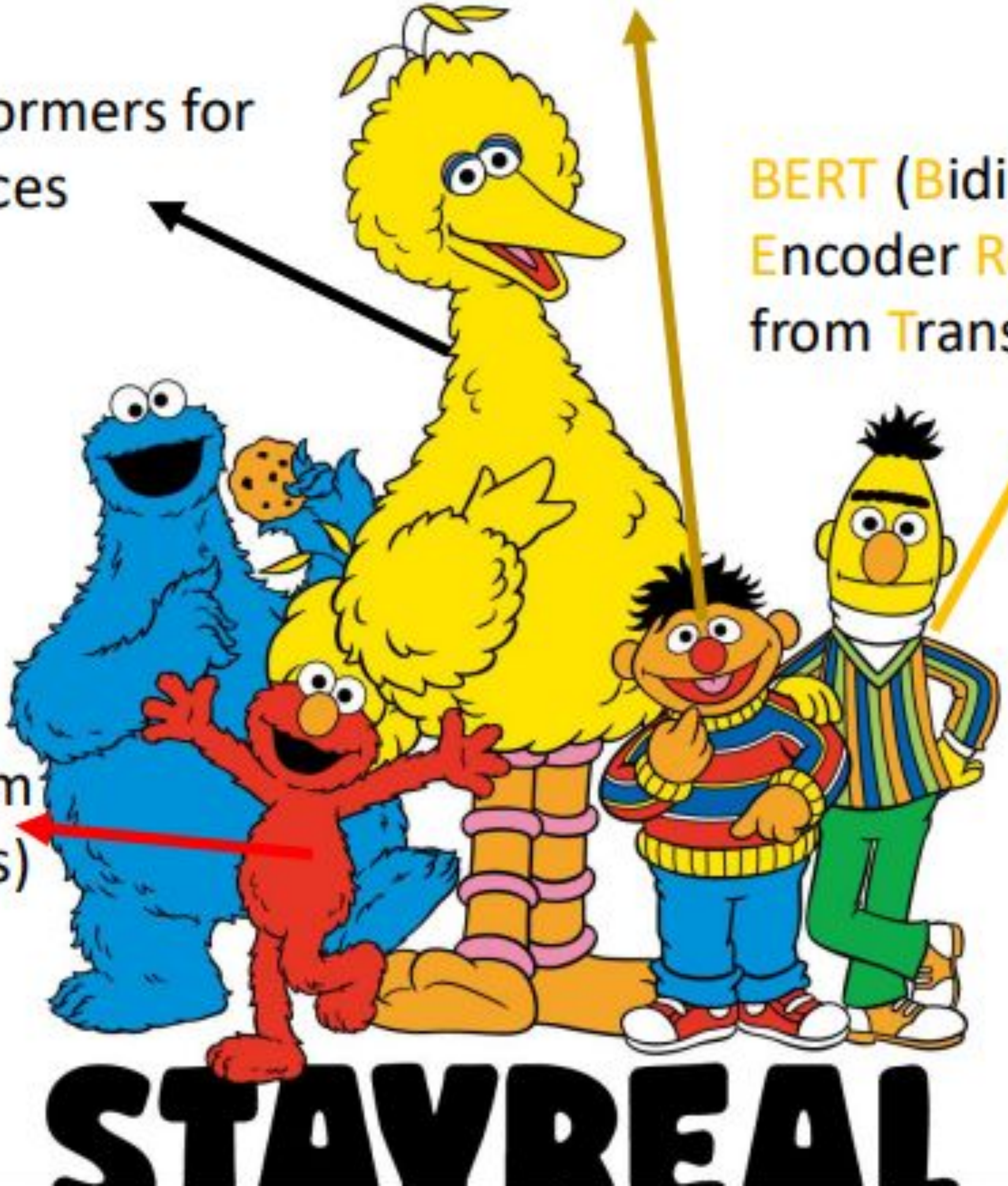
ERNIE (Enhanced Representation through Knowledge Integration)

Big Bird: Transformers for Longer Sequences

BERT (Bidirectional Encoder Representations from Transformers)



ELMo
(Embeddings from Language Models)

OSCAR: Explai(o)nable AI in Trash Classification(r)

Samara Goltz and
Bryn Archambault

What Is Our Project?

Trash Classification Using Convolutional Neural Networks
Project Category: Computer Vision

Yujie He
ICME
Stanford University
yujiehe@stanford.edu

Qinyue Gu
ICME
Stanford University
gqy94@stanford.edu

Maguo Shi
ICME
Stanford University
smgyl@stanford.edu

Abstract

As the waste problem becomes increasingly eminent across the globe, we aim to provide an automated waste sorting tool to make it easier for residents to classify trash. Our project used TrashNet [2] as our dataset, and classified recyclables or trash into six categories. To achieve our objective, we focused on Convolutional Neural Networks (CNN), and explored several well-known architectures at early stages. We ended up with modified AlexNet by taking two layers out, and experimented different techniques based on this model architecture, including dropout, data augmentation and learning rate decay. We also experimented two classifiers, Softmax and Support Vector Machine (SVM), as the last layer of our model structure. The highest test accuracy we achieved was 79.94% with the model using partial data augmentation and SVM classifier.

Problem Statement: What aspects of an image does AlexNet focus on when classifying an object within the TrashNet dataset? Additionally, how do different aspects of the model impact this?

Our Baseline Model

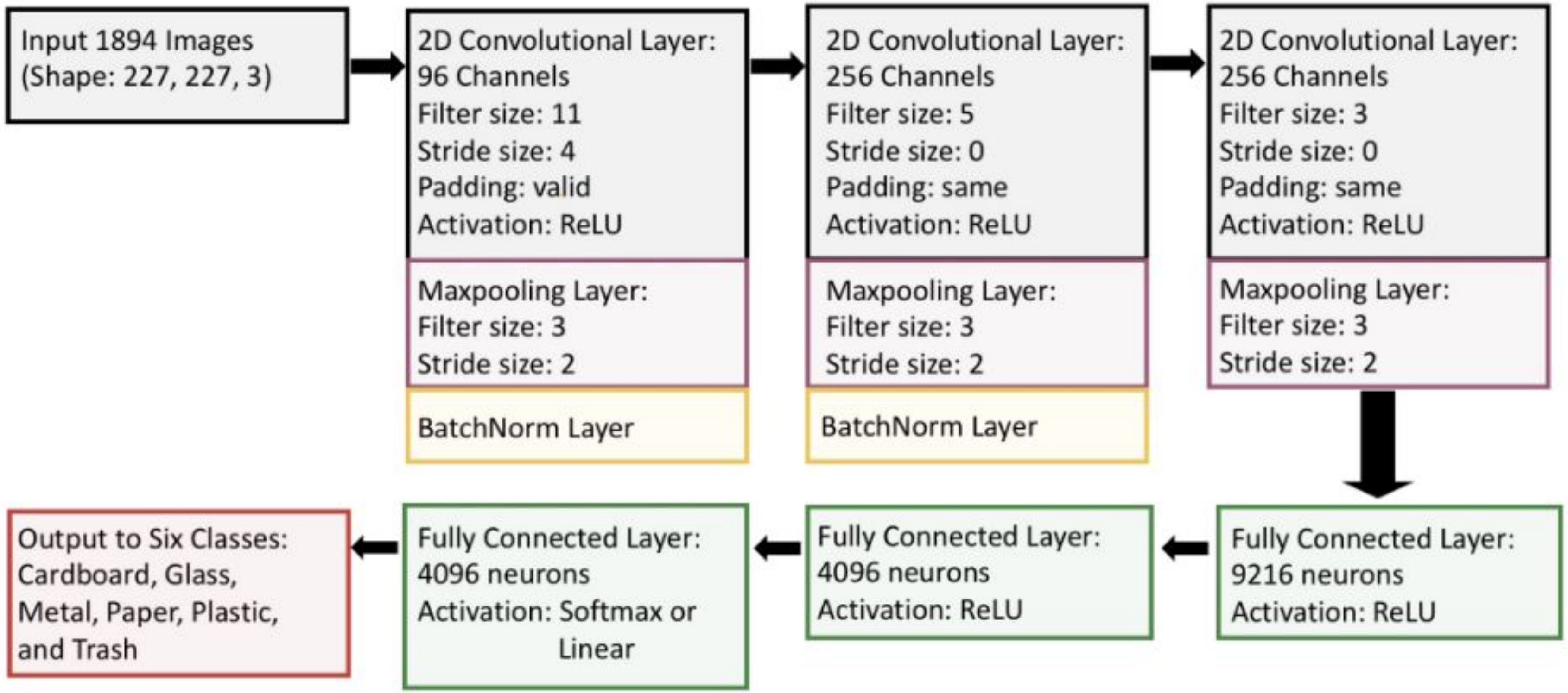


Figure 2: Basic Model Architecture

Comparison Plan

Using local explainable AI, perform ablation study using AlexNet Milestone #2 as the Baseline.

