DL Project Checkpoint Report

CSE 676 Spring 2024
Team 129

Contribution Chart

Name	**UBIT**	**Contribution**
Akshobhya Sharma	akshobhy	33%
Ibrahim Bahadir Altun	ialtun	33%
Nazmus Saquib	nsaquib2	33%

1. Project name

Identifying products in grocery catalogs using image classification

2. Overview

We developed a CNN model that will take images of products as input and be able to identify them from the catalog of a particular grocery to find information about the product through multiclass classification.

This report catalogs our final solution as well as the limitations that we have faced in the process of building this setup.

Although we aimed to achieve 90% accuracy with test data, after trying out multiple models the best we have been able to achieve is 85.19% with 10 classes. We aim to refine this problem further as the future work of this project. Our model's outputs will act as labels for product classification for user input.

3. Objectives

Our objective in this project is to have a multiclass classifier capable of matching the given products to the items in our catalog with the best accuracy we can come up with. Our solution makes the products more accessible to the customers by making the process of searching for the product more convenient.

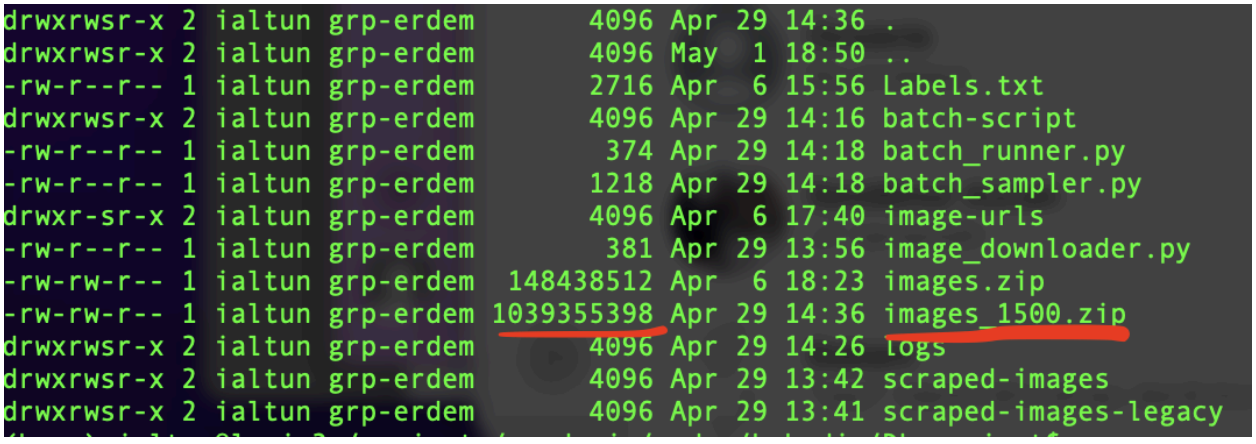
The problem is interesting because it is an intuitive and convenient way for users to search for products with the advent of VR and AR technologies. By focusing on visual information, which is going to be a go-to technology adopted by the majority soon, we see this as a relevant area of exploration.

4. Data Pipeline

4.1. Introduction

Despite great advancements in AI models, proper datasets are a very important indicator of the performance of AI models. For this reason, we put effort into finding datasets for our problem. However, since the problem and required dataset is relatively specific, we have to come up with a solution for collecting data.

First and foremost, we generated classes that will be used to generate images. Some of these classes include, but not limited to, “Laptop”, “Wall-Clock”, “Bottle”, “Sunglasses”, and more. Our main emphasis was collecting 100 images per class to train the models. However, after our experiments using 100 images per class, we realized that the models perform very poorly. And, specifically, it might get confused since some classes have similar images. Therefore, we decided to increase the number of images per class, and we collected 200 images per class. Even though it is a decent increase, our models still performed poorly. We investigated the proper ratios between number of classes and number of samples, and conducted experimental analysis on this. In conclusion, we show that, for most models to perform above 80% of accuracy, the expected ratio between number of images and number of classes should be 1500 per class. Therefore, for every class we collected 1500 images in total. Since we decided to work on 10 classes, the approximate number of total images we collect is 15000, which costed around 1GB.



