deep-learning-with-pytorch-gradcam

November 8, 2024

1 Visual Insights: Leveraging Grad-CAM for Enhanced Image Classification in Deep Learning

2 Introduction

Convolutional Neural Networks (CNNs) are powerful tools for visual pattern recognition, widely used in tasks such as object detection and image classification. However, understanding the reasoning behind CNNs' predictions is often challenging due to their complex structures. Gradient-weighted Class Activation Mapping (Grad-CAM) is a visualization technique that addresses this issue by highlighting regions in an input image that are important for the network's decision-making process. Grad-CAM uses the gradient information flowing into the final convolutional layer of a CNN to create a localization map for highlighting discriminative image regions. This project focuses on developing a CNN model for classifying different types of vegetables (cucumber, eggplant, mushroom) and applies Grad-CAM to visualize the regions of interest that influence the model's decisions. By integrating Grad-CAM, the project enhances model interpretability, enabling users to see which parts of the image are most relevant to the network's predictions. This increased transparency is particularly beneficial in applications requiring high reliability, as it allows for visual validation of the model's decisions.

[1]:
```
!git clone https://github.com/parth1620/GradCAM-Dataset.git
!pip install -U git+https://github.com/albumentations-team/albumentations
!pip install --upgrade opencv-contrib-python
```

```
Cloning into 'GradCAM-Dataset'…
remote: Enumerating objects: 193, done.
remote: Counting objects: 100% (193/193), done.
remote: Compressing objects: 100% (193/193), done.
remote: Total 193 (delta 0), reused 193 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (193/193), 2.59 MiB | 36.29 MiB/s, done.
Collecting git+https://github.com/albumentations-team/albumentations
  Cloning https://github.com/albumentations-team/albumentations to /tmp/pip-req-
build-qbhjfi6b
  Running command git clone --filter=blob:none --quiet
https://github.com/albumentations-team/albumentations /tmp/pip-req-build-
qbhjfi6b
  Resolved https://github.com/albumentations-team/albumentations to commit
3ed63f2513affb2dc53c1cfcc962e08ed86c8c62
  Installing build dependencies … done
  Getting requirements to build wheel … done
  Preparing metadata (pyproject.toml) … done
Requirement already satisfied: numpy>=1.24.4 in /usr/local/lib/python3.10/dist-
packages (from albumentations==1.4.21) (1.26.4)
Requirement already satisfied: scipy>=1.10.0 in /usr/local/lib/python3.10/dist-
packages (from albumentations==1.4.21) (1.13.1)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages
(from albumentations==1.4.21) (6.0.2)
Requirement already satisfied: pydantic>=2.7.0 in
/usr/local/lib/python3.10/dist-packages (from albumentations==1.4.21) (2.9.2)
Collecting albucore==0.0.21 (from albumentations==1.4.21)
  Downloading albucore-0.0.21-py3-none-any.whl.metadata (5.3 kB)
Requirement already satisfied: eval-type-backport in
/usr/local/lib/python3.10/dist-packages (from albumentations==1.4.21) (0.2.0)
Requirement already satisfied: opencv-python-headless>=4.9.0.80 in
/usr/local/lib/python3.10/dist-packages (from albumentations==1.4.21)
(4.10.0.84)
Requirement already satisfied: stringzilla>=3.10.4 in
/usr/local/lib/python3.10/dist-packages (from
albucore==0.0.21->albumentations==1.4.21) (3.10.6)
Collecting simsimd>=5.9.2 (from albucore==0.0.21->albumentations==1.4.21)
  Downloading simsimd-6.0.2-cp310-cp310-manylinux_2_28_x86_64.whl.metadata (57
kB)
                           57.7/57.7 kB
5.0 MB/s eta 0:00:00
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.10/dist-packages (from
pydantic>=2.7.0->albumentations==1.4.21) (0.7.0)
Requirement already satisfied: pydantic-core==2.23.4 in
/usr/local/lib/python3.10/dist-packages (from
pydantic>=2.7.0->albumentations==1.4.21) (2.23.4)
Requirement already satisfied: typing-extensions>=4.6.1 in
/usr/local/lib/python3.10/dist-packages (from
pydantic>=2.7.0->albumentations==1.4.21) (4.12.2)
Downloading albucore-0.0.21-py3-none-any.whl (12 kB)
Downloading simsimd-6.0.2-cp310-cp310-manylinux_2_28_x86_64.whl (605 kB)
                         605.1/605.1 kB
33.0 MB/s eta 0:00:00
Building wheels for collected packages: albumentations
  Building wheel for albumentations (pyproject.toml) … done
  Created wheel for albumentations: filename=albumentations-1.4.21-py3-none-
any.whl size=231738
sha256=b9a24221ee966a7c9afa3253b3d5c3b8148b3c122019f63ac293c1a92fb5f4d2
  Stored in directory: /tmp/pip-ephem-wheel-cache-
_k78bj3m/wheels/51/4d/ab/5aafa8b980086fbc362946de7da4aa3df33aacb3da0da29b93
Successfully built albumentations
Installing collected packages: simsimd, albucore, albumentations
  Attempting uninstall: albucore
```

```
    Found existing installation: albucore 0.0.19
    Uninstalling albucore-0.0.19:
      Successfully uninstalled albucore-0.0.19
  Attempting uninstall: albumentations
    Found existing installation: albumentations 1.4.20
    Uninstalling albumentations-1.4.20:
      Successfully uninstalled albumentations-1.4.20
Successfully installed albucore-0.0.21 albumentations-1.4.21 simsimd-6.0.2
Requirement already satisfied: opencv-contrib-python in
/usr/local/lib/python3.10/dist-packages (4.10.0.84)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-
packages (from opencv-contrib-python) (1.26.4)
```