Focus on:

- Dropout
- Normalization
- ReLu
- CNN layers
- Max pooling - Resource Exhaustion error, requires too much memory
- Wanted to compare horizontally with AlexNet vs ResNet & VGG - Was going to take 16 & 28 hours to run

Compare:

- Accuracy
- Loss
- One explainable AI output image from each trash category

Our Inputs

TrashNet is a popular dataset for trash classification (6 features; 2,527 images):

- size (227 by 227)
- Plain background
- Trash is used, not perfect





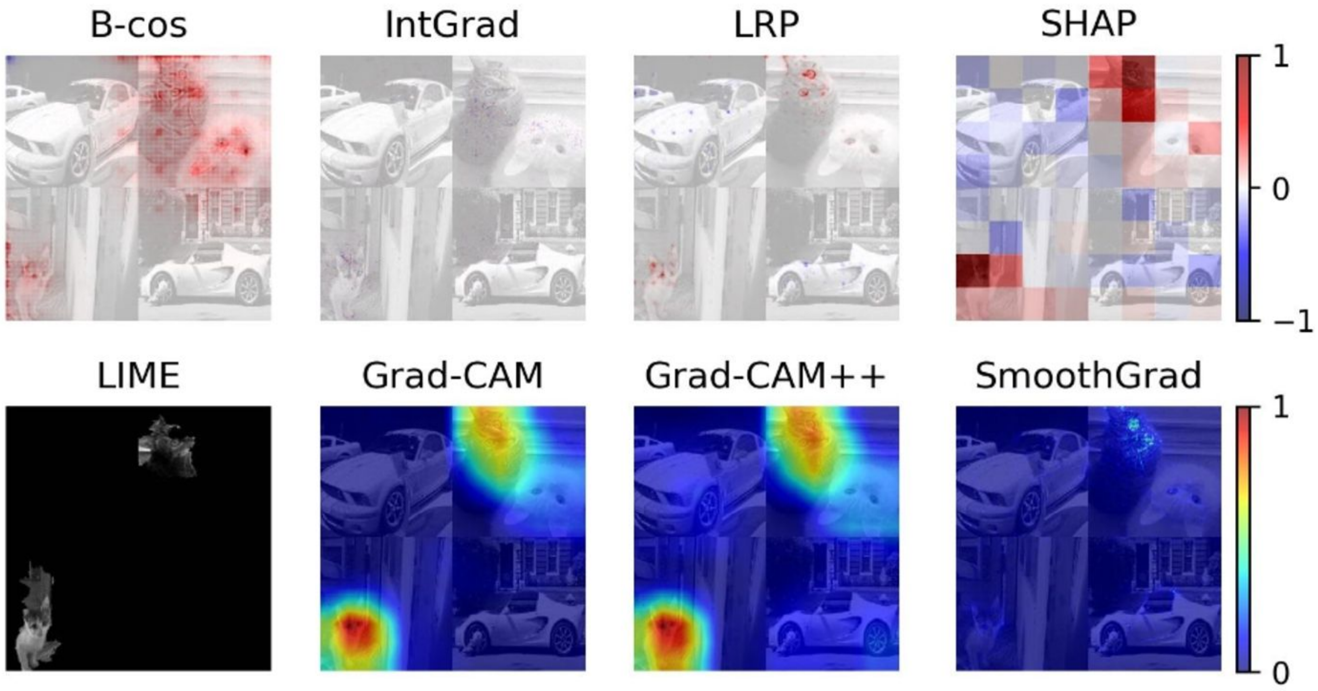

Dataset Troubleshooting

```
import os
import numpy as np
import tensorflow as tf

# Path to the "cardboard" subfolder
cardboard_folder_path = '/content/drive/MyDrive/Senior Year/Machine Learning/Final Project/dataset-resized/cardboard'

# Check if the folder exists
if os.path.exists(cardboard_folder_path) and os.path.isdir(cardboard_folder_path):
    cardboard = []
    for file_name in os.listdir(cardboard_folder_path):
        image_path = os.path.join(cardboard_folder_path, file_name)
        temp = tf.keras.preprocessing.image.load_img(
            path=image_path,
            color_mode='rgb',
            target_size=(227, 227)
        )
        X = np.array(temp)
        cardboard.append(X)

    cardboard = np.array(cardboard)
    cardboard = np.take(cardboard, np.random.permutation(cardboard.shape[0]), axis=0)
    print(cardboard.shape)
else:
    #print("The 'cardboard' folder does not exist or the path is incorrect.")
    cardboard_folder_path = '/content/drive/MyDrive/Junior Year/Machine Learning/Final Project/dataset-resized/cardboard'
    if os.path.exists(cardboard_folder_path) and os.path.isdir(cardboard_folder_path):
      cardboard = []
      for file_name in os.listdir(cardboard_folder_path):
        image_path = os.path.join(cardboard_folder_path, file_name)
        temp = tf.keras.preprocessing.image.load_img(
            path=image_path,
            color_mode='rgb',
            target_size=(227, 227)
            )
        X = np.array(temp)
        cardboard.append(X)

      cardboard = np.array(cardboard)
      cardboard = np.take(cardboard, np.random.permutation(cardboard.shape[0]), axis=0)
      print(cardboard.shape)
    else:
       print("The 'cardboard' folder does not exist or the path is incorrect.")
```

What is Explainable AI

What is it?

Explainable AI allows us to understand the inner workings of the model by opening up the “black box” to see what the model is focusing on



Benefits

- Trust in the model, important when considering money, health, safety, etc
- Sorting trash correctly is beneficial to the environment and is something simple everyone can do

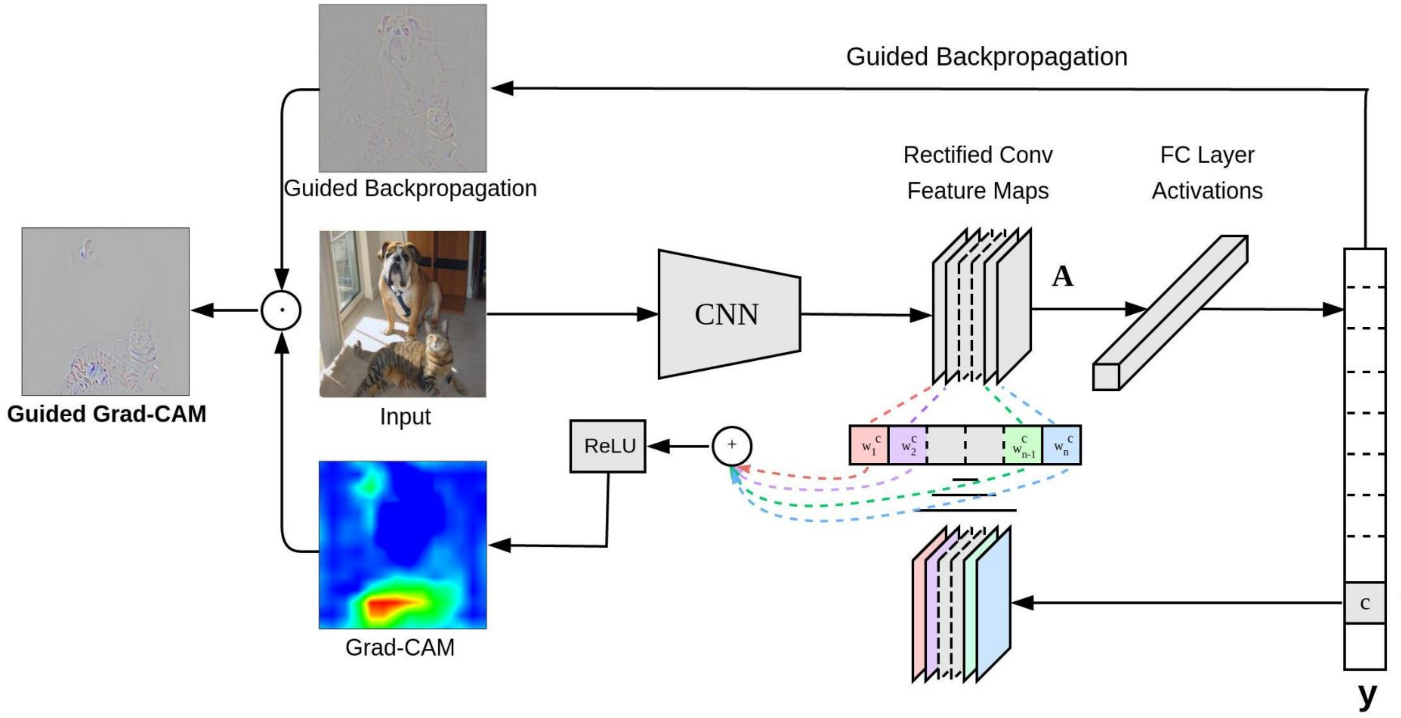
Different Types

- Saliency Maps
- LIME
- LRP
- Decision Trees (Forests) - Global

What is Grad-CAM

What is it?

Inputs: A trained CNN model (AlexNet), an input image, and the target class index.
Outputs: A heatmap showing which parts of the image influenced the prediction



How Does It Work?