Collecting 1GB data online, since there is no public dataset for our purpose, is a challenge. Therefore, we conducted an analysis on the doability of this, and followed a

few tutorials on data scraping using Python. One way of collecting data is using Python libraries that allow us to scrape data online.

However, most of these libraries are just parsers, thus, using it on some websites could dramatically slow the process. We used a local scraping tool and realized that it approximately takes 10 minutes to download just 5 images. This is usually because of the search engine policies. In order to solve this issue, we came up with a two step approach using eBay's search engine. 1) We first collected the URLs of each image instead of downloading it, 2) we use Center for Computational Research (CCR) to run 750 jobs simultaneously that will download the image from the collected URL. We will explain these 2 steps in the next sections.

4.2. Approach

Collecting Image URLs

The reasonable number of Python libraries allow many individuals to collect data. Similarly, we use BeautifulSoup, which is a Python library for pulling data out of HTML and XML files. However, as we explained before, downloading the plain image takes time on a single computer. Therefore, we first collected links of images instead of downloading them. For example:

```
https://www.ebay.com/sch/i.html?_from=R40&_nkw=Laptop&_sacat=0&LH_TitleDesc=0&_ipg=240&_pgn=1
```

This url returns an HTML file which has images on it. We set the maximum images per page to 240 as it is the maximum amount eBay allows. And we repeat this process for almost 6 pages for a class to collect links for a single class. Overall algorithm is given below:

```
for cls in classes:
  links = []
  for i in range(5):
     htmldata = getdata("search URL")
     soup = BeautifulSoup(htmldata, 'html.parser')
     links.append(soup.find_all('img'))
  file.write(links)
```

With this approach, we are able to collect links of 1500 images per class, without downloading. You may see a few lines on an example class and their links.

```
(base) ialtun@login1:/projects/academic/erdem/bahadir/DL-project$ head image-urls/laptop.txt
https://ir.ebaystatic.com/rs/v/fxxj3ttftm5ltcqnto1o4baovyl.png
https://i.ebayimg.com/thumbs/images/g/Tx4AAOSwyxtleh2t/s-l300.jpg
https://i.ebayimg.com/thumbs/images/g/VLcAAOSwz6hl4LXT/s-l300.jpg
https://i.ebayimg.com/thumbs/images/g/l9gAAOSwohZmDXtq/s-l300.jpg
https://i.ebayimg.com/thumbs/images/g/XgEAAOSwZSxmBE~D/s-l300.jpg
https://i.ebayimg.com/thumbs/images/g/hj0AAOSw~uRl~IJ5/s-l300.jpg
https://i.ebayimg.com/thumbs/images/g/MBUAAOSw59JmEAQf/s-l300.jpg
https://i.ebayimg.com/thumbs/images/g/OZAAAOSwEfJl1L3q/s-l300.jpg
https://i.ebayimg.com/thumbs/images/g/8GAAAOSwDwtjGz5F/s-l300.jpg
https://i.ebayimg.com/thumbs/images/g/zqQAAOSwx5VlzUCa/s-l300.jpg
(base) ialtun@login1:/projects/academic/erdem/bahadir/DL-project$
```

Downloading Collected URLs

Even though we are successfully able to collect image URLs, it is still not practical to download them on a single instance. Therefore, we utilize CCR to create multiple jobs that will download image batches separately. Since we have to download 15000 images, we use 750 jobs that will download 20 images separately. You may see the number of batch scripts we have in the below image, however, since we try multiple runs and multiple experiments here, the amount is shown a lot larger. However, with the last approach, we run only 750 jobs in a single submission.

```
(base) ialtun@login1:/projects/academic/erdem/bahadir/DL-project$ ls batch-script/ | wc -l
2151
(base) ialtun@login1:/projects/academic/erdem/bahadir/DL-project$
```

