CV Track: Convolutional Neural Networks

Vaishnavi
Roll No: 24124048
Department of Mathematics and Computing
IIT BHU

Task 1: CNN Model Design and Training

1 Introduction

This project tackles image classification using the **Caltech-256** dataset. The goal is to build a convolutional neural network (CNN) from scratch that can accurately classify images into their respective classes.

Our approach follows a two-stage pipeline:

- **Model development:** A baseline CNN was implemented and iteratively improved through tuning architectural components such as convolutional layers, activation functions, and regularization.

2 Dataset Description

The experiments in this project were conducted using the **Caltech-256** dataset. It contains **30,607 images** across **256 object categories** and one background class. Each class has a minimum of 80 images, with varying object sizes, poses, and backgrounds, making it suitable for real-world classification challenges.

3 Preprocessing Steps

The following preprocessing steps were applied before training:

- **Resizing:** All images were resized to `224` $\times$ `224` pixels to match the input dimensions expected by standard CNN architectures.
- **ToTensor:** Images were converted to PyTorch tensors using `ToTensor()`, scaling pixel values to the [0, 1] range.
- **Normalization:** The dataset was normalized using the **computed** channel-wise mean and standard deviation:

- Mean: `[0.5520, 0.5336, 0.5050]`
- Std: `[0.2353, 0.2345, 0.2372]`

Note: Dataset-specific mean and standard deviation were used for normalization instead of standard values like ImageNet's, ensuring better alignment with the input distribution and more stable training.

Train-Test Split

The dataset was split into **80% training** and **20% test** sets using PyTorch's `random_split()` function.

---

4 Baseline CNN: MyCNN

Architecture Overview

The network comprises 4 convolutional blocks for feature extraction followed by fully connected layers for classification.

- **Input size:** $3 \times 224 \times 224$ RGB image
- **Block 1:** Conv2d(3, 32) → BatchNorm → ReLU → MaxPool(2x2)
- **Block 2:** Conv2d(32, 64) → BatchNorm → ReLU → MaxPool(2x2)
- **Block 3:** Conv2d(64, 128) → BatchNorm → ReLU → MaxPool(2x2)
- **Block 4:** Conv2d(128, 256) → BatchNorm → ReLU → AdaptiveAvgPool(4x4)
- **Classifier:** Flatten → Linear(4096, 512) → ReLU → Dropout(0.5) → Linear(512, 257)

Training Setup

- **Loss function:** CrossEntropyLoss (multi-class classification)
- **Optimizer:** Adam with initial learning rate of 0.001
- **Scheduler:** StepLR with step size = 10, $\gamma = 0.1$
- **Batch size:** 64
- **Epochs:** 20
- **Split:** 80% train, 20% test
- **Best model saved** based on validation accuracy

Training Progress and Results

- Training and validation loss decreased steadily over epochs.
- Accuracy improved consistently, with the model achieving:
  - **Final Training Accuracy:** 24.29%
  - **Final Validation Accuracy:** 30.53%
- Gradual improvement showed the model successfully learned features despite being trained from scratch on a large dataset.