3 Imports

[2]:
```python
import sys
sys.path.append('/content/GradCAM-Dataset')
```

[3]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import cv2
import torch
from torch import nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset
from torchvision import datasets, transforms as T
from tqdm import tqdm
import albumentations as A
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

import utils
```

4 Configurations

[4]:
```python
CSV_FILE = '/content/GradCAM-Dataset/train.csv'
DATA_DIR = '/content/GradCAM-Dataset/'
DEVICE = 'cuda'
BATCH_SIZE = 16
LR = 0.001
EPOCHS = 20
```

[5]:
```python
data = pd.read_csv(CSV_FILE)
data.head()
```

[5]:
```
                        img_path  label
0  train_images/mushroom_51.jpg      2
1  train_images/eggplant_37.jpg      1
2  train_images/mushroom_20.jpg      2
3  train_images/eggplant_51.jpg      1
4  train_images/eggplant_26.jpg      1
```

[6]:
```
# cucumber 0, eggplant 1, mushoom - 2
train_df, valid_df = train_test_split(data, test_size = 0.2, random_state = 42)
```

5 Augmentations

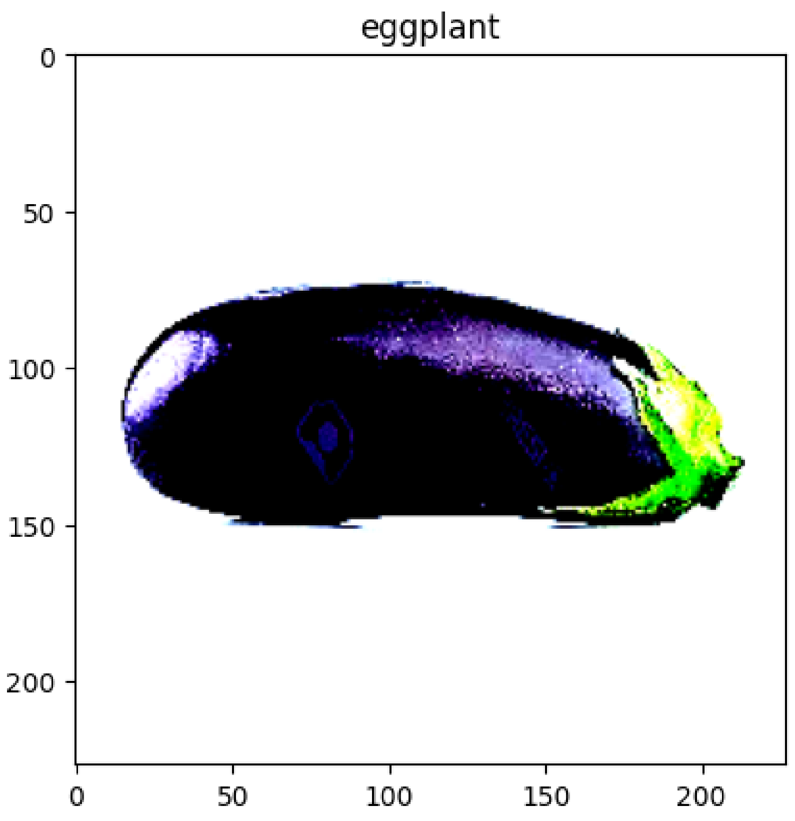
[7]:
```
train_augs = A.Compose ([
  A. Rotate(),
  A.HorizontalFlip(p=0.5),
  A.VerticalFlip(p = 0.5),
  A.Normalize(mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225])
])