To automate this process, we 1) generate a base batch script, which is a base script that will be used as a model to generate 750 batch scripts. 2) write a python script (batch_sampler.py) that will copy base script according to our requirements, which is running another python script (image_downloader.py) that just downloads the given URLs, in our final setup, we download 20 images per batch, 3) we will have another script (batch_runner.py) that will submit 750 jobs. Since, submitting a job is very fast, we do not need this to be concurrent.

```
(base) ialtun@login1:/projects/academic/erdem/bahadir/DL-project$ ls
Labels.txt       batch_sampler.py     images.zip       scraped-images
batch-script     image-urls           images_1500.zip  scraped-images-legacy
batch_runner.py  image_downloader.py  logs
(base) ialtun@login1:/projects/academic/erdem/bahadir/DL-project$
```

You may see the underlined files in the above image. There are 3 scripts we use to generate 750 batch scripts that each will download 20 images. With this way, we are able download 1GB of image data quicker.

4.3. Conclusion on Data Pipeline

In conclusion, we use BeautifulSoup, which is a Python library that enables parsing HTML and XML files. In addition, we use a systematic approach that will download 15000 images (1500 per class) in a short amount of time using CCR. We first collect the image links and then download the urls we collected using another script. Overall, we use about 1 GB of image data to train models.

5. Methodology

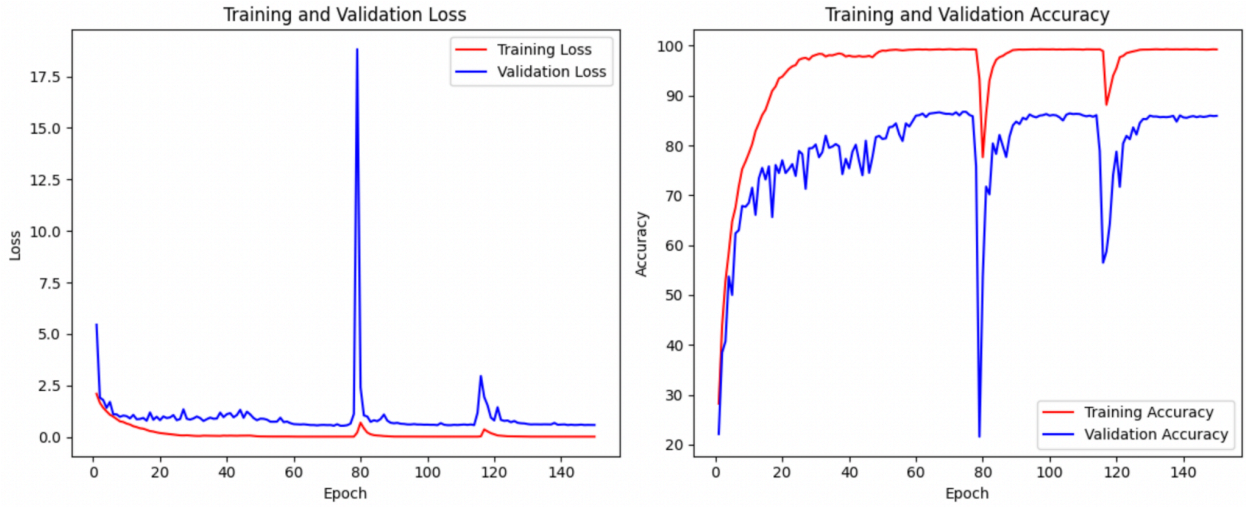
We used CNN networks to classify the given images into one of the classes in our catalogs. We extensively tested multiple existing setups to find the best setup for this task and tried to formulate our setup which matches the evaluation results for the existing setups by testing different hyperparameters and structures. The setup we used include Resnet18_D, Squeeze-and-Excitation Resnet, ResNet-18 with Transformer, ResNet18 including SE blocks with transformer layers, ResNet50 and transformer, and Deep CNN with Transformer.