Performance Visualization

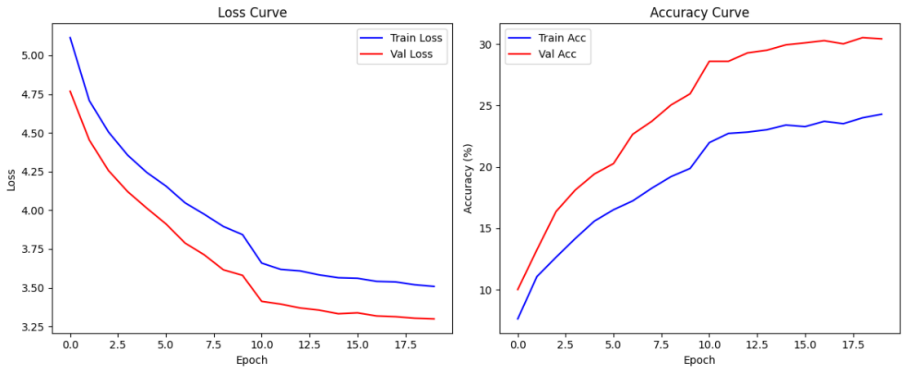

Figure 1: Training vs Validation Loss (left), Accuracy (right) for MyCNN

MyCNN with SGD

- **Motivation:** After training the baseline model using the Adam optimizer, I switched to **Stochastic Gradient Descent (SGD)** to explore its potential for better generalization. Unlike Adam, which adapts learning rates, SGD with momentum often converges more smoothly and encourages flatter minima.
- **Training Configuration:**
  - Optimizer: SGD with learning rate 0.01, momentum 0.9, and weight decay $1e^{-4}$
  - Learning Rate Scheduler: StepLR with step size 10 and decay factor 0.1
  - Epochs: 20 Batch size: 64
  - Loss Function: CrossEntropyLoss

- **Performance:**
  - **Final Training Accuracy:** 47.92%
  - **Final Validation Accuracy: 43.25%**
  - Training and validation accuracy improved steadily with SGD, especially after learning rate decay at epoch 10.
  - Compared to Adam, SGD achieved **13% higher** validation accuracy within the same number of epochs, showing better generalization.

Performance Visualization

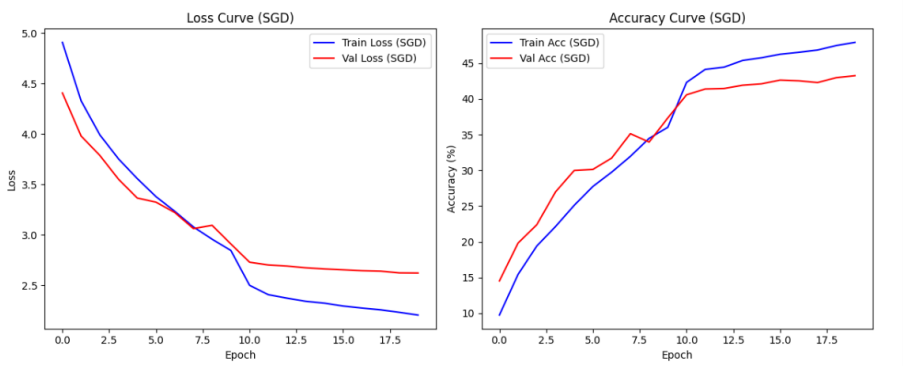

Figure 2: Training vs Validation Loss (left), Accuracy (right) for MyCNN with SGD

ImprovedCNN: Deeper Architecture

- **Architecture Enhancements:**
  - Increased depth with 4 convolutional blocks and two Conv layers per block.
  - Used **Batch Normalization** after each convolution to stabilize learning and speed up convergence.
  - Incorporated **Dropout layers** in the classifier to reduce overfitting.
  - Applied **Adaptive Average Pooling** to reduce spatial dimensions before the fully connected layers.
- **Rationale:**
  - Deeper networks can capture more abstract hierarchical features.
  - BatchNorm helps in reducing internal covariate shift and speeds up training.

- Dropout helps regularize the model and avoid overfitting on training data.

- **Training Configuration:**
  - Optimizer: SGD with learning rate 0.01, momentum 0.9, weight decay $1e^{-4}$
  - Learning Rate Scheduler: StepLR (step size 10, gamma 0.1)
  - Loss Function: CrossEntropyLoss
  - Epochs: 30 Batch size: 64

- **Performance:**
  - Final Training Accuracy: **54.83%**
  - Final Validation Accuracy: **46.10%**
  - Training loss continuously decreased and validation accuracy improved steadily, showing strong convergence.
  - Outperformed both baseline (Adam) and Variant 2 (SGD shallow) models.