1. Focuses on the last convolutional layer
2. Passes image through network, assigning it to a class (trash, glass, etc)
3. The **gradient** of the score for a specific class shows how much each neuron contributed to that score
4. The last CNN outputs a **feature map** that uses gradients to highlight the influence of a feature.
   a. Grad-CAM calculates the importance of each map, and weights them according to the gradients.
5. The weighted maps are combined into an **activation map**, and then normalized into a **heatmap**
6. Grad-CAM combines original image with heatmap

What is Grad-CAM

What is it doing?

1. Retrieves the output of the last con. layer
2. Computes the gradient
3. Averages gradients spatially to get “importance weights” (pooled gradients) for each channel
4. Weighs the layer’s outputs by these gradients to create a heatmap
5. Rescales and normalizes heatmap for visualization

```
[ ] def grad_cam(model, image, class_idx):
        """Generates a heat map using Grad-CAM."""
        print(f"Inside grad_cam, image shape: {image.shape}")
        # Expand dimensions of the image to match input shape
        image = np.expand_dims(image, axis=0)  # Ensure batch dimension is included
        # Get the gradient model
        grad_model = tf.keras.Model(
            inputs=[model.inputs], outputs=[model.get_layer("conv5").output, model.output])

        with tf.GradientTape() as tape:
            last_conv_layer = model.get_layer('conv5')  # Ensure this matches your model's last conv layer
            iterate = tf.keras.models.Model([model.input], [last_conv_layer.output, model.output])

            # Use the already expanded image
            conv_outputs, preds = iterate(image)  # Image is now (1, 227, 227, 3)
            loss = preds[:, class_idx]

        grads = tape.gradient(loss, conv_outputs)[0]
        pooled_grads = tf.reduce_mean(grads, axis=(0, 1))  # Average over the spatial dimensions
        # Convert the conv_outputs to NumPy array
        heatmap = conv_outputs[0].numpy()  # Convert to NumPy array for manipulation

        print(f"Shape of pooled_grads: {pooled_grads.shape}")  # Debugging line
        print(f"Shape of heatmap before scaling: {heatmap.shape}")  # Debugging line

        # Ensure pooled_grads is a numpy array
        pooled_grads = pooled_grads.numpy()  # Convert to NumPy for manipulation
        pooled_grads = np.expand_dims(pooled_grads, axis=(0, 1))  # Shape becomes (1, 1, 256)

        # Check shapes before multiplication
        print(f"Shape of pooled_grads after expansion: {pooled_grads.shape}")  # Debugging line
        print(f"Shape of heatmap before applying pooled_grads: {heatmap.shape}")  # Debugging line

        # Multiply each channel in the heatmap with the corresponding pooled gradient
        # Ensure correct dimensions for multiplication
        heatmap = np.tensordot(heatmap, pooled_grads, axes=(2, 2))  # Shape will be (13, 13)
        heatmap = np.maximum(heatmap, 0)  # Ensure non-negative values
        heatmap /= np.max(heatmap)  # Normalize to [0, 1]

        # Squeeze the heatmap
        heatmap = np.squeeze(heatmap)  # Remove dimensions of size 1
        # Check heatmap shape before resizing
        print(f"Heatmap shape before resizing: {heatmap.shape}")
        # Resize heatmap to match the input image dimensions
        if heatmap.ndim == 2:  # Ensure the heatmap is 2D
            heatmap = cv2.resize(heatmap, (image.shape[2], image.shape[1]))  # Adjust to your image shape
        else:
            raise ValueError("Heatmap does not have the expected shape for resizing.")
        return heatmap
```

What is Grad-CAM

What is it doing?

Inputs: The original image and the heatmap

Outputs: Visualization of the heatmap over the original image

1. Converts heatmap to RGB color map
2. Blends the heatmap with the original image to create a superimposed image
3. Displays the superimposed image using matplotlib

```python
def display_grad_cam(image, heatmap):
    # Resize heatmap to match the original image size
    heatmap = np.uint8(255 * heatmap)  # Scale to 0-255
    heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)  # Apply color map

    # Overlay the heatmap on the original image
    superimposed_img = heatmap * 0.4 + image  # Adjust alpha as needed
    plt.imshow(superimposed_img / 255.0)  # Display as float image
    plt.axis('off')
    plt.show()

# Run Grad-CAM on the selected images
print(f"Number of selected images: {len(selected_images)}")  # Debug statement

for idx, selected_image in enumerate(selected_images):
    print(f"Selected image {idx} shape: {selected_image.shape}")  # Debug statement

    # Check if the loop is entering the body
    if selected_image is None:
        print(f"Image {idx} is None.")  # Check for None
        continue  # Skip if None

    # Ensure we expand the dimensions correctly for the model
    expanded_image = np.expand_dims(selected_image, axis=0)
    print(f"After expansion, expanded image shape: {expanded_image.shape}")

    # Predict class using the model
    class_idx = np.argmax(model.predict(expanded_image))
    print(f"Image {idx}: Predicted class index: {class_idx}")

    # Call Grad-CAM
    print(f"Calling grad_cam for image {idx} with class index {class_idx}")
    heatmap = grad_cam(model, np.squeeze(selected_image), class_idx)  # Remove extra dimensions
    display_grad_cam(selected_image, heatmap)
```