6. Models

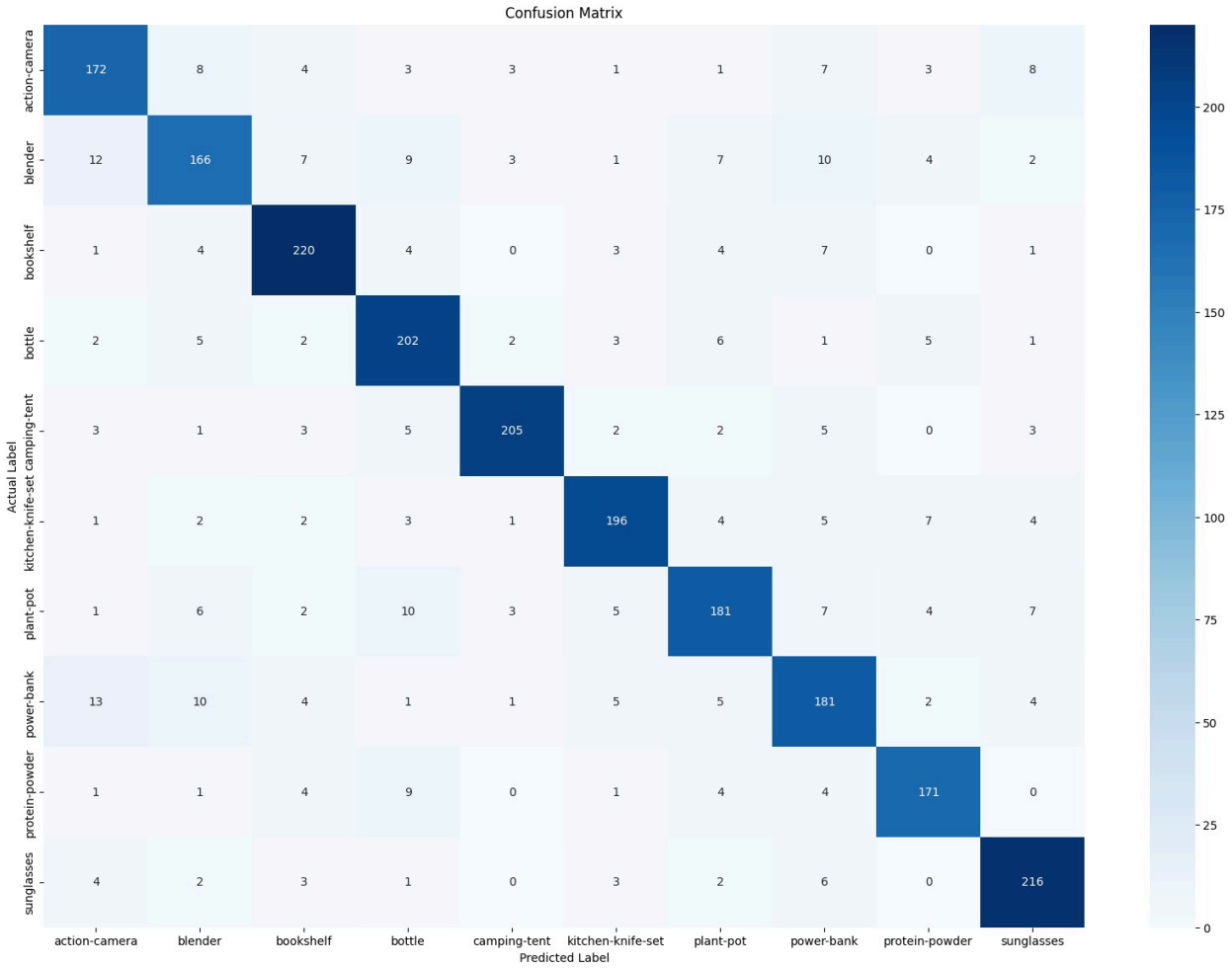
6.1. Resnet18_D

This model is a variant of the ResNet architecture, which includes dropout regularization. It consists of an initial convolutional layer (conv1) with a large kernel size (7x7) and a stride of 2, followed by batch normalization (bn1) and a ReLU activation function. The max pooling layer reduces the spatial dimensions of the feature maps. The model includes four stages, each containing multiple residual blocks that help mitigate the vanishing gradient problem during training. Each block consists of two convolutional layers with batch normalization and ReLU activation, maintaining the same spatial dimensions. The number of channels is increased in each stage, starting from 64 channels in the initial stage to 512 channels in the final stage.
An adaptive average pooling layer (avgpool) reduces the spatial dimensions of the feature maps to a fixed size, followed by dropout regularization to prevent overfitting. Finally, a fully connected layer (fc) outputs the predicted class probabilities based on the number of classes in the dataset.