Performance Visualization

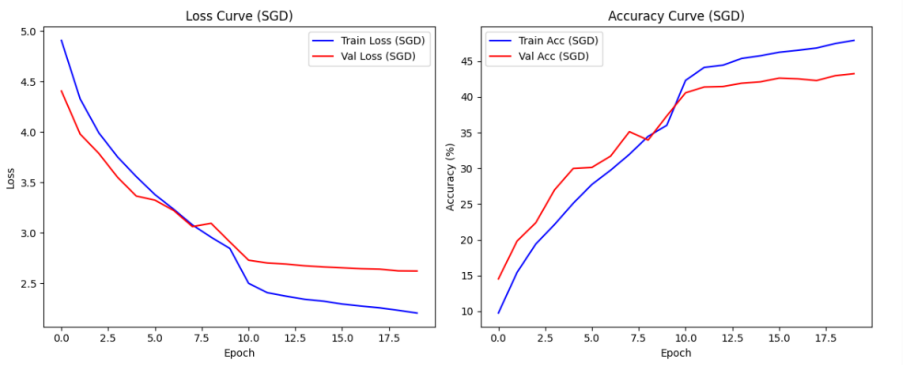

Figure 3: Training vs Validation Loss, Accuracy for ImprovedCNN with SDG

ImprovedCNN with OneCycleLR Scheduler

- **Motivation:**
  - **OneCycleLR** is a dynamic learning rate scheduling technique that increases the learning rate initially and then decreases it smoothly following a cosine annealing pattern.

- It encourages faster convergence in the early epochs and avoids sharp minima by annealing the learning rate down gradually.
- This method is especially effective for deep architectures and SGD-based training, helping models to generalize better.

- **Training Configuration:**
  - Model: **MyImprovedCNN** (deep CNN with BatchNorm, Dropout)
  - Optimizer: SGD with initial learning rate 0.01, momentum 0.9, weight decay $1e^{-4}$
  - Scheduler: OneCycleLR with peak LR 0.1, `cosine` annealing, 20 epochs, `pct_start` = 0.3
  - Loss Function: CrossEntropyLoss
  - Epochs: 20 Batch size: 64

- **Results:**
  - **Final Training Accuracy: 70.36%**
  - **Best Validation Accuracy: 48.35%**
  - Dynamic learning rate helped achieve faster convergence with better accuracy in fewer epochs than traditional schedulers like StepLR.
  - ImprovedCNN + OneCycleLR outperformed all previous models, including baseline Adam and fixed-schedule SGD models.

Performance Visualization

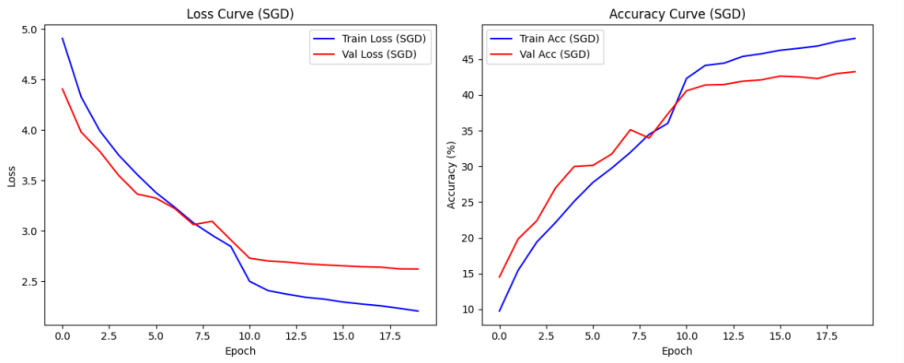

Figure 4: Training vs Validation Loss, Accuracy for ImprovedCNN with OneCycleLR

Conclusion of Task 1

Through the iterative development of convolutional neural network (CNN) architectures and training strategies, several key insights emerged:

- **Optimizer Choice Matters:** While Adam led to quick convergence, it plateaued early. SGD, though slower initially, ultimately generalized better.
- **Scheduling Enhances Performance:** Using a fixed StepLR improved results over no scheduler, but the **OneCycleLR** dynamic scheduler proved most effective by boosting early learning and then annealing to refinement.
- **Architecture is Crucial:** The transition from a shallow CNN to a deeper model (ImprovedCNN) with Batch Normalization and Dropout layers significantly increased validation accuracy and stability.
- **Best Performing Model:** The **ImprovedCNN with OneCycleLR** achieved the highest validation accuracy of **48.35%**, showcasing the synergy of depth, regularization, and dynamic learning rates.
- **Overall Learning:** The task reinforced how model architecture, training strategies, and hyperparameter tuning must be co-optimized for the best real-world results.

This best-performing model ImprovedCNN with OneCy- cleLR was saved and will be used in **Task 2**.

Task 2: Class Activation Mapping (CAM) and Interpretability

1. Objective

2. Model interpretability: The goal of Task 2 is to use **Grad-CAM** to visualize which regions of an input image most influence the CNN's predictions. This helps interpret the model's focus and validate that it learns meaningful features.

CAM Technique Chosen

Method Used: Grad-CAM (Gradient-weighted Class Activation Mapping)

Grad-CAM was selected for its compatibility with any CNN-based architecture and its effectiveness in localizing class-discriminative regions using only the gradients and feature maps from a target convolutional layer.

3. CAM Implementation

- **Target Layer:** `model.features[24]` (last convolutional layer).
- **Why This Layer:** Provides spatial information critical for localization before flattening.
- **Model Used:** The best-performing ImprovedCNN with OneCycleLR from Task 1.

Hooks were registered to capture:

- **Forward Pass:** Stores the activation maps.
- **Backward Pass:** Stores the gradients with respect to the selected layer.

4. Grad-CAM: Hooks and Internals

The core idea of Grad-CAM is to weight the feature maps by the average of their gradients for a specific class. This highlights regions contributing most to the prediction.

5. Grad-CAM Generation and Visualization

For each sampled image:

- Forward pass determined the predicted class.

- Backward pass computed gradients with respect to the predicted class score.
- Grad-CAM heatmaps were generated and overlayed on the original image.
- Three visuals were displayed:
  - Original input image.
  - Grad-CAM heatmap.
  - Overlay of heat

The implementation involved several helper functions:

- **Denormalization:** Reverses dataset normalization using Caltech-256 statistics to display original images.
- **Heatmap Generation:**
  - Gradients are averaged over spatial dimensions to compute importance weights.
  - Weights are multiplied with activation maps and averaged channel-wise.
  - ReLU and normalization are applied to produce the final heatmap.
- **Visualization:**
  - The heatmap is resized and converted to a JET colormap using OpenCV.
  - It is blended with the original image to highlight salient regions.

6. Image Sampling Strategy

- Randomly selected 5 classes from the dataset.
- Sampled 2 images per class using a shuffled DataLoader.
- Images were stored class-wise for interpretability inspection.

7. Generated Heatmaps

Note: Images were randomly sampled and may come from either the training or test set.

Grad-CAM Results: Class 177 (School Bus)

