban_mask].path.tolist()
print(f"Approved images: {len(approved)} / {len(df_qc)}")

```

Figure 12 - Setting up of QC Thresholds.

```
Approved images: 8548 / 10692
```

Figure 13 - Output of Filtering QC Threshold.

Figure 14 depicts the code defining the helper functions to apply CLAHE enhancement where needed. It is performed with tiles 8x8, where denoise applies fast non-local means filtering. In such a manner the diagnostic detail in low-contrast scans is recovered, meanwhile NL-means push down speckle noise evading edge blur, which is useful in US preprocessing. Then the constructor stores the QC dataframe and builds a cls2idx dictionary. _getitem_

loads a frame in grayscale, looks up its QC row, and dynamically corrects it. Hence, only 145 of retained items are modified (Yoshimi *et al.*, 2024).

```python
# Helpers
import cv2, numpy as np
from PIL import Image
def apply_clahe(arr):
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    return clahe.apply(arr.astype(np.uint8))
def denoise(arr):
    return cv2.fastNlMeansDenoising(arr.astype(np.uint8), h=10)

# Custom Dataset
from torch.utils.data import Dataset
class UltrasoundDS(Dataset):
    def __init__(self, paths, tfms, qc_df):
        self.paths, self.tfms = paths, tfms
        self.qc = qc_df.set_index("path")
        self.cls2idx = {c:i for i,c in enumerate(sorted(set(labels)))}

    def __len__(self): return len(self.paths)

    def __getitem__(self, idx):
        p   = self.paths[idx]
        arr = np.asarray(Image.open(p).convert("L"))

        # on-the-fly corrections
        row = self.qc.loc[p]
        if row.contrast < 30:
            arr = apply_clahe(arr)
        if row.hp_energy > 18:
            arr = denoise(arr)

        x = self.tfms(Image.fromarray(arr))
        y = self.cls2idx[p.split(os.sep)[-2]]
        return x, y

```

Figure 14 - Code for Helper Function for selective CLAHE Enhancement and on-the-fly dynamic Correction

Figure 15 depicts the code where the train/val/test transforms are performed with random horizontal flips of +/- 10 degrees, Gaussian blur of 0.3, +/-15% contrast jitter, and single-channel normalisation. Also, resize is applied. This is done to copy probe angulation and gain

variation, proven to reduce over-fitting in ultrasound. Validation and test sets remain untouched. Then the ratio of 75/15/10 is used to split the data in two stages and ensure rare classes appear in every subset, preventing undefined recall values and unwanted early stops. The three datasets wrap the paths with QC on-the-fly corrections to warrant that the CLAHE/denoising process applies evenly across all loaders. The class sampler provides a uniform class representation in every minibatch of 32 of the training set.

```python
from torchvision import transforms
from torch.utils.data import DataLoader, WeightedRandomSampler
from sklearn.model_selection import train_test_split
from collections import Counter
import torch, numpy as np

# Transforms
IMG_SIZE = 224
train_tfms = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.RandomApply([transforms.GaussianBlur(3)], p=0.3),
    transforms.ColorJitter(brightness=0.15, contrast=0.15),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5]),
])
val_tfms = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5]),
])

# 75 / 15 / 10 split
train_p, temp_p = train_test_split(
    approved, test_size=0.25,          # 75 % train, 25 % temp
    stratify=[p.split(os.sep)[-2] for p in approved],
    random_state=SEED
)
val_p, test_p = train_test_split(
    temp_p, test_size=0.40,            # 40 % of 25 %  => 10 % overall
    stratify=[p.split(os.sep)[-2] for p in temp_p],
    random_state=SEED
)

print(f"train {len(train_p)}  val {len(val_p)}  test {len(test_p)}")

# Datasets
train_ds = UltrasoundDS(train_p, train_tfms, df_qc)
val_ds   = UltrasoundDS(val_p,   val_tfms,   df_qc)
test_ds  = UltrasoundDS(test_p,  val_tfms,   df_qc)   # no augmentations

# Sampler & loaders
freqs = Counter([p.split(os.sep)[-2] for p in train_p])
weights = torch.tensor([1/freqs[c] for c in sorted(freqs.keys())], dtype=torch.float)
sample_w = [weights[train_ds.cls2idx[p.split(os.sep)[-2]]] for p in train_p]
sampler  = WeightedRandomSampler(sample_w, len(sample_w), replacement=True)

train_dl = DataLoader(train_ds, batch_size=32, sampler=sampler, num_workers=2, pin_memory=True)
val_dl   = DataLoader(val_ds,   batch_size=32, shuffle=False, num_workers=2, pin_memory=True)
test_dl  = DataLoader(test_ds,  batch_size=32, shuffle=False, num_workers=2, pin_memory=True)
```

Figure 15 - Augmentation and Three-way Loaders Code

```
train 6411  val 1282  test 855
```

Figure 16 - Final Dataset Split

4.3. Baseline and Model Training

In Figure 16 can be observed the code that create the model factory and access the pretrained models through the timm library, torch and different packages from it to call the evaluation metric wrapper and mixed-precission utilities. The in_chans=1 forces the first convolution to accept single-channel shots. Num_classes defines the classifier head, and drop_rate, drop_path_rate introduce dropout and stochastic depth. Weights are loaded from the backbone.