Our Baseline Model

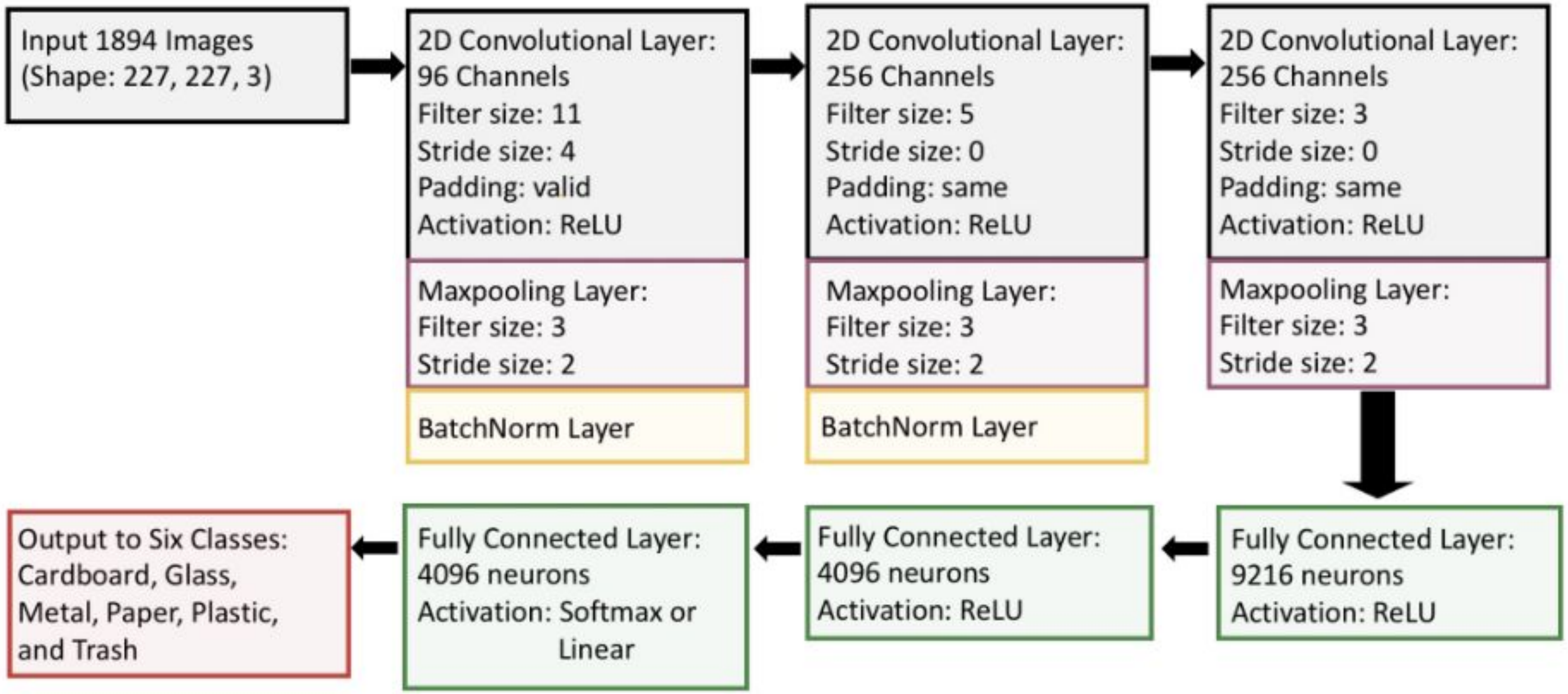


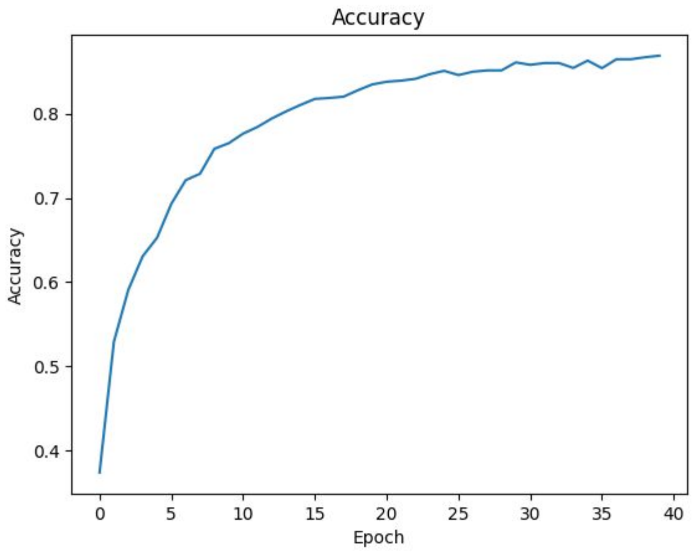
Figure 2: Basic Model Architecture

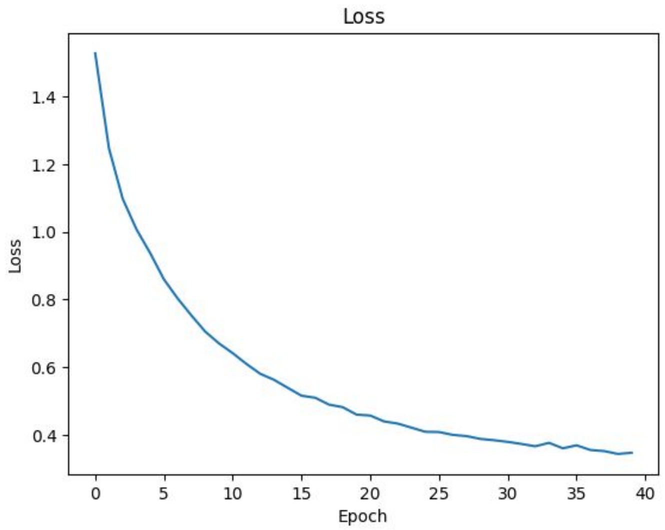
Baseline Outputs

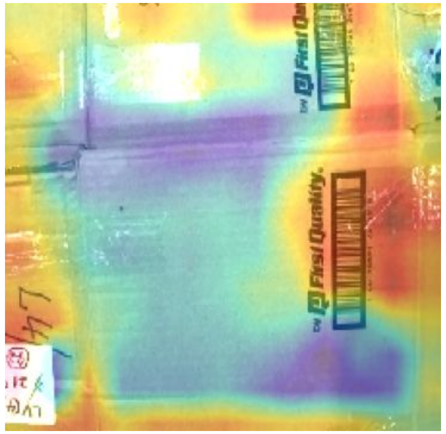
Accuracy: 0.8709

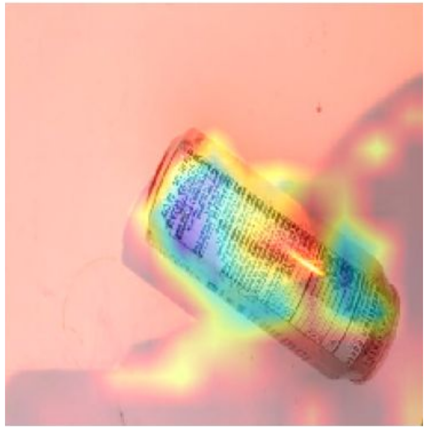
Loss: 0.3401

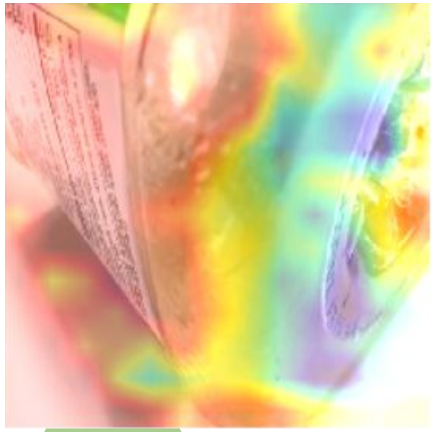
Blue indicates what the model is focusing on

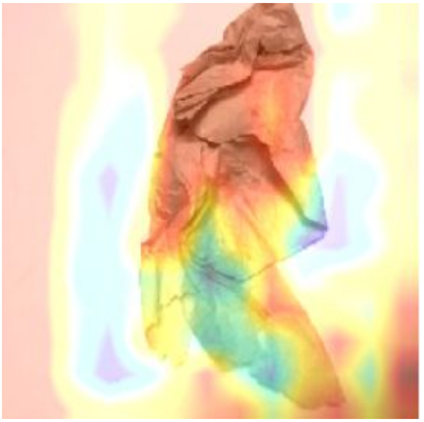












Adding Dropout

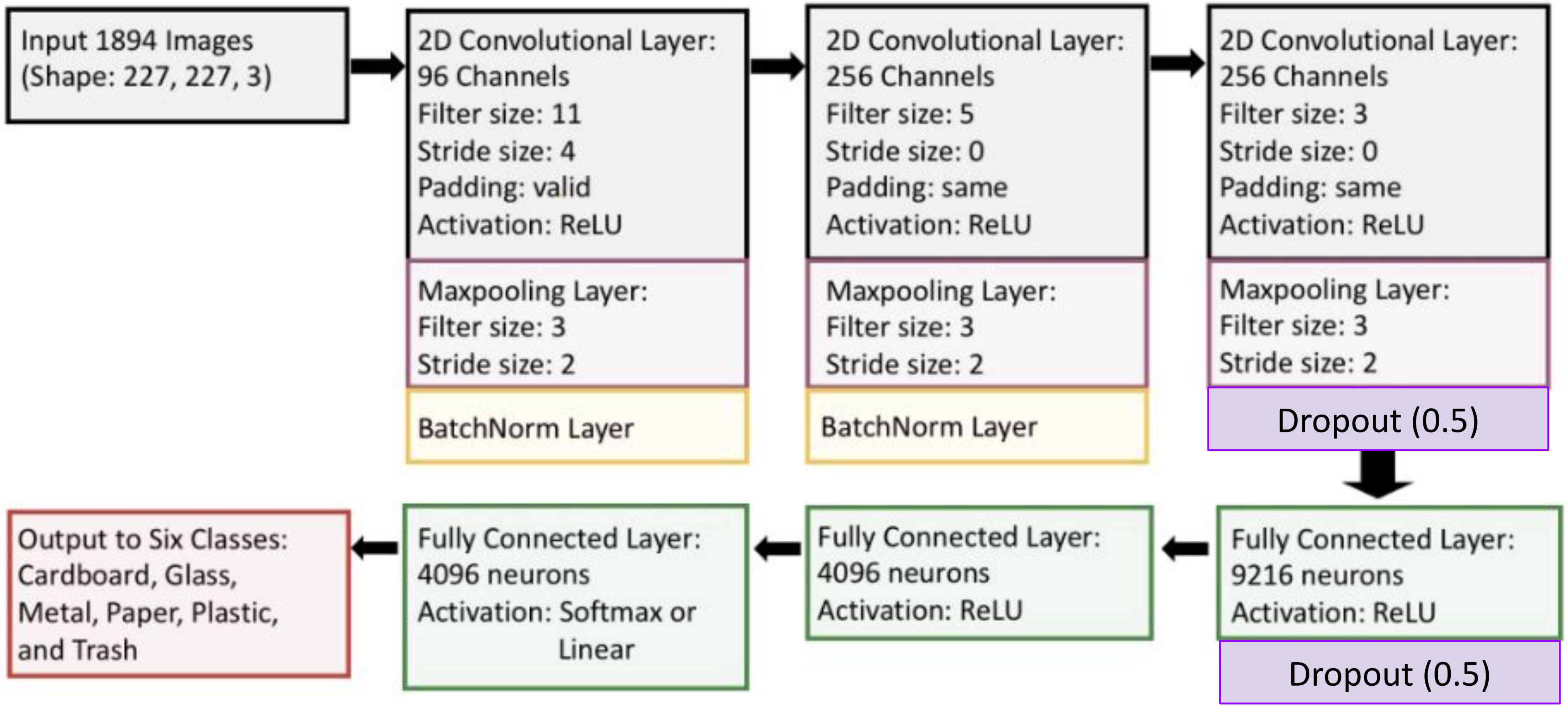
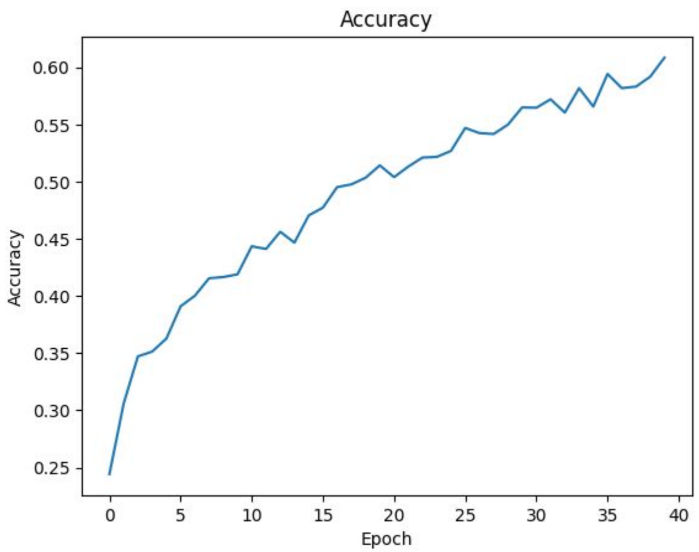


