Convolutional Neural Network and Grad-CAM: Caltech-256 Classification Report

Saksham Madan
IIT BHU, Mathematics & Computing

June 8, 2025

Abstract

This report presents the design, training, and analysis of a Convolutional Neural Network (CNN) from scratch, applied to the Caltech-256 dataset. I further explore model interpretability via Gradient-weighted Class Activation Mapping (Grad-CAM). The report elaborates on the dataset, preprocessing pipeline, VGG-inspired architecture, backpropagation, reasons for low validation accuracy, and mathematical foundations of Grad-CAM. I also reflect on implementation choices and learning outcomes.

1 Caltech-256 Dataset

The Caltech-256 dataset is a benchmark for object recognition in cluttered real-world images. It contains 30,607 images across 257 object categories, including animals, instruments, vehicles, etc.

- **Min images per class**: 80
- **Max images per class**: 827
- **Median**: 100
- **Mean**: 119

Its diversity and clutter make classification a challenging task, suitable for learning robust feature representations.

2 Data Preprocessing and Transformer Pipeline

To prepare images for CNN training, the following preprocessing pipeline was used:

```
transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485,0.456,0.406],
                         std=[0.229,0.224,0.225])
])
```

2.1 Rationale

- **Resize**: Ensures uniform input size of 224×224.
- **ToTensor**: Converts images to PyTorch tensors.
- **Normalize**: Matches ImageNet statistics for stable training.

3 Computer Vision and Convolutions

3.1 Convolutions in CNNs

Convolutions help learn spatially local patterns in images:

$$y_{i,j,k} = \sum_{c=1}^{C} \sum_{u,v} x_{i+u,j+v,c} \cdot w_{u,v,c,k} + b_k \tag{1}$$

Each kernel w learns specific features like edges or textures, while stacking multiple layers allows learning hierarchical representations.

3.2 Pooling and Activation

- **Max Pooling**: Reduces spatial dimensions, retains dominant features.
- **ReLU**: Introduces non-linearity: $\text{ReLU}(z) = \max(0, z)$.

4 CNN Architecture and Model Working

I implemented a **5-block VGG-like CNN** from scratch.

4.1 Architecture Overview

Layer	Output Shape
Input	3×224×224
Block1	64×112×112
Block2	128×56×56
Block3	256×28×28
Block4	512×14×14
Block5	512×7×7
Dense Layers	4096, 1024, 257 outputs

Table 1: VGG-inspired architecture summary.

4.2 Block Details

Each block contains:

- 2-3 Conv layers with 3×3 kernels
- Batch Normalization
- ReLU activation
- Max pooling with stride 2

4.3 Classifier

- AdaptiveAvgPool2D (7×7)
- Flattening
- Dense layers: $4096 \rightarrow 1024 \rightarrow 257$

4.4 Training Setup

- Optimizer: SGD + momentum
- Learning rate scheduler: OneCycleLR
- Loss function: Cross-entropy
- Batch size: 64
- Epochs: 25

5 Backpropagation and Loss

5.1 Cross-Entropy Loss

$$L = -\sum_{i=1}^{N} y_i \log p_i \tag{2}$$

5.2 Gradient Derivation

Gradient through softmax + log-loss:

$$\frac{\partial L}{\partial z_k} = p_k - y_k \tag{3}$$

For a weight w:

$$\frac{\partial L}{\partial w} = (p_k - y_k)x \tag{4}$$

Epoch	Train Loss	Train Acc (%)	Val Acc (%)
1	5.3324	5.17	6.62
5	4.4747	13.32	15.94
10	3.1979	30.00	30.08
15	1.9497	51.69	36.51
20	0.5316	84.77	41.76
25	0.1274	96.55	44.48

Table 2: Training and validation metrics.

5.3 Training Results

6 Low Accuracy: Reasons and Analysis

6.1 Challenges Faced

- **Deep model from scratch** → no pretrained weights.
- **Highly imbalanced dataset** → classes with as few as 80 samples.
- **257-way classification**: Random baseline $\sim 0.4\%$.
- **Overfitting**: Train accuracy > Val accuracy.

6.2 Solutions Applied

- **Learning rate scheduling**: OneCycleLR helped stabilize training.
- **Regularization**: Dropout layers in dense head.
- **Data pipeline**: Normalization and consistent transforms.

7 Grad-CAM: Interpretability

7.1 Why Grad-CAM?

Grad-CAM generalizes CAM to any CNN architecture by using gradient information. Our architecture does not have GAP (Global Average Pooling) + linear head → original CAM would not work.