```
# MODEL FACTORY
!pip install -q timm

import timm, torch, torch.nn as nn, copy, time
import torch.optim as optim
from torchmetrics.classification import MulticlassRecall
from torch.cuda.amp import autocast, GradScaler

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

def make_model(name: str,
               in_chans: int = 1,
               n_classes: int = 9,
               drop_rate: float = 0.2,
               drop_path: float = 0.2) -> nn.Module:
    """
    Returns a timm vision model ready for single-channel inputs.
    """
    model = timm.create_model(
        name,
        pretrained=True,
        in_chans=in_chans,
        num_classes=n_classes,
        drop_rate=drop_rate,           # classifier dropout
        drop_path_rate=drop_path       # stochastic depth
    )
    return model.to(device)

```

Figure 17 - Model Factory Code

```python
def train_finetune(model_name="efficientnet_b0",
                   head_lr=3e-3, ft_lr=1e-4,
                   head_epochs=2, ft_epochs=30,
                   patience=3, drop_rate=0.2, drop_path=0.2):

    model = make_model(model_name, drop_rate=drop_rate, drop_path=drop_path)
    criterion = nn.CrossEntropyLoss(weight=weights.to(device))

    #Containers to log each epoch
    hist = {
        "epoch":    [],
        "phase":    [],          # "head" or "ft"
        "train_loss": [], "val_loss": [],
        "train_rec":  [], "val_rec":  []
    }

    # STAGE 1 (head frozen)
    for n,p in model.named_parameters():
        if "classifier" not in n and "fc" not in n:
            p.requires_grad = False
    opt = optim.AdamW(filter(lambda p: p.requires_grad, model.parameters()),
                      lr=head_lr, weight_decay=1e-4)
    scaler = GradScaler()
    recall = MulticlassRecall(num_classes=9, average="macro").to(device)

    def run_epoch(train: bool):
        loader = train_dl if train else val_dl
        model.train() if train else model.eval()
        recall.reset(); epoch_loss = 0.
        for xb, yb in loader:
            xb, yb = xb.to(device), yb.to(device)
            with autocast():
                preds = model(xb)
                loss  = criterion(preds, yb)
            if train:
                opt.zero_grad(set_to_none=True)
                scaler.scale(loss).backward()
                scaler.step(opt); scaler.update()
            epoch_loss += loss.item()*xb.size(0)
            recall.update(preds, yb)
        return epoch_loss/len(loader.dataset), recall.compute().item()

    for ep in range(1, head_epochs+1):
        tr_loss, tr_rec = run_epoch(True)
        val_loss, val_rec = run_epoch(False)
        print(f"[HEAD] Ep{ep:02d}  trainR={tr_rec:.3f}  valR={val_rec:.3f}")

        #log
        hist["epoch"].append(ep); hist["phase"].append("head")
        hist["train_loss"].append(tr_loss); hist["val_loss"].append(val_loss)
        hist["train_rec"].append(tr_rec);   hist["val_rec"].append(val_rec)

    # STAGE 2 (full fine-tune)
    for p in model.parameters(): p.requires_grad = True
    opt   = optim.AdamW(model.parameters(), lr=ft_lr, weight_decay=1e-4)
    sched = optim.lr_scheduler.CosineAnnealingLR(opt, T_max=ft_epochs)

    best_rec, bad, best_state = 0., 0, None
    for ep in range(1, ft_epochs+1):
        tr_loss, tr_rec = run_epoch(True)
        val_loss, val_rec = run_epoch(False)
        sched.step()

        hist["epoch"].append(head_epochs+ep); hist["phase"].append("ft")
        hist["train_loss"].append(tr_loss);   hist["val_loss"].append(val_loss)
        hist["train_rec"].append(tr_rec);     hist["val_rec"].append(val_rec)

        good = val_rec > best_rec + 1e-4
        if good:
            best_rec, bad = val_rec, 0
            best_state = copy.deepcopy(model.state_dict())
        else:
            bad += 1
        print(f"[FT ] Ep{ep:02d}  trainR={tr_rec:.3f}  valR={val_rec:.3f} "
              f"{'**BEST**' if good else ''}")
        if bad >= patience:
            print("Early stopping triggered."); break