Figure 2: Basic Model Architecture

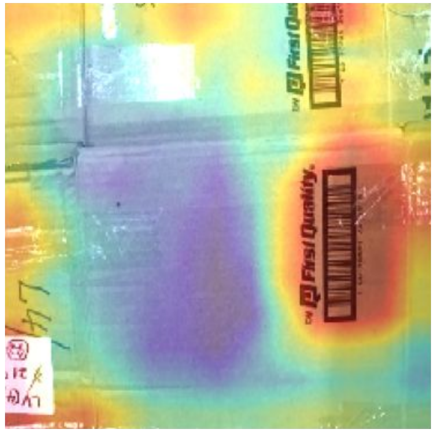
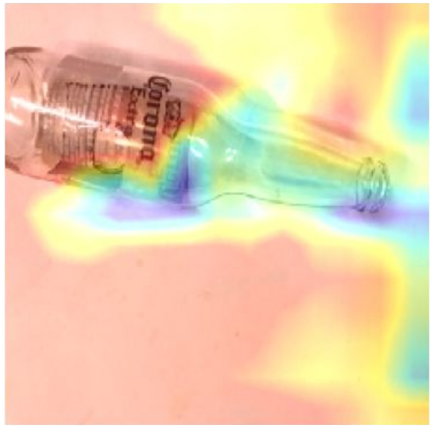
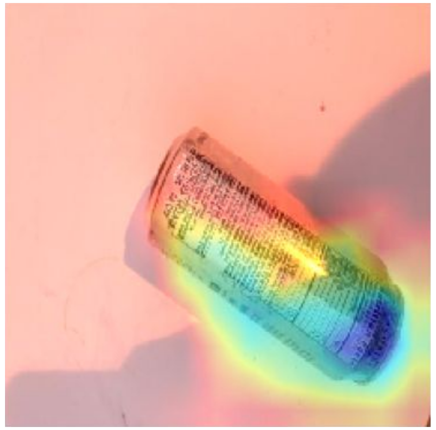


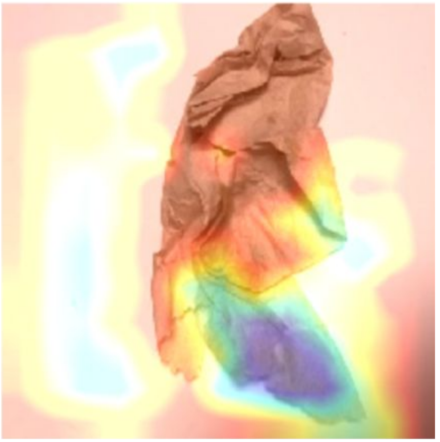
Dropout Change

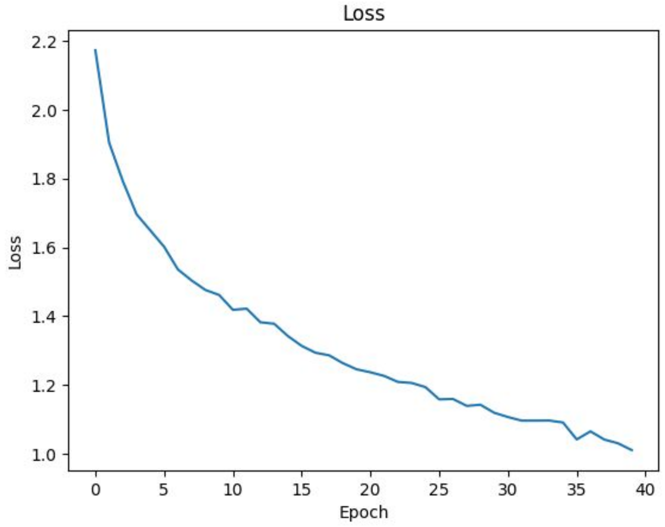
Accuracy: 0.6147 (0.8709)

Loss: 0.9988 (0.3401)

Blue indicates what the model is focusing on



Removing Normalization

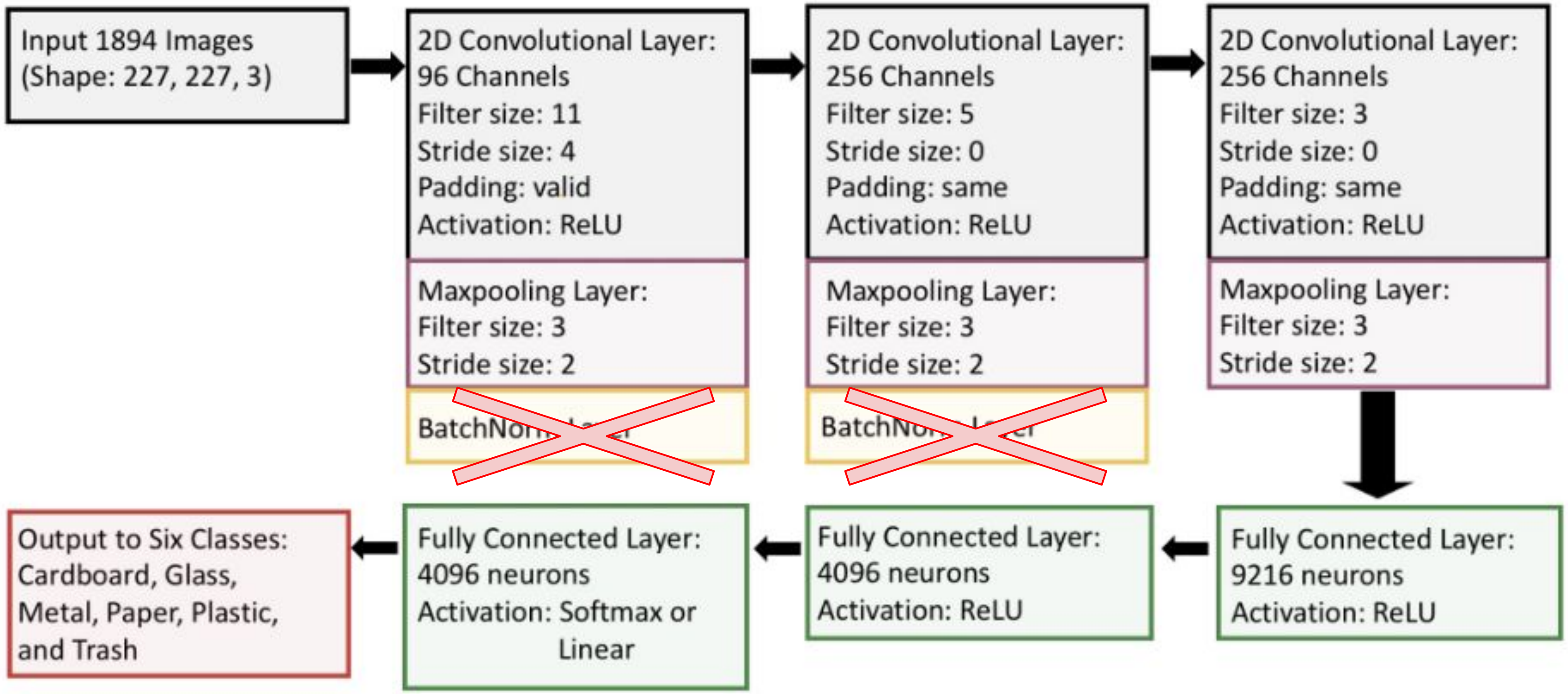
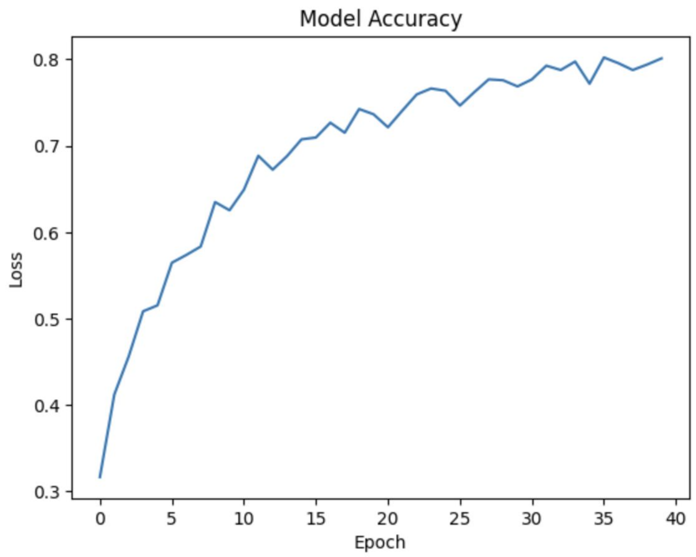
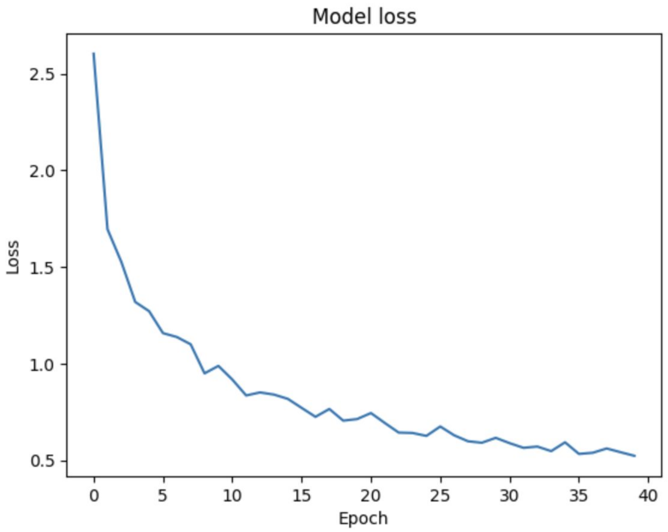