Grad-CAM provides class-discriminative localization using:

$$\alpha_k^c = \frac{1}{Z} \sum_{i,j} \frac{\partial y^c}{\partial A_{i,j}^k} \tag{5}$$

7.2 Mathematical Formulation

- A^k: Feature map activations
- y^c: Class score
- α_k^c: Importance weight for feature map k

Final Grad-CAM map:

$$L^c_{\text{Grad-CAM}} = \text{ReLU}(\sum_k \alpha^c_k A^k) \tag{6}$$

7.3 Implementation

- Register forward and backward hooks on `features[42]` layer.
- Accumulate activations and gradients.
- Compute α^c_k and weighted combination.
- Overlay heatmap on original image using OpenCV.

7.4 Results

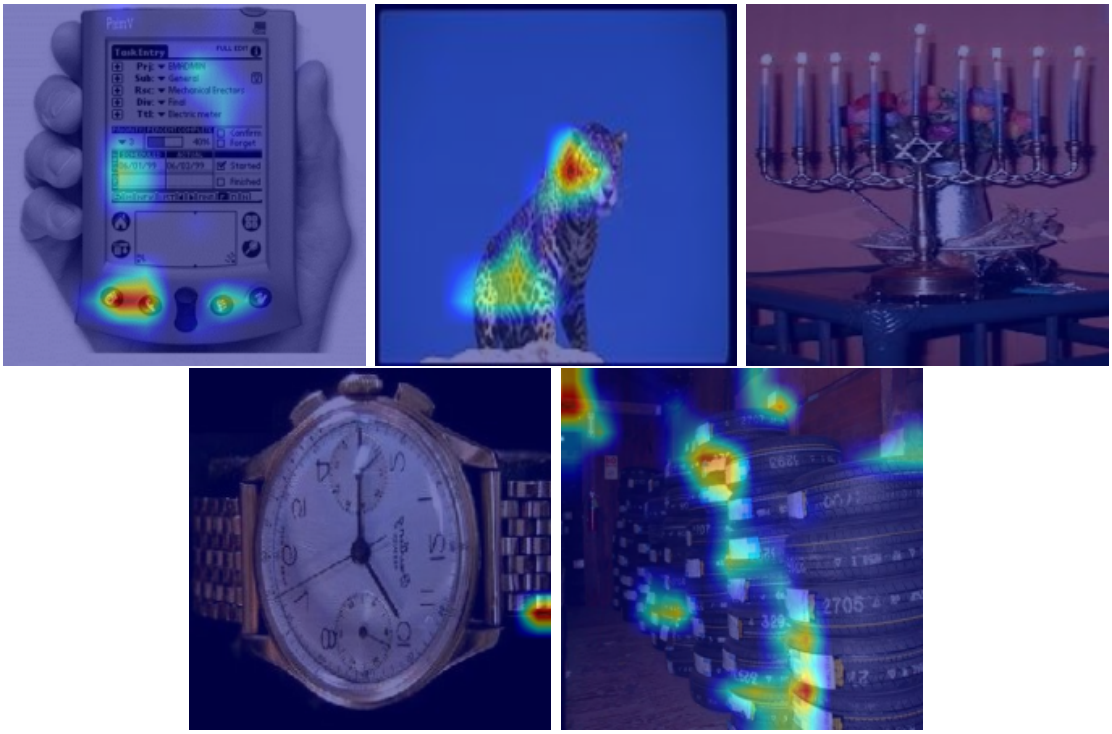

Figure 1: Sample Grad-CAM visualizations.

7.5 Observations

- Even with moderate accuracy, the model focused on correct object regions.
- CAM highlighted semantically meaningful parts.
- Helped diagnose which images failed due to background clutter.

8 Reflection and Improvements

8.1 What I Learned

- Building CNNs from scratch is extremely informative.
- Architecture design and data augmentation matter a lot.
- Grad-CAM helps explain deep models even when accuracy is low.

8.2 Problems Faced

- Initial accuracy stuck $\sim 5\%$: required tuning learning rate scheduler.
- Training instability: fixed with better normalization and OneCycleLR.
- Grad-CAM integration: required careful hook placement and gradient handling.

8.3 Future Improvements

- Add stronger data augmentation: random crop, flips, color jitter.
- Explore ResNet-based architectures.
- Use label smoothing to handle noisy labels.
- Experiment with progressive resizing.

9 Conclusion

I successfully built and trained a 5-block VGG-style CNN on Caltech-256 from scratch, achieving $\sim 44\%$ validation accuracy. I used Grad-CAM to interpret model decisions and visualized important attention regions.

Despite challenges like imbalanced data and deep architecture training from scratch, this project helped solidify my understanding of CNNs, backpropagation, and interpretability techniques.