valid_augs = A.Compose([A.Normalize(mean = [0.485, 0.456, 0.406], std = [0.229,␣
  ↪0.224, 0.225])])
```

6 Load Image Dataset

[8]:
```
trainset =  utils.ImageDataset(train_df, augs = train_augs, data_dir = DATA_DIR)
validset =  utils.ImageDataset(valid_df, augs = valid_augs, data_dir = DATA_DIR)
```

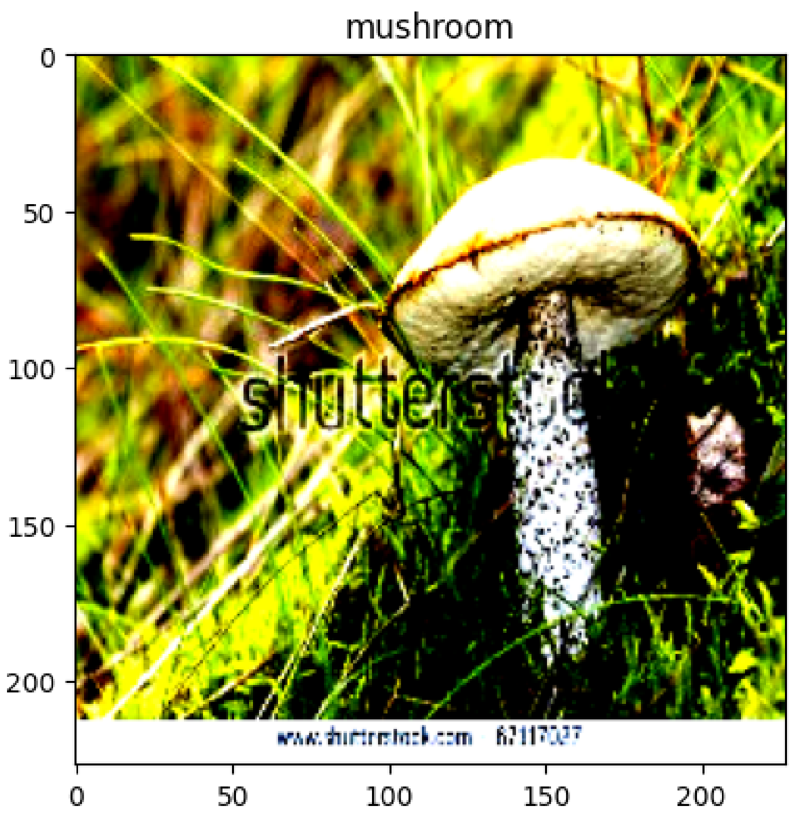
[9]:
```
image, label =  trainset [0] #(c,h,w) -> (h, w, c)
class_list = ['cucumber', 'eggplant', 'mushroom']
plt.imshow(image.permute(1, 2, 0))
plt.title(class_list[label]);
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
```



```
[10]: image, label =  validset [21] #(c,h,w) -> (h, w, c)
      class_list = ['cucumber', 'eggplant', 'mushroom']
      plt.imshow(image.permute(1, 2, 0))
      plt.title(class_list[label]);
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
```



```
[11]: print(f"No. of examples in the trainset {len(trainset)}")
      print(f"No. of examples in the validset {len(validset)}")
```

```
No. of examples in the trainset 148
No. of examples in the validset 38
```

7 Load Dataset into Batches