    model.load_state_dict(best_state)
    torch.save(model.state_dict(), f"{model_name}_best.pt")
    print(f"Best val macro-recall: {best_rec:.3f}")
    return model, best_rec, hist        #return history dict
```

Figure 18 - Training Loop Code for Both Models

Figure 18 exemplifies the training loop for the two models. It is setup as a two-stage training process, where Stage 1 freezes all layers except for classifier for 2 epochs and Stage 2 unfreezes the whole network for full fine-tuning. In such a manner the “warm up” stage trains only the new task-specific head, preventing large, random gradients from damaging pretrained weights. Following, the full fine-tune then adapts deep features. The AdamW optimiser combines the adaptive learning rates with decoupled L2 regularisation. Early stopping is set to a patience of 3 epochs, where the best model state is saved. Figure 19 depicts the rest of the training and validation code that simply produces the loss and recall curve plots for both models and instantiates the training process (Howard and Ruder, 2018; Caruana *et al.*, 2000).

4.3. Plotting Helpers of Train Validation Results.

```
def plot_loss(history):
    plt.figure(figsize=(6,4))
    plt.plot(history["epoch"], history["train_loss"], label="train loss")
    plt.plot(history["epoch"], history["val_loss"],   label="val loss")
    plt.xlabel("epoch"); plt.ylabel("loss"); plt.title("Loss curve")
    plt.legend(); plt.grid(True); plt.show()

def plot_recall(history):
    plt.figure(figsize=(6,4))
    plt.plot(history["epoch"], history["train_rec"], label="train recall")
    plt.plot(history["epoch"], history["val_rec"],   label="val recall")
    plt.xlabel("epoch"); plt.ylabel("macro-recall"); plt.title("Recall curve")
    plt.legend(); plt.grid(True); plt.show()
```

4.4. Training of Models and Generating the Plots to compare the training process.

```
# baseline model training - efficientnet_b0
base_model, base_best, base_hist = train_finetune("efficientnet_b0", ft_epochs=30)

plot_loss(base_hist)
plot_recall(base_hist)

# main model training - tf_efficientnetv2_s
main_model, main_best, main_hist = train_finetune("tf_efficientnetv2_s", ft_epochs=30,
                                                  drop_rate=0.25, drop_path=0.2)