Figure 2: Basic Model Architecture





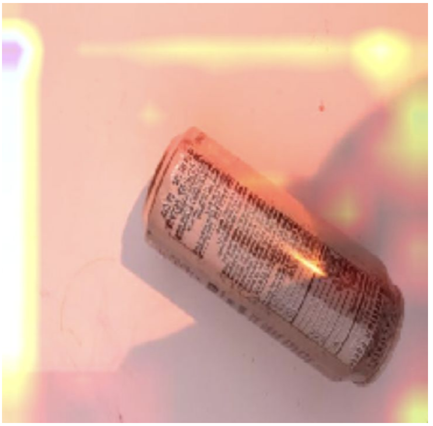
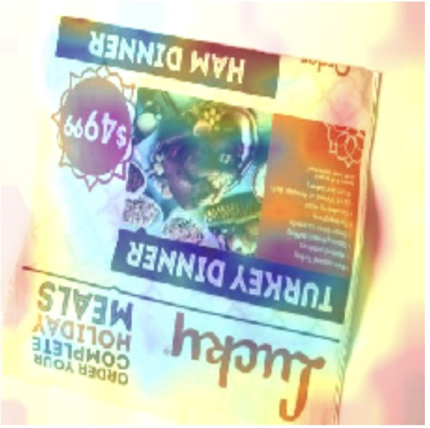
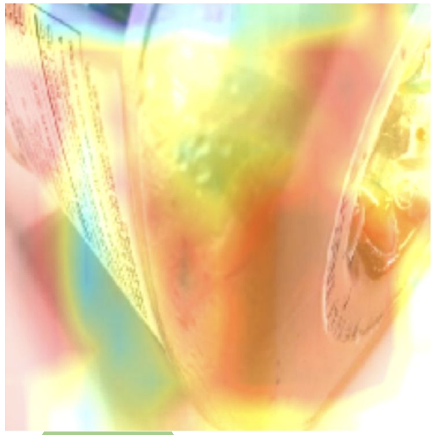
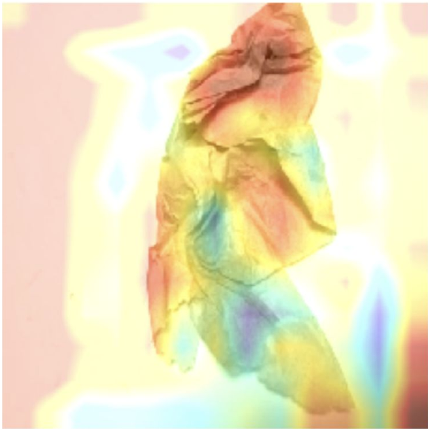
Normalization Change

Accuracy: 0.8161 (0.8709)

Loss: 0.4870 (0.3401)

Blue indicates what the model is focusing on



Removing ReLU

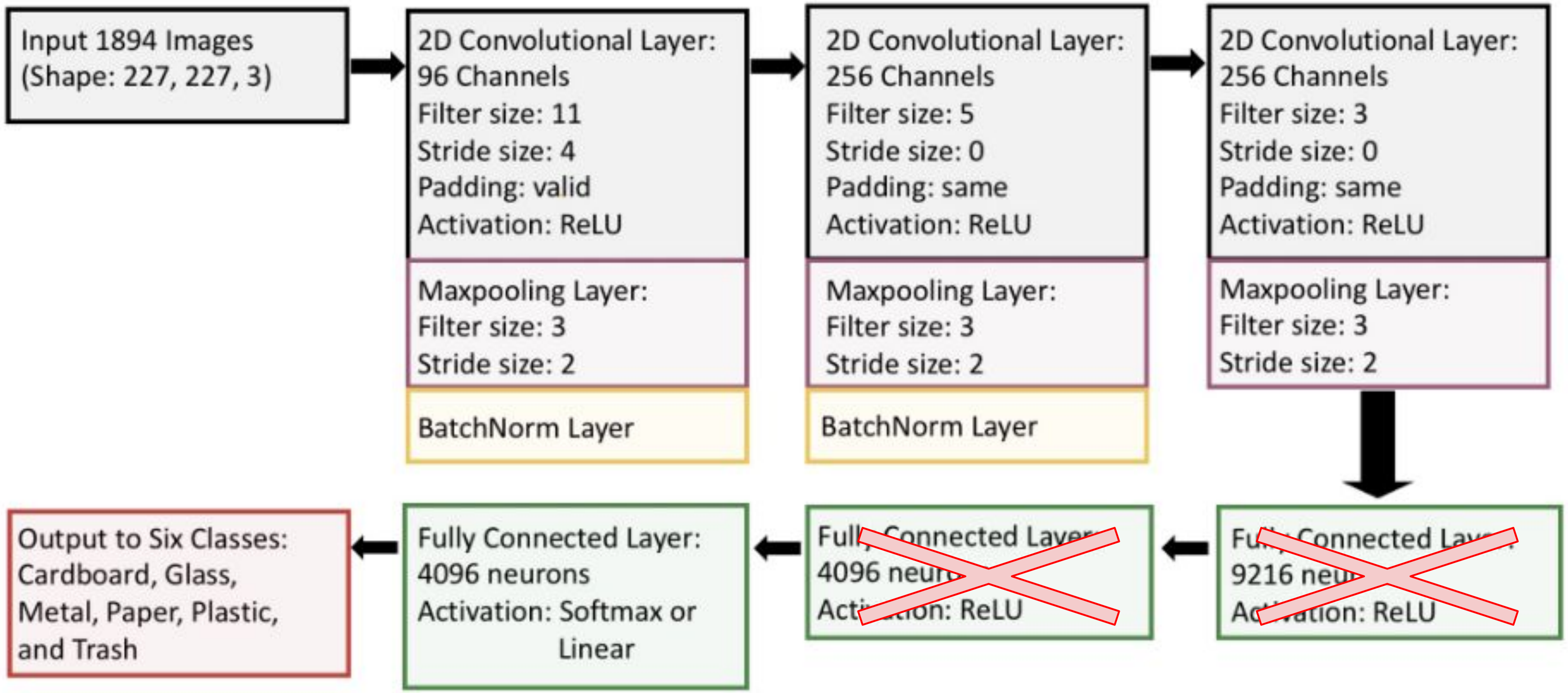


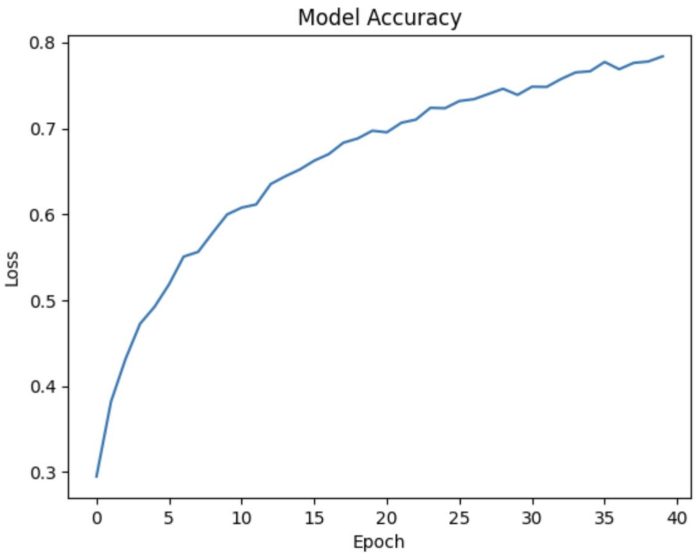
Figure 2: Basic Model Architecture

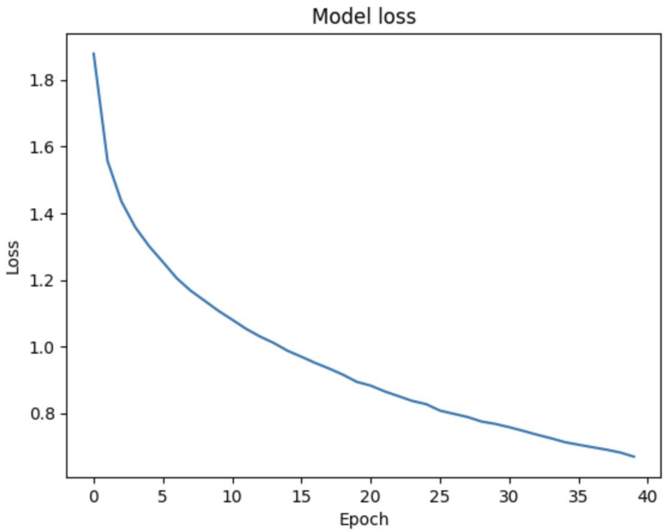
ReLU Change

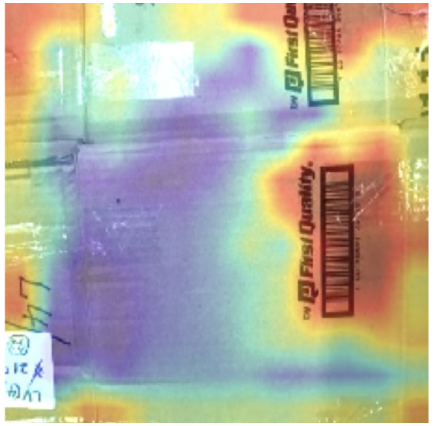
Accuracy: 0.6021 (0.8709)