We ran it for 150 epochs and got Training Accuracy of 99.26% and Validation Accuracy of 85.94%.



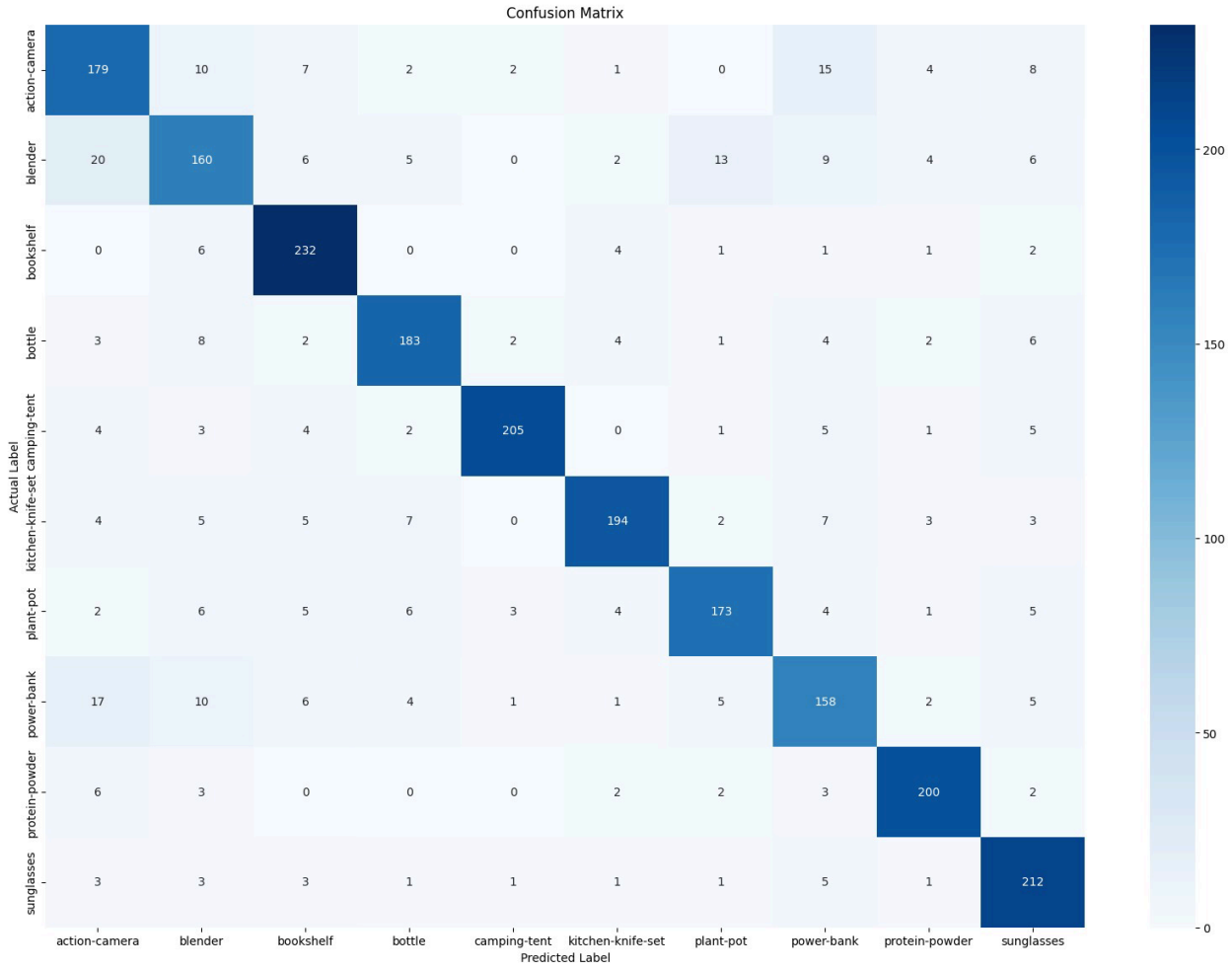
Finally, we got pretty good final result on test data consisting Precision: 0.8524, Recall: 0.8519, F1 Score: 0.8517, and Accuracy: 85.19%. Below is the confusion matrix of our output on the test data.