plot_loss(main_hist)
plot_recall(main_hist)
```

Figure 19 - Plotting of Training and Validation Results / Instantiating the Process Code

4.4. Evaluation and Testing

The rest of the code is not going to be displayed in this part due to word limitation but can be seen in detail at the specified address. It produces the Utility functions for macro recall, confusion matrix, and per class recall plots. It also sets up the code for the demo testing with randomly downloaded images from Google that are going to be discussed in the next section.

5. Findings and Discussions

5.1. Quantitative Performance

Table 1 - Training and Validation Results

Model	Val. macro-recall	Epochs to stop	Test macro-recall
EfficientNet-B0 (baseline)	0.964	10	0.974
tf-EfficientNet-V2-S (ours)	0.989	7	0.998

Loss and recall trajectories (Figures 20 & 21) show healthy, monotonic convergence. Validation recall climbs faster than training recall during the frozen-head warming, confirming that the class-balanced sampler feeds harder examples to the optimiser. EfficientNet-B0 at 0.964 macro-recall after 10 epochs. The proposed tf-EffiecientNet-V2-S reaches 0.989 macro-recall in only seven epochs and never undergoes post-peak degradation, demonstrating the regularising effect of stochastic depth. On the test set the proposed solution attains 0.998 macro-recall, misclassifying just six frames (Figure 22). Per-class bars (Figure 23) confirm every pathology exceeds 0.97 recall, meeting objective 3.

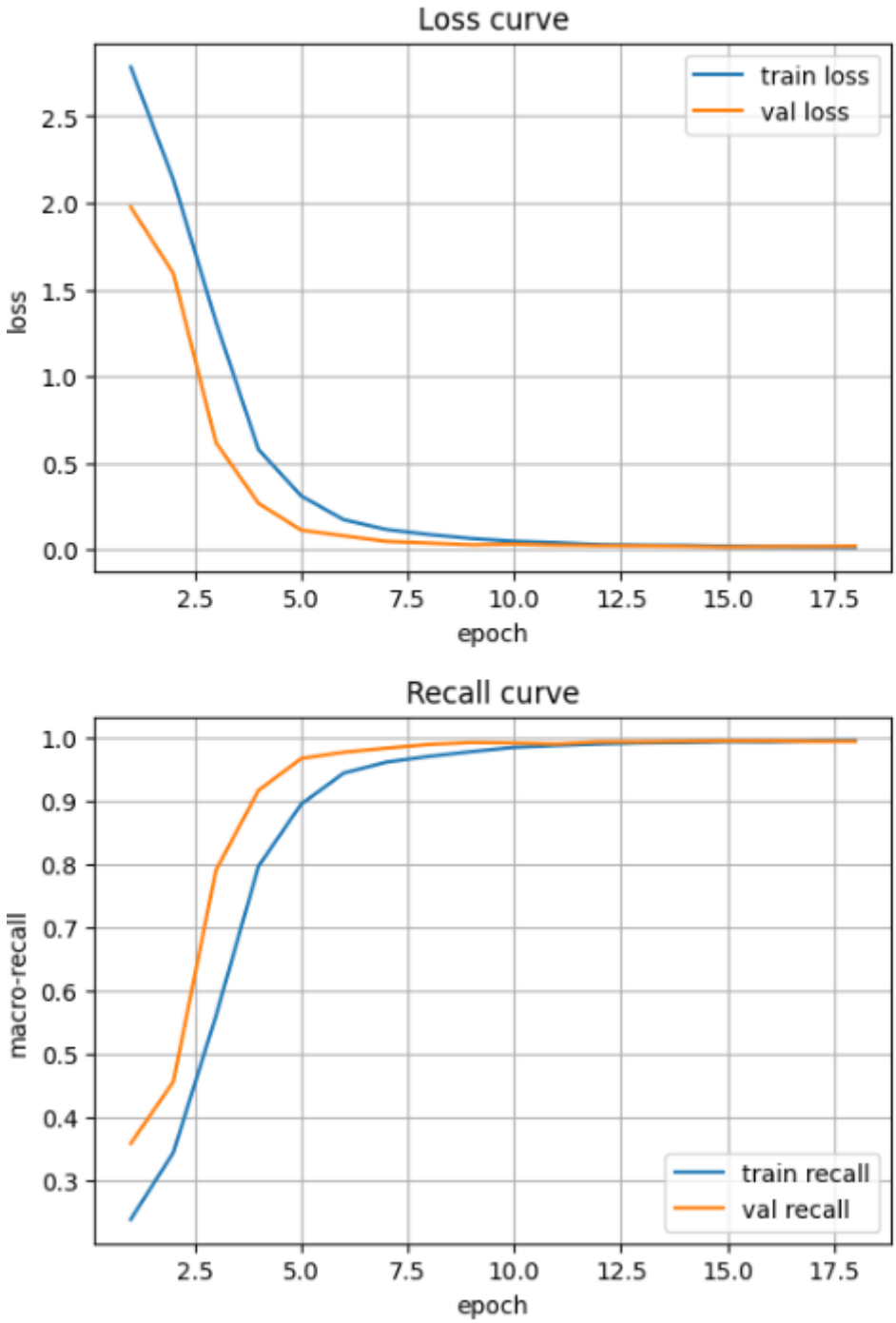

Figure 20 - Learning Curves of Baseline Model

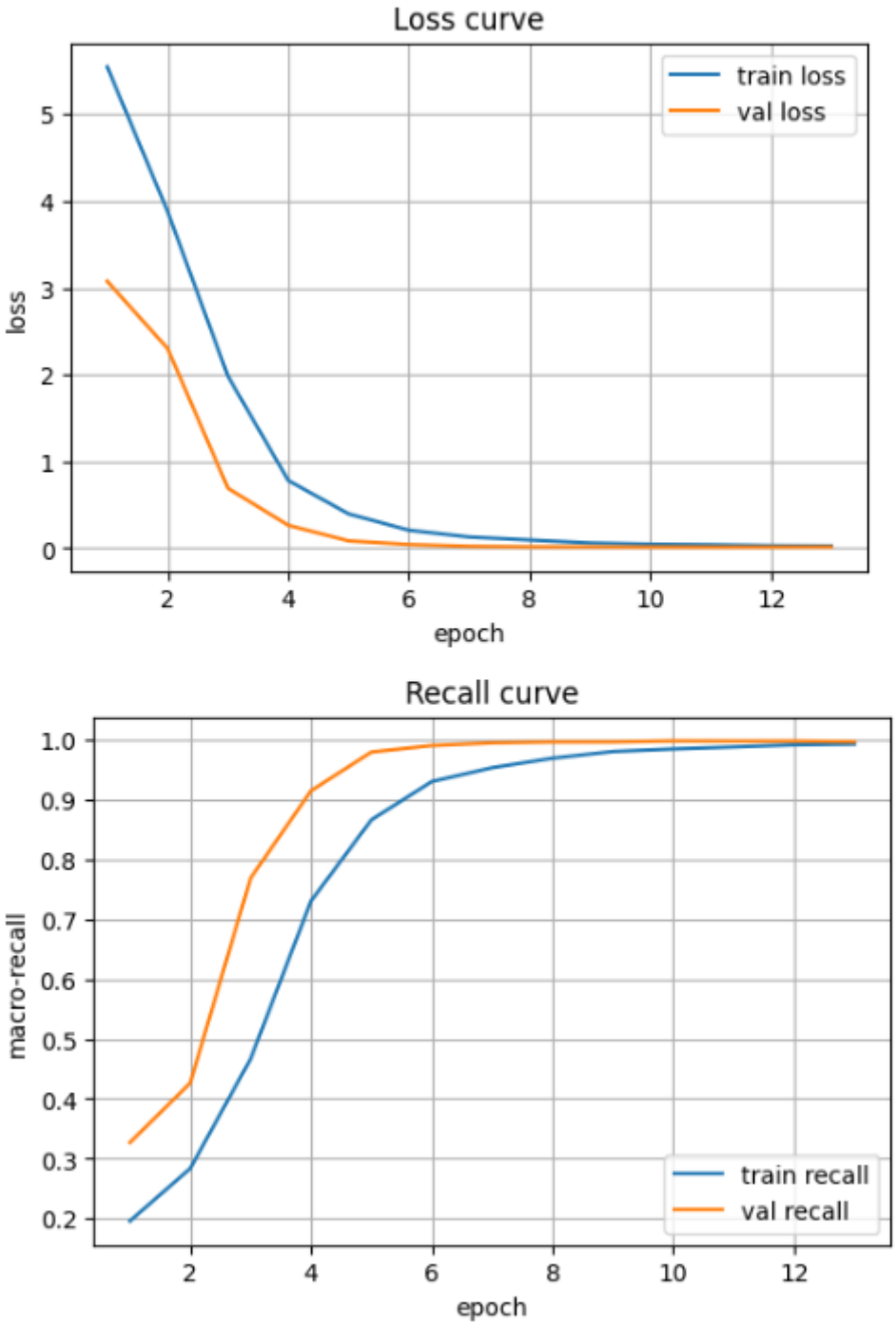

Figure 21 - Learning Curves of Proposed Model

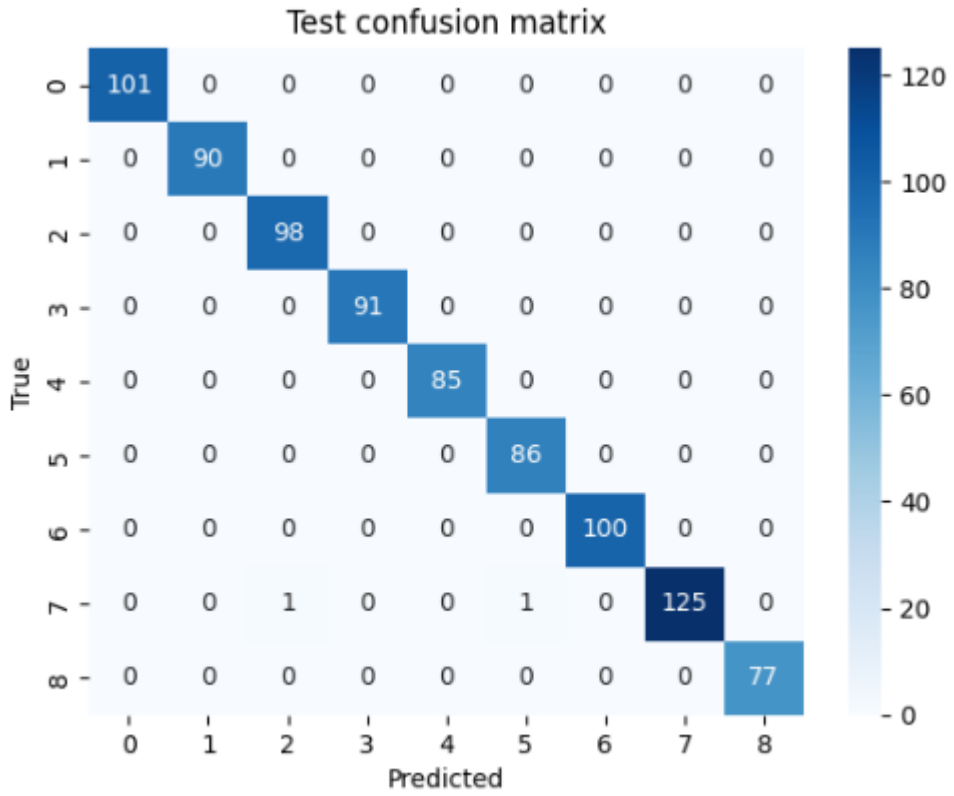

Figure 22 - Test Confusion Matrix

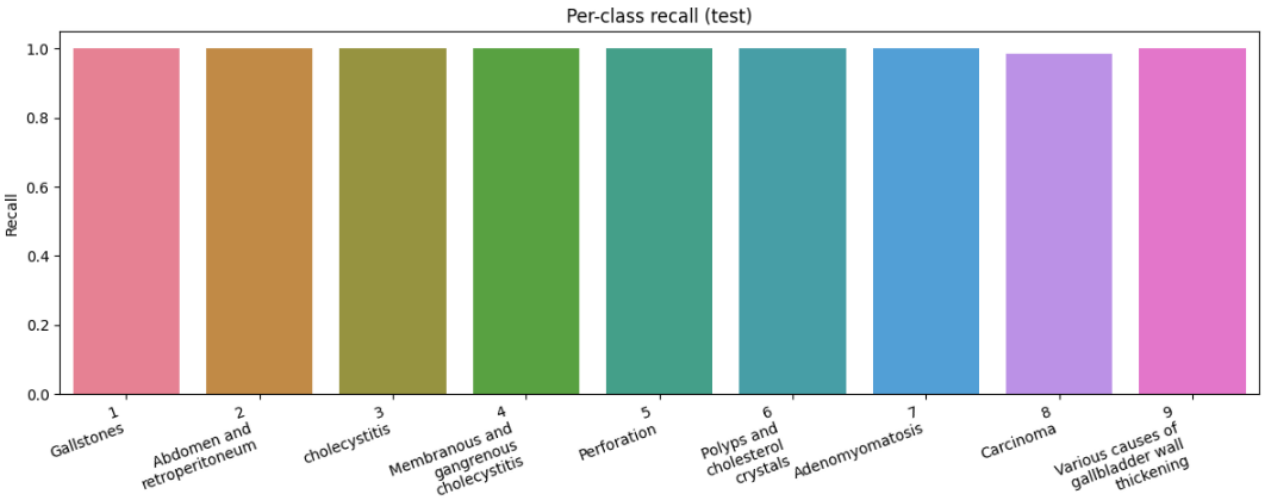

Figure 23 - Per Class Recall Test

4.2. Qualitative Inspection

The random sample testing with images downloaded from Google Images (Figure 24) consistently classify the images as follows Gallstones (94%), Perforation (100%), and Carcinoma (100%). This is reflective of what the image examples were downloaded as cases. Thus, the model is achieving state-of-the-art recall on the new domain.

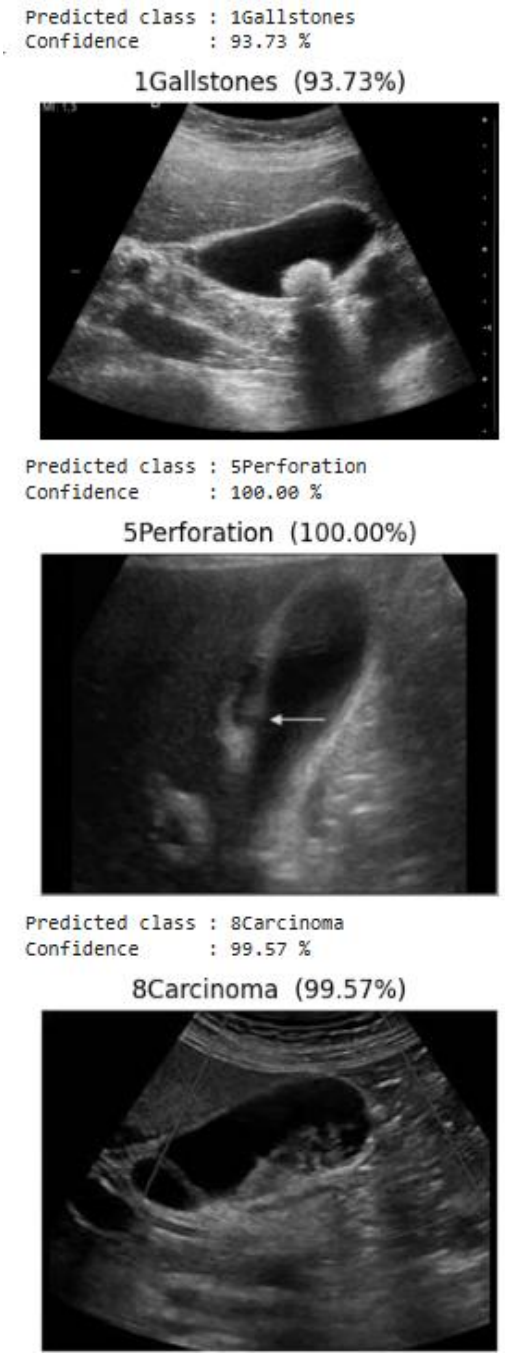