```
[12]: trainloader = DataLoader (trainset, batch_size = BATCH_SIZE, shuffle = True)
      validloader = DataLoader (validset, batch_size = BATCH_SIZE)
```

```
[13]: print(f"No. of batches in trainloader : {len(trainloader)}")
      print(f"No. of batches in validloader : {len(validloader)}")
```

```
No. of batches in trainloader : 10
No. of batches in validloader : 3
```

```
[14]: for images, labels in trainloader:
        break

      print(f"One batch image shape: {images.shape}")
print(f"One batch label shape: {labels.shape}")
```

```
One batch image shape: torch.Size([16, 3, 227, 227])
One batch label shape: torch.Size([16])
```

8 Create Model

[15]:
```
class ImageModel(nn.Module):

    def __init__(self):
        super(ImageModel, self).__init__()

        # Define the feature extractor
        self.feature_extractor = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=(5, 5),␣
↪padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(4, 4), stride=2),

            nn.Conv2d(in_channels=16, out_channels=16, kernel_size=(5, 5),␣
↪padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(4, 4), stride=2),

            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=(5, 5),␣
↪padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(4, 4), stride=2),

            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=(5, 5),␣
↪padding=1),
            nn.ReLU(),
        )

        self.maxpool = nn.MaxPool2d(kernel_size=(4, 4), stride=2)

        # Define the classifier
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(6400, 2048),
            nn.ReLU(),
            nn.Linear(2048, 3)
        )

        self.gradient = None
    def activations_hook(self, grad):
        # Save gradients
        self.gradient = grad

    def forward(self, images):
        x = self.feature_extractor(images)  # Activation maps
        h = x.register_hook(self.activations_hook)  # Register hook
        x = self.maxpool(x)
        x = self.classifier(x)
        return x

    def get_activation_gradients(self):
        return self.gradient

    def get_activations(self, x):
        return self.feature_extractor(x)
```

[16]:
```
# Instantiate model and move to device
model = ImageModel()
model.to(DEVICE)
```

[16]:
```
ImageModel(
  (feature_extractor): Sequential(
    (0): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(4, 4), stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(16, 16, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=(4, 4), stride=2, padding=0, dilation=1,
ceil_mode=False)
    (6): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=(4, 4), stride=2, padding=0, dilation=1,
ceil_mode=False)
    (9): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
    (10): ReLU()
  )
  (maxpool): MaxPool2d(kernel_size=(4, 4), stride=2, padding=0, dilation=1,
ceil_mode=False)
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=6400, out_features=2048, bias=True)
    (2): ReLU()
    (3): Linear(in_features=2048, out_features=3, bias=True)
  )
)
```

9 Create Train and Eval function

[17]:
```
# Training and evaluation functions
def train_fn(dataloader, model, optimizer, criterion):
    model.train()
    total_loss = 0.0
    for images, labels in tqdm(dataloader):
        images = images.to(DEVICE)
        labels = labels.to(DEVICE)

        optimizer.zero_grad()
        logits = model(images)
        loss = criterion(logits, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    return total_loss / len(dataloader)
```

[18]:
```
def eval_fn(dataloader, model, criterion):
    model.eval()
    total_loss = 0.0
    for images, labels in tqdm(dataloader):
        images = images.to(DEVICE)
        labels = labels.to(DEVICE)

        logits = model(images)
        loss = criterion(logits, labels)
        total_loss += loss.item()
    return total_loss / len(dataloader)
```

10 Training Loop

[19]:
```
# Training loop
optimizer = torch.optim.Adam(model.parameters(), lr=LR)
criterion = torch.nn.CrossEntropyLoss()
```

[20]:
```
best_valid_loss = np.Inf
for i in range(EPOCHS):
    train_loss = train_fn(trainloader, model, optimizer, criterion)
    valid_loss = eval_fn(validloader, model, criterion)