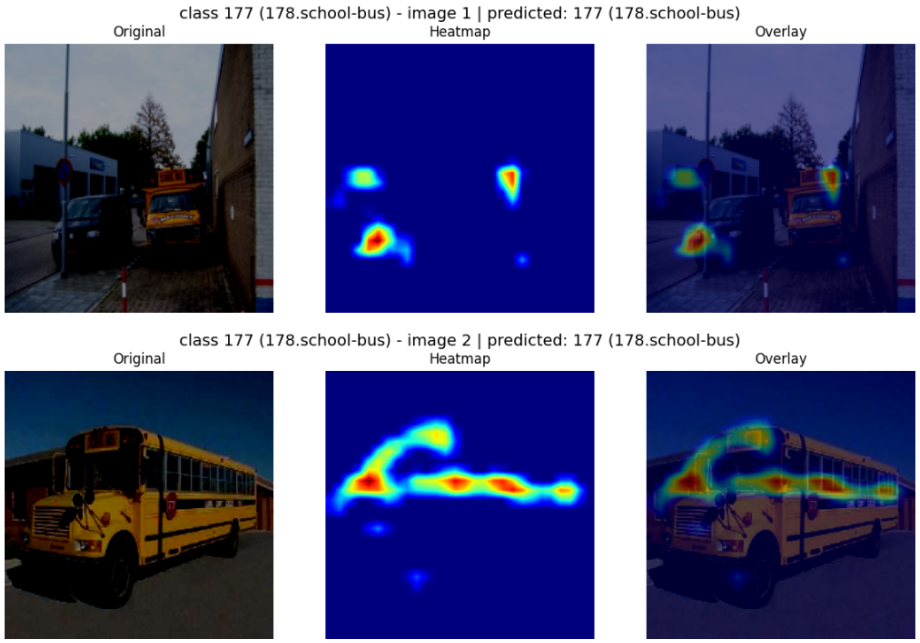

Image 1 (Top Row)

- Correctly classified as school-bus.
- The heatmap activates over parts of the bus but also highlights irrelevant areas like surrounding cars and pavement.
- Suggests the model uses partial object cues and nearby context.

Image 2 (Bottom Row)

- Also correctly classified.
- Heatmap strongly activates across the full bus structure (roof, wheels, front).
- Indicates effective object localization and high model confidence.

Grad-CAM Results: Class 159 (Pez Dispenser)

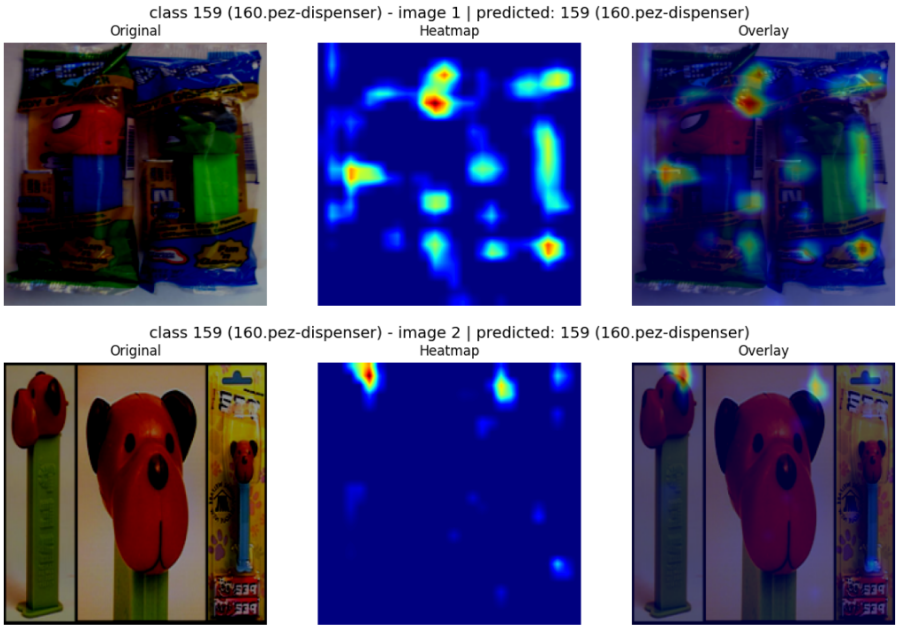

Image 1 (Top Row)

- Correctly classified as pez-dispenser.
- The heatmap highlights both the dispenser heads and some packaging elements.
- Some activation in background suggests the model may rely on contextual packaging patterns.

Image 2 (Bottom Row)

- Also correctly classified.
- Focused activation around the head and stem of the dispenser toy.
- Good object localization with minimal distraction from background.