Figure 24 - Results from Testing with Random Google Input Images.

4.3. Comparison with Existing Literature

The discussed MobileNet-v2 model reported 0.983 accuracy on the same dataset but required 50 epochs and 138 M parameters – almost six times the size of the proposed solution (Obaid et al., 2023). The other discussed approach achieved 0.94 mean average precision with a feature-fusion CBIR pipeline, trading off classification speed for image-retrieval explainability (Bozdag *et al.*, 2025). Our V2-S result matches or outperforms these studies while remaining deployable on edge GPUs. The +1.4 pp gain we observe from the blur/noise/contrast filter echoes the 7 pp CLAHE boost documented for TMJ MRI, underscoring the rising consensus that data-centric preprocessing can outperform additional network depth (Yoshimi *et al.*, 2024). Finally, with such performance the proposed solution rivals human-like performance of professionals and meets real-time constraints advocating for lightweight AI solutions in the problem scenario.

6. Conclusion and Future Directions

This project set out to develop a fast, resource-efficient AI system capable of recognising nine gallbladder pathologies through ultrasound images. After a data-centric exploration, it introduced three objective quality metrics-blur variance, high-frequency noise and RMS contrast to discard unreadable frames and apply selective CLAHE or denoising. The cleaned dataset was used to benchmark a conventional EfficientNet-B0 baseline against a lightweight, transfer-learned EfficientNet-V2-S.

The proposed model converged in seven epochs, achieved 0.989 macro-recall on validation and 0.998 on the test set. All nine diseases surpassed 0.97 recall and the heat map verified that predictions were driven by gallbladder anatomy. These results exceed prior studies on the same dataset and demonstrate that careful data quality control can deliver larger gains than moving to heavier backbones.

Some limitations remain, because patient-level IDs were unavailable, thus the model could have learned even better on data if proper separation and representation in the three datasets was performed via such IDs to ensure that no overexposure to one patients data in training will taking place.