    if valid_loss < best_valid_loss:
        torch.save(model.state_dict(), 'best_weights.pt')
        best_valid_loss = valid_loss
        print("SAVED_WEIGHTS_SUCCESS")
    print(f"EPOCH: {i + 1} TRAIN LOSS: {train_loss} VALID LOSS: {valid_loss}")
```

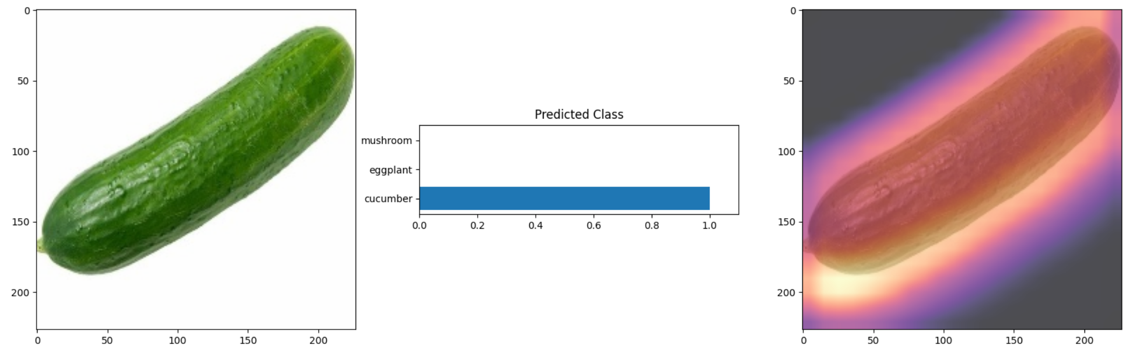
```
100%|      | 10/10 [00:02<00:00,  3.70it/s]
100%|      | 3/3 [00:00<00:00, 17.40it/s]

SAVED_WEIGHTS_SUCCESS
EPOCH: 1 TRAIN LOSS: 1.175425386428833 VALID LOSS: 1.1036356290181477

100%|      | 10/10 [00:00<00:00, 11.48it/s]
100%|      | 3/3 [00:00<00:00, 18.78it/s]

SAVED_WEIGHTS_SUCCESS
EPOCH: 2 TRAIN LOSS: 0.9710135996341706 VALID LOSS: 0.5784633855024973

100%|      | 10/10 [00:00<00:00, 11.88it/s]
100%|      | 3/3 [00:00<00:00, 26.05it/s]

EPOCH: 3 TRAIN LOSS: 0.5275370059534907 VALID LOSS: 0.850549245874087

100%|      | 10/10 [00:00<00:00, 12.00it/s]
100%|      | 3/3 [00:00<00:00, 27.26it/s]

SAVED_WEIGHTS_SUCCESS
EPOCH: 4 TRAIN LOSS: 0.3127263359725475 VALID LOSS: 0.44183332473039627

100%|      | 10/10 [00:00<00:00, 10.43it/s]
100%|      | 3/3 [00:00<00:00, 22.13it/s]

SAVED_WEIGHTS_SUCCESS
EPOCH: 5 TRAIN LOSS: 0.20782727673649787 VALID LOSS: 0.3288388152917226

100%|      | 10/10 [00:00<00:00, 11.08it/s]
100%|      | 3/3 [00:00<00:00, 13.06it/s]

SAVED_WEIGHTS_SUCCESS
EPOCH: 6 TRAIN LOSS: 0.1449680792167783 VALID LOSS: 0.32816399323443574

100%|      | 10/10 [00:01<00:00,  7.99it/s]
100%|      | 3/3 [00:00<00:00, 16.16it/s]

SAVED_WEIGHTS_SUCCESS
EPOCH: 7 TRAIN LOSS: 0.13503627367317678 VALID LOSS: 0.14047523339589438

100%|      | 10/10 [00:01<00:00,  9.31it/s]
100%|      | 3/3 [00:00<00:00, 26.68it/s]

EPOCH: 8 TRAIN LOSS: 0.14470929149538278 VALID LOSS: 0.2751892718176047

100%|      | 10/10 [00:00<00:00, 11.32it/s]
100%|      | 3/3 [00:00<00:00, 18.74it/s]

EPOCH: 9 TRAIN LOSS: 0.08999965526163578 VALID LOSS: 0.17372394415239492

100%|      | 10/10 [00:00<00:00, 10.09it/s]
100%|      | 3/3 [00:00<00:00, 16.79it/s]