Grad-CAM Results: Class 132 (Lightning)

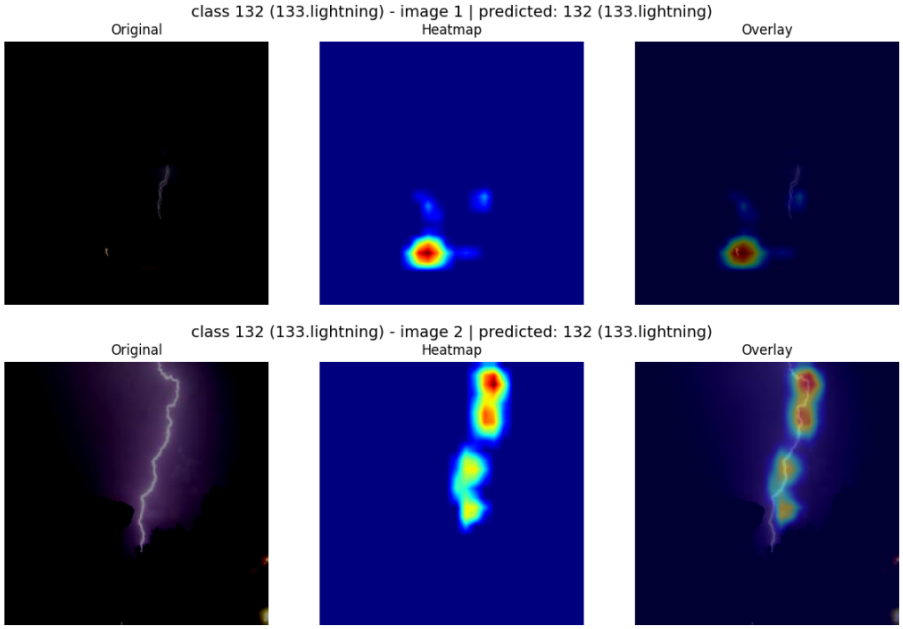

Image 1 (Top Row)

- Correctly classified as lightning.
- The heatmap activates on a bright region below the lightning bolt.
- Model appears to rely on indirect cues, possibly due to low visibility of the bolt.

Image 2 (Bottom Row)

- Also correctly classified.
- The heatmap follows the lightning bolt structure closely.
- Indicates strong focus on the actual object and effective visual grounding.

Grad-CAM Results: Class 1 (American Flag)

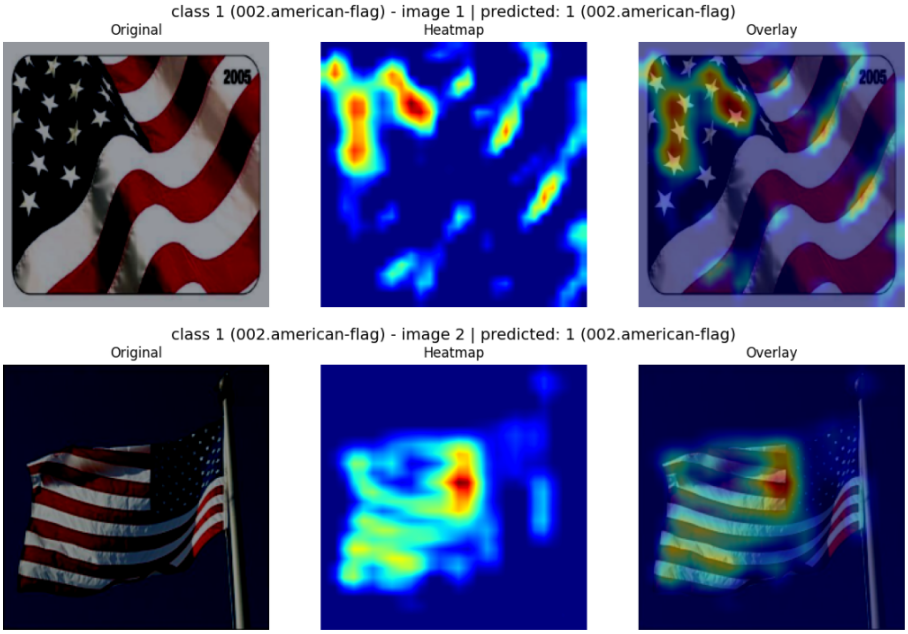

Image 1 (Top Row)

- Correctly classified as american-flag.
- Heatmap emphasizes the top-left portion with stars and stripes.
- Highlights model's focus on distinct, symbolic features of the flag.