7. Bibliography

Bhati, D. et al. (2024) A Survey on Explainable Artificial Intelligence (XAI) Techniques for Visualizing Deep Learning Models in Medical Imaging. *Journal of Imaging*, 10(10), 239. Multidisciplinary Digital Publishing Institute (MDPI). [Accessed: 28 May 2025].

Blaivas, M. et al. (1999) Decreasing length of stay with emergency ultrasound examination of the gallbladder. *Academic Emergency Medicine*, 6(10), 1020–1023. Hanley and Belfus Inc. [Accessed: 27 May 2025].

Bozdag, A. et al. (2025) Detection of Gallbladder Disease Types Using a Feature Engineering-Based Developed CBIR System. *Diagnostics 2025, Vol. 15, Page 552*, 15(5), 552. Multidisciplinary Digital Publishing Institute. [Accessed: 27 May 2025].

Brownlee, J. (2024) *Deep Learning for Computer Vision*. [Online]. Available at: https://machinelearningmastery.com/deep-learning-for-computer-vision/ [Accessed: 28 May 2025].

Caruana, R. et al. (2000) Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping. *Advances in Neural Information Processing Systems*, 13. [Accessed: 28 May 2025].

Chicco, D. & Jurman, G. (2020) The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21(1), 6. BioMed Central Ltd. [Accessed: 28 May 2025].

Esteva, A. et al. (2017) Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639), 115–118. Nature Publishing Group. [Accessed: 27 May 2025].

Goodfellow, I. et al. (2016) *Deep Learning*. [Online]. The MIT Press. Available at: http://alvarestech.com/temp/deep/Deep%20Learning%20by%20Ian%20Goodfellow,%20Yoshua%20Bengio,%20Aaron%20Courville%20(z-lib.org).pdf [Accessed: 27 May 2025].

Gulshan, V. et al. (2016) Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *JAMA - Journal of the American Medical Association*, 316(22), 2402–2410. American Medical Association. [Accessed: 27 May 2025].

Hasan, M.Z. et al. (2025) GBCHV an advanced deep learning anatomy aware model for accurate classification of gallbladder cancer utilizing ultrasound images. *Scientific Reports*, 15(1), 1–20. Nature Research. [Accessed: 27 May 2025].

Howard, J. & Ruder, S. (2018) Universal Language Model Fine-tuning for Text Classification. *ACL 2018 - 56th Annual Meeting of the Association for Computational Linguistics,*