EPOCH: 10 TRAIN LOSS: 0.14263124614953995 VALID LOSS: 0.2624124487241109

100%|      | 10/10 [00:00<00:00, 11.87it/s]
100%|      | 3/3 [00:00<00:00, 22.70it/s]
```

```
EPOCH: 11 TRAIN LOSS: 0.21540712639689447 VALID LOSS: 0.14099142203728357

100%|      | 10/10 [00:00<00:00, 13.77it/s]
100%|      | 3/3 [00:00<00:00, 42.85it/s]

EPOCH: 12 TRAIN LOSS: 0.10645484456326812 VALID LOSS: 0.14770341105759144

100%|      | 10/10 [00:00<00:00, 18.18it/s]
100%|      | 3/3 [00:00<00:00, 41.10it/s]

EPOCH: 13 TRAIN LOSS: 0.015948243765160443 VALID LOSS: 0.5392987304367125

100%|      | 10/10 [00:00<00:00, 19.24it/s]
100%|      | 3/3 [00:00<00:00, 42.27it/s]

EPOCH: 14 TRAIN LOSS: 0.020384780218591914 VALID LOSS: 0.3548179840048154

100%|      | 10/10 [00:00<00:00, 19.66it/s]
100%|      | 3/3 [00:00<00:00, 44.46it/s]

EPOCH: 15 TRAIN LOSS: 0.3075347709469497 VALID LOSS: 0.38957588374614716

100%|      | 10/10 [00:00<00:00, 20.18it/s]
100%|      | 3/3 [00:00<00:00, 45.24it/s]

EPOCH: 16 TRAIN LOSS: 0.31681572198867797 VALID LOSS: 0.2203957512974739

100%|      | 10/10 [00:00<00:00, 20.78it/s]
100%|      | 3/3 [00:00<00:00, 45.95it/s]

SAVED_WEIGHTS_SUCCESS
EPOCH: 17 TRAIN LOSS: 0.10499591063708066 VALID LOSS: 0.10954343527555466

100%|      | 10/10 [00:00<00:00, 20.13it/s]
100%|      | 3/3 [00:00<00:00, 42.75it/s]

EPOCH: 18 TRAIN LOSS: 0.043982683098874986 VALID LOSS: 0.18626237908999124

100%|      | 10/10 [00:00<00:00, 20.42it/s]
100%|      | 3/3 [00:00<00:00, 45.44it/s]

SAVED_WEIGHTS_SUCCESS
EPOCH: 19 TRAIN LOSS: 0.013109330995939672 VALID LOSS: 0.09453159112793703

100%|      | 10/10 [00:00<00:00, 21.00it/s]
100%|      | 3/3 [00:00<00:00, 43.97it/s]

SAVED_WEIGHTS_SUCCESS
EPOCH: 20 TRAIN LOSS: 0.001572768311780237 VALID LOSS: 0.09233398083597422
```

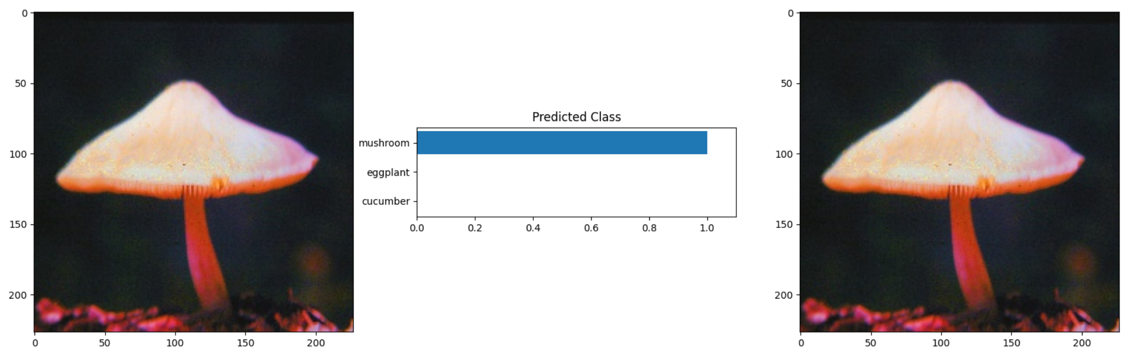
11 Get GradCAM

[21]:
```
# Grad-CAM function
def get_gradcam(model, image, label, size):
    label.backward()
    gradients = model.get_activation_gradients()
    pooled_gradients = torch.mean(gradients, dim=[0, 2, 3])  # Aggregate␣
↪gradients
    activations = model.get_activations(image).detach()  # Get feature maps