6.2. Squeeze-and-Excitation Resnet

The model is a Residual Network (ResNet) variant that incorporates a Squeeze-and-Excitation (SE) block for channel-wise feature recalibration. The SE block adaptively recalibrates channel-wise feature responses by leveraging global information. It consists of two main parts: the feature extraction part (convolutional layers followed by batch normalization and ReLU activation) and the SE part (a small network that computes channel-wise scaling factors). The ResidualBlock_SE module is a basic building block for the network, consisting of two convolutional layers with batch normalization and ReLU activation, followed by the SE block and optional downsampling. The SE block recalibrates channel-wise features by modeling interdependencies between channels. Finally, the block adds the identity mapping (shortcut connection) to the output and applies the ReLU activation function, preserving the gradient flow during training.

We ran it for 150 epochs and got Training Accuracy of 99.27% and Validation Accuracy of 84.61%. Finally, we got final result on test data consisting Precision: 0.8471, Recall: 0.8457, F1 Score: 0.8456, Accuracy :84.56, which is fair enough but not better than the first model. Below is the confusion matrix of our output on the test data.



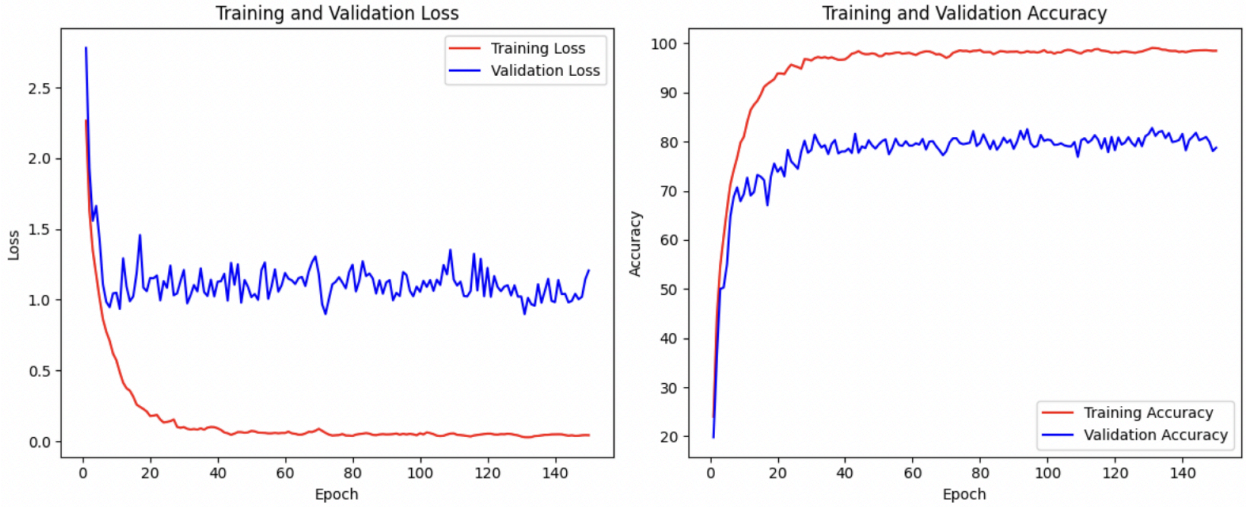
6.3. ResNet-18 with Transformer

The model is a variant of the ResNet-18 architecture combined with a Transformer network. The ResNet-18 part consists of a series of residual blocks, each containing two convolutional layers, with the number of blocks varying per stage (4 stages in total). The convolutional layers are followed by batch normalization and ReLU activation. The model uses max-pooling after the initial convolutional layer to downsample the input.