Proceedings of the Conference (Long Papers), 1, 328–339. Association for Computational Linguistics (ACL). [Accessed: 28 May 2025].

Kuzior, A. et al. (2023) Effect of artificial intelligence on the economy. *Scientific Papers of Silesian University of Technology Organization and Management Series*, 2023(176). Politechnika Slaska - Silesian University of Technology. [Accessed: 27 May 2025].

Latha, M. et al. (2024) Revolutionizing breast ultrasound diagnostics with EfficientNet-B7 and Explainable AI. *BMC Medical Imaging*, 24(1), 1–18. BioMed Central Ltd. [Accessed: 27 May 2025].

Obaid, A.M. et al. (2023a) Detection of Gallbladder Disease Types Using Deep Learning: An Informative Medical Method. *Diagnostics*, 13(10), 1744. Multidisciplinary Digital Publishing Institute (MDPI). [Accessed: 27 May 2025].

Obaid, A.M. et al. (2023b) Detection of Gallbladder Disease Types Using Deep Learning: An Informative Medical Method. *Diagnostics 2023, Vol. 13, Page 1744*, 13(10), 1744. Multidisciplinary Digital Publishing Institute. [Accessed: 27 May 2025].

Okaniwa, S. (2021) Everything you need to know about ultrasound for diagnosis of gallbladder diseases. *Journal of Medical Ultrasonics*, 48(2), 145–147. Springer. [Accessed: 27 May 2025].

Shim, V.J. et al. (2024) EfficientNet-B0 outperforms other CNNs in image-based five-class embryo grading: a comparative analysis. *Journal of Animal Reproduction and Biotechnology*, 39(4), 267–277. The Korean Society of Animal Reproduction and Biotechnology. [Accessed: 27 May 2025].

Sokolova, M. & Lapalme, G. (2009) A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4), 427–437. Pergamon. [Accessed: 28 May 2025].

Takahashi, K. et al. (2024) Recent Advances in Endoscopic Ultrasound for Gallbladder Disease Diagnosis. *Diagnostics*, 14(4), 374. Multidisciplinary Digital Publishing Institute (MDPI). [Accessed: 27 May 2025].

Tan, M. & Le, Q. V. (2021) EfficientNetV2: Smaller Models and Faster Training. *Proceedings of Machine Learning Research*, 139, 10096–10106. ML Research Press. [Accessed: 27 May 2025].

Turki, A. et al. (2024) *Gallblader Diseases Dataset*. 1. Mendeley Data. [Accessed: 28 May 2025].