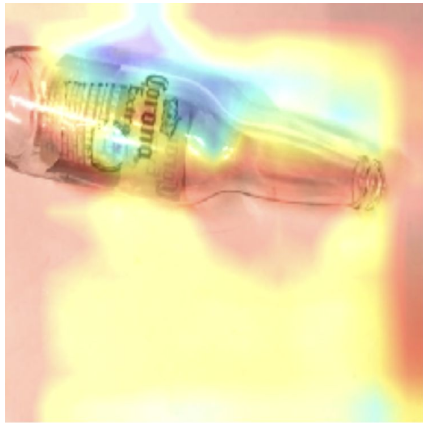
Loss: 1.1777 (0.3401)

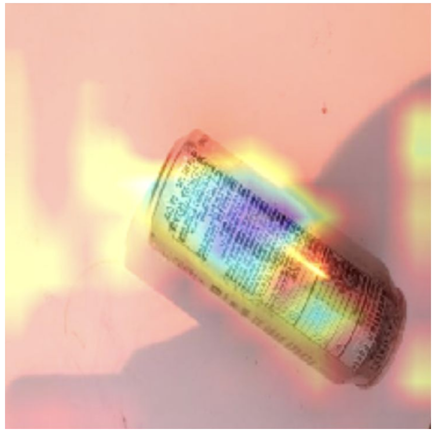
Blue indicates what the model is focusing on

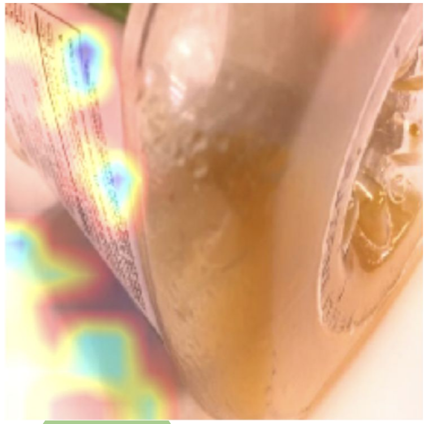












Removing CNN layers

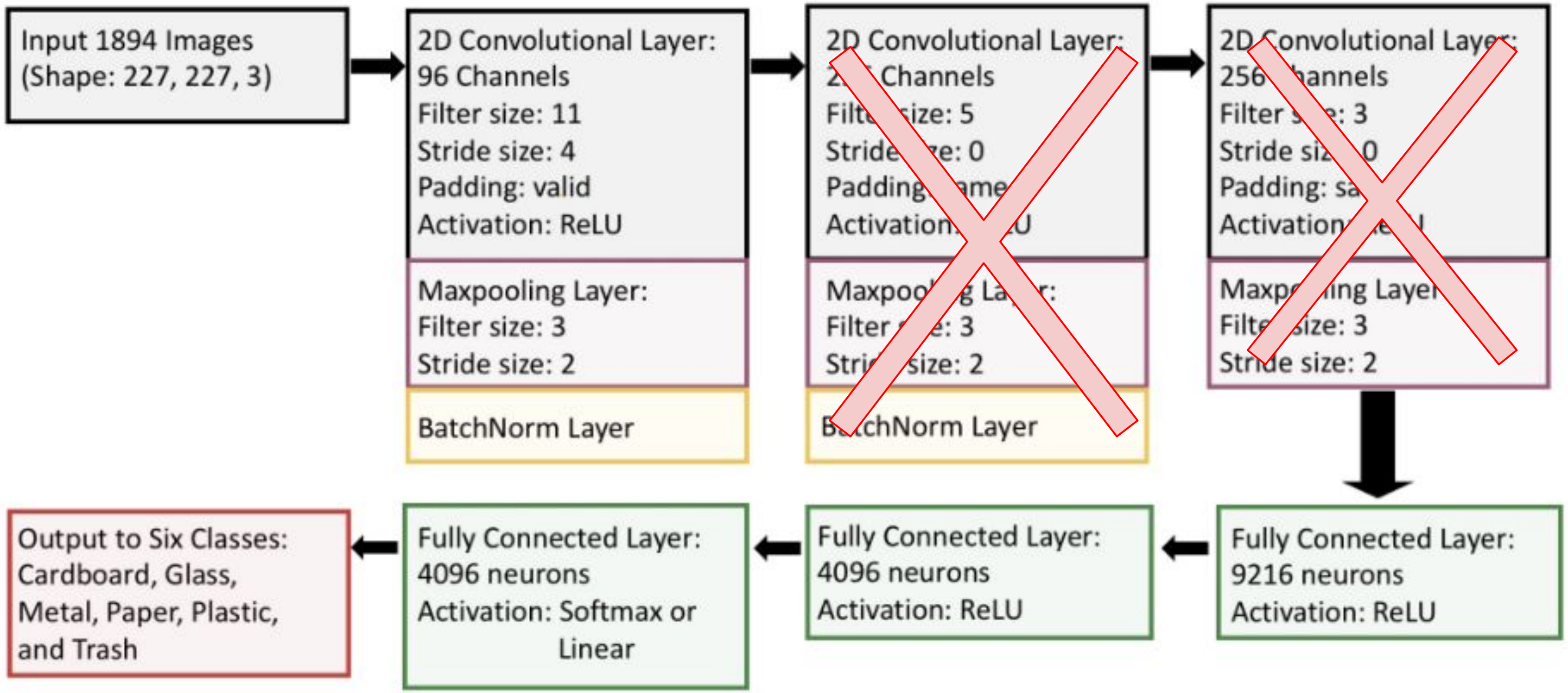
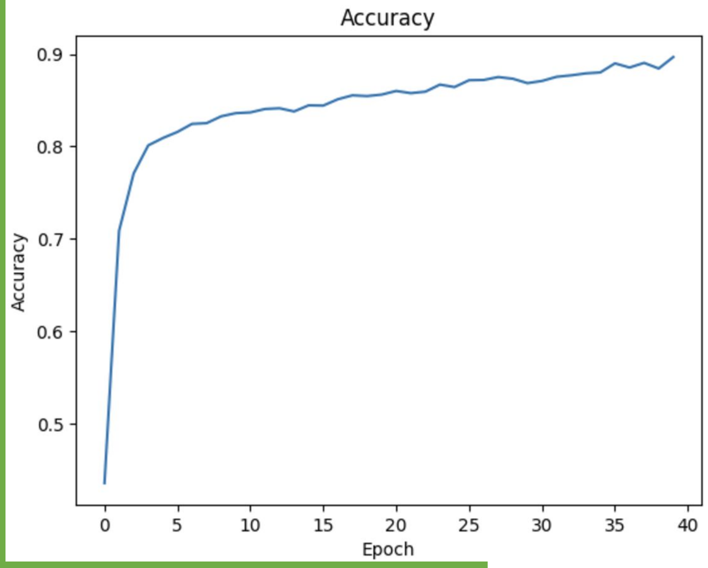
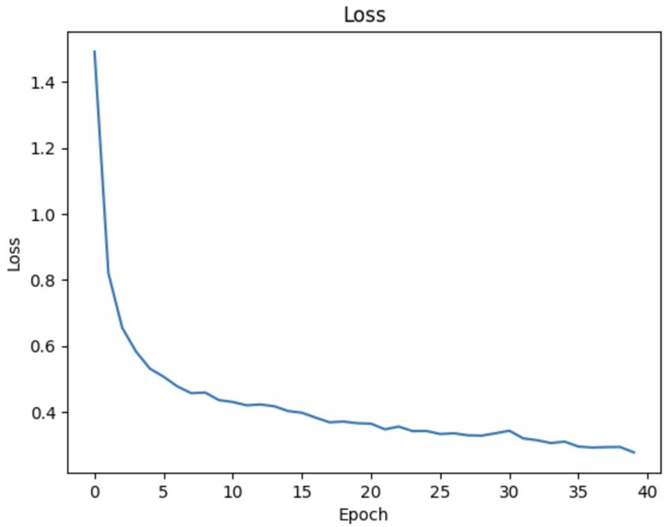