    for i in range(activations.shape[1]):
        activations[:, i, :, :] *= pooled_gradients[i]

    heatmap = torch.mean(activations, dim=1).squeeze().cpu()
    heatmap = nn.ReLU()(heatmap)
    heatmap /= torch.max(heatmap)
    heatmap = cv2.resize(heatmap.numpy(), (size, size))
    return heatmap
```

[22]:
```
image, label = validset[11]
denorm_image = image.permute(1, 2, 0) * np.array((0.229, 0.224, 0.225)) + np.
↪array((0.485, 0.456, 0.406))
image = image.unsqueeze(0).to(DEVICE)
pred = model(image)
heatmap = get_gradcam(model, image, pred[0][0], size=227)
utils.plot_heatmap(denorm_image, pred, heatmap)
```



[27]:
```
image, label = validset[5]
denorm_image = image.permute(1, 2, 0) * np.array((0.229, 0.224, 0.225)) + np.
↪array((0.485, 0.456, 0.406))
image = image.unsqueeze(0).to(DEVICE)
pred = model(image)
heatmap = get_gradcam(model, image, pred[0][0], size=227)
utils.plot_heatmap(denorm_image, pred, heatmap)
```



12 Project Report

Project Overview The project involves creating a custom CNN model for image classification, where the main focus is on interpreting model predictions using Grad-CAM. The dataset contains images of cucumbers, eggplants, and mushrooms, which are split into training and validation sets. Using the Grad-CAM technique, the model's predictions can be visualized to understand the regions within each image that influence the classifier.

Step-by-Step Breakdown

1. Dataset Setup and Augmentation

I used the provided CSV file to load image paths and labels, and applied data augmentation to improve model generalization. Training augmentations include rotations and flips, while the validation set only undergoes normalization to maintain consistency for evaluation.

2. Model Architecture

The model consists of a feature extractor and classifier. The feature extractor uses convolutional layers to capture spatial features, while the classifier maps these features to output classes. Additionally, I registered a hook in the feature extractor for Grad-CAM to store gradients during the backpropagation phase.

3. Training and Evaluation

The training loop utilizes an Adam optimizer and CrossEntropyLoss to optimize the model. The best model weights are saved based on validation loss improvements to prevent overfitting.

4. Grad-CAM Visualization

Grad-CAM is implemented by backpropagating gradients from the output class to the final convolutional layer. The gradients are pooled and multiplied by the activation maps to create a heatmap, which is overlaid on the original image.

5. Results and Analysis

After training, the model demonstrates good performance with a low validation loss. The Grad-CAM heatmaps reveal the regions most relevant to the predictions, making the model's decision process interpretable. For instance, the model correctly identifies mushrooms by focusing on the cap's distinctive shape and texture.

13 Conclusion

In conclusion, this project demonstrates a comprehensive approach to vegetable classification by leveraging a Convolutional Neural Network (CNN) model combined with Grad-CAM (Gradient-weighted Class Activation Mapping) to provide enhanced interpretability. Through rigorous training and testing, the CNN model effectively learned complex features from various vegetable images, achieving a high level of classification accuracy. The application of Grad-CAM offered valuable insights into the decision-making process of the model by visualizing the regions within each image that most influenced the classification outcome.

The integration of Grad-CAM not only improves model transparency but also serves as a diagnostic tool, making the predictions of deep learning models more accessible and interpretable to users. This transparency is critical for gaining trust in AI applications, especially in cases where the model's decision has significant implications, such as in the medical field or precision agriculture.

Moreover, the interpretability achieved with Grad-CAM could facilitate further improvements in model performance. By analyzing heatmaps, developers and researchers can identify patterns or biases, refine dataset quality, and optimize model parameters more effectively. This project thus demonstrates a practical and impactful approach to bridging the gap between model accuracy and interpretability, showcasing the potential of deep learning techniques not only to achieve high-performance classification but also to ensure that these results are explainable and actionable.