Image 2 (Bottom Row)

- Correct classification again.
- Heatmap covers the central and waving striped region of the flag.
- Confirms that the model uses prominent color and pattern cues.

Grad-CAM Results: Class 76 (French Horn)

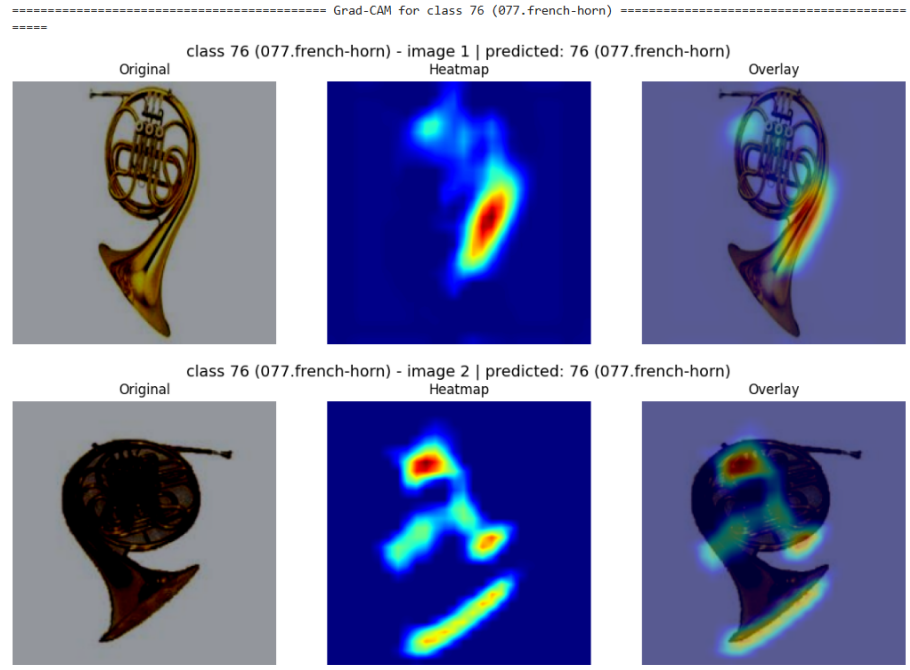

Image 1 (Top Row)

- Correctly classified as french-horn.
- Heatmap is focused over the horn's main coiled structure and bell.
- Highlights that the model correctly attends to distinctive parts of the instrument.

Image 2 (Bottom Row)

- Also correctly predicted.
- Activation covers key circular features and mouthpiece area.
- Shows model robustness to orientation and lighting variations.

---

8. Conclusion: Task 2–Grad-CAM and Interpretability

- Grad-CAM was applied on the best-performing model (ImprovedCNN with OneCycleLR), which achieved a validation accuracy of **48.35%**.
- Visualizations showed that the model often highlighted relevant object regions (e.g., bus, flag, lightning), indicating correct reasoning.
- In some images, it also activated on background or packaging (e.g., pez-dispenser), suggesting partial reliance on context.
- Strong activation on full object structure in many cases reflected good feature learning despite moderate accuracy.
- These results confirm that the model predictions were generally based on meaningful visual cues, improving interpretability.

---

9. Suggestion for better Task- 2 implementation

To improve the reliability of interpretability analysis, a fixed random seed (e.g., 42) can be used during dataset splitting. This ensures consistent train-test partitions across experiments.

Grad-CAM can then be selectively applied on the test set to better evaluate how the model localizes features on unseen (untrained) data, providing more meaningful insights into generalization.

End of Report