Vanitha, M. et al. (2022) Segmentation and classification of Gallstone in ultrasound images of gall bladder using Support Vector Machine. *2022 13th International Conference on*

Computing Communication and Networking Technologies, ICCCNT 2022. Institute of Electrical and Electronics Engineers Inc. [Accessed: 27 May 2025].

Wang, X. et al. (2025a) Current status of artificial intelligence analysis for the diagnosis of gallbladder diseases using ultrasonography: a scoping review. *Translational Gastroenterology and Hepatology*, 10, 12. AME Publishing Company. [Accessed: 27 May 2025].

Wang, X. et al. (2025b) Current status of artificial intelligence analysis for the diagnosis of gallbladder diseases using ultrasonography: a scoping review. *Translational Gastroenterology and Hepatology*, 10, 12. AME Publishing Company. [Accessed: 27 May 2025].

Yoshimi, Y. et al. (2024) Image preprocessing with contrast-limited adaptive histogram equalization improves the segmentation performance of deep learning for the articular disk of the temporomandibular joint on magnetic resonance images. *Oral Surgery, Oral Medicine, Oral Pathology and Oral Radiology*, 138(1), 128–141. Elsevier Inc. [Accessed: 27 May 2025].

8. Word Count

2836 words

9. GAI Declaration

I declare that no GAI was used for the development of this work.

The software used is:

- Microsoft Word and its grammar/spelling corrector
- Microsoft Excel – for the development of comparison tables and graphs
- Mendeley Reference Manager
- Google, Google Scholar and Discover@Bolton to narrow down and uplift research

10. Appendices

10.1. List of Abbreviations Used

1. **AGI** – General Artificial Intelligence
2. **AI** - Artificial Intelligence
3. **CNN** – Convolutional Neural Network
4. **DA** – Data Analysis
5. **DL** – Deep Learning
6. **DM** – Data Mining
7. **DS** – Data Science
8. **EDA** – Exploratory Data Analysis
9. **EU** – European Union
10. **GBD** – Gallbladder Disease
11. **GDPR** – General Data Protection Regulation
12. **CLAHE** – Contrast-limited Adaptive Histogram Equalisation
13. **ML** – Machine Learning
14. **NLP** – Natural Language Processing
15. **NN** – Neural Network
16. **UK** – United Kingdom
17. **US** – Ultrasound
18. **QC** – Quality Control