Figure 2: Basic Model Architecture





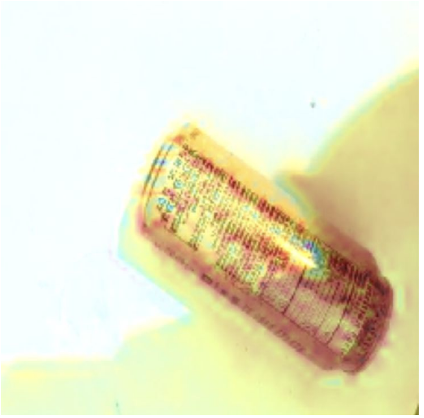
CNN Changes

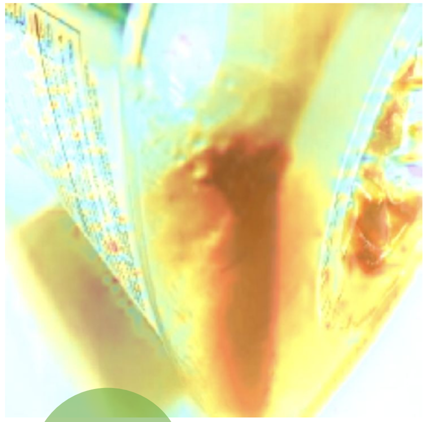
Accuracy: 0.8920 (0.8709)

Loss: 0.2812 (0.3401)

Blue indicates what the model is focusing on



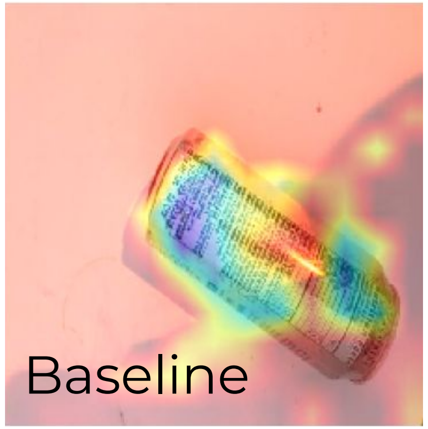





Notable Differences

Accuracy:	B: 0.8709	N: 0.6147
Loss:	B: 0.3401	N: 0.9988

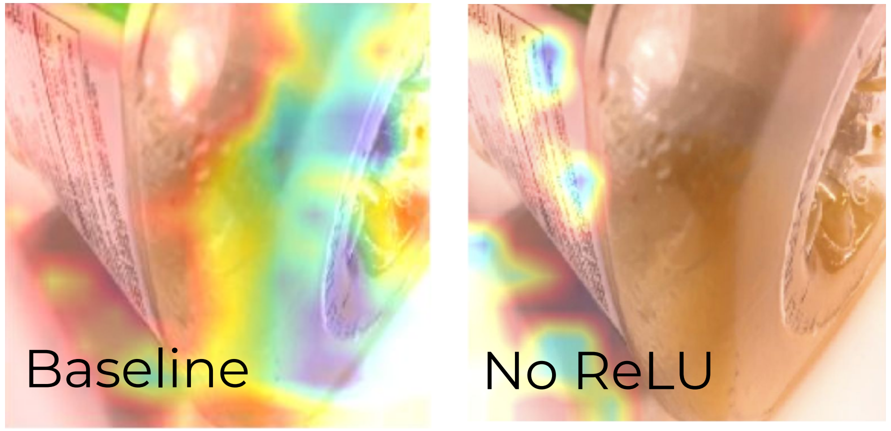
Baseline model focuses better and has a better accuracy, shows dropout is not necessary for this trash classification.



Accuracy:	B: 0.8709	N: 0.8161
Loss:	B: 0.3401	N: 0.4870

While accuracy and loss are similar for both models, explainable AI clearly shows that normalization is needed, as it doesn't focus on the can when removed.

Notable Differences



Accuracy: B: 0.8709 N: 0.6021

Loss: B: 0.3401 N: 1.1777

ReLU seems to help focus the model and make the focus more intense. It should be kept in the model.

Accuracy: B: 0.8709 N: 0.8920

Loss: B: 0.3401 N: 0.2812

Removing the 2 CNN layers makes the model lose focus and intensity. While the accuracy and loss get better, the model appears to have sparse attention.

Challenges

- Issues uploading Dataset
- Updating old code for image augmentation
  - Converting images to same dimensions (227 x 227), was 512 X 384
  - Dimension inflation issues with Grad-CAM
- Space and time issues
  - Extremely long runtimes which limited our scope
  - limited GPU access

Conclusion:

Our findings suggest that they chose the right elements for their model as the baseline was the best in accuracy, loss, and where the model was focusing.

For the future

We ended up implementing 2 chunks of code in order to get all of these comparisons. We would recommend utilizing this tool whenever you decide to improve a model (given sufficient resources).

```
grad_cam(model, image, class_idx):
"""Generates a heat map using Grad-CAM."""
print(f"Inside grad_cam, image shape: {image.shape}")

# Expand dimensions of the image to match input shape
image = np.expand_dims(image, axis=0)  # Ensure batch dimension is included

# Get the gradient model
grad_model = tf.keras.Model(
    inputs=[model.inputs], outputs=[model.get_layer("conv5").output, model.output]
)

with tf.GradientTape() as tape:
    last_conv_layer = model.get_layer('conv5')  # Ensure this matches your model's last conv layer
    iterate = tf.keras.models.Model([model.input], [last_conv_layer.output, model.output])

    # Use the already expanded image
    conv_outputs, preds = iterate(image)  # Image is now (1, 227, 227, 3)
    loss = preds[:, class_idx]

grads = tape.gradient(loss, conv_outputs)[0]
pooled_grads = tf.reduce_mean(grads, axis=(0, 1))  # Average over the spatial dimensions

# Convert the conv_outputs to NumPy array
heatmap = conv_outputs[0].numpy()  # Convert to NumPy array for manipulation

print(f"Shape of pooled_grads: {pooled_grads.shape}")  # Debugging line
print(f"Shape of heatmap before scaling: {heatmap.shape}")  # Debugging line

# Ensure pooled_grads is a numpy array
pooled_grads = pooled_grads.numpy()  # Convert to NumPy for manipulation
pooled_grads = np.expand_dims(pooled_grads, axis=(0, 1))  # Shape becomes (1, 1, 256)

# Check shapes before multiplication
print(f"Shape of pooled_grads after expansion: {pooled_grads.shape}")  # Debugging line
print(f"Shape of heatmap before applying pooled_grads: {heatmap.shape}")  # Debugging line

# Multiply each channel in the heatmap with the corresponding pooled gradient
# Ensure correct dimensions for multiplication
heatmap = np.tensordot(heatmap, pooled_grads, axes=(2, 2))  # Shape will be (13, 13)

heatmap = np.maximum(heatmap, 0)  # Ensure non-negative values
heatmap /= np.max(heatmap)  # Normalize to [0, 1]

# Squeeze the heatmap
heatmap = np.squeeze(heatmap)  # Remove dimensions of size 1

# Check heatmap shape before resizing
print(f"Heatmap shape before resizing: {heatmap.shape}")

# Resize heatmap to match the input image dimensions
if heatmap.ndim == 2:  # Ensure the heatmap is 2D
    heatmap = cv2.resize(heatmap, (image.shape[2], imag
else:
    raise ValueError("Heatmap does not have the expecte

return heatmap
```

```
display_grad_cam(image, heatmap):
    # Resize heatmap to match the original image size
    heatmap = np.uint8(255 * heatmap)  # Scale to 0-255
    heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)  # Apply color map

    # Overlay the heatmap on the original image
    superimposed_img = heatmap * 0.4 + image  # Adjust alpha as needed
    plt.imshow(superimposed_img / 255.0)  # Display as float image
    plt.axis('off')
    plt.show()

# Run Grad-CAM on the selected images
print(f"Number of selected images: {len(selected_images)}")  # Debug statement

for idx, selected_image in enumerate(selected_images):
    print(f"Selected image {idx} shape: {selected_image.shape}")  # Debug statement

    # Check if the loop is entering the body
    if selected_image is None:
        print(f"Image {idx} is None.")  # Check for None
        continue  # Skip if None

    # Ensure we expand the dimensions correctly for the model
    expanded_image = np.expand_dims(selected_image, axis=0)
    print(f"After expansion, expanded image shape: {expanded_image.shape}")

    # Predict class using the model
    class_idx = np.argmax(model.predict(expanded_image))
    print(f"Image {idx}: Predicted class index: {class_idx}")

    # Call Grad-CAM
    print(f"Calling grad_cam for image {idx} with class index {class_idx}")
    heatmap = grad_cam(model, np.squeeze(selected_image), class_idx)  # Remove extra dimensions
    display_grad_cam(selected_image, heatmap)
```

For Milestone 2, we would hope to have the resources to

- compare AlexNet with ResNet and VGG.
- Implement saliency maps
- Further vertical comparisons, add more to the model
- Global explanation
- Incorporate Mask-R-CNN to differ between overlapped classes (plastic)

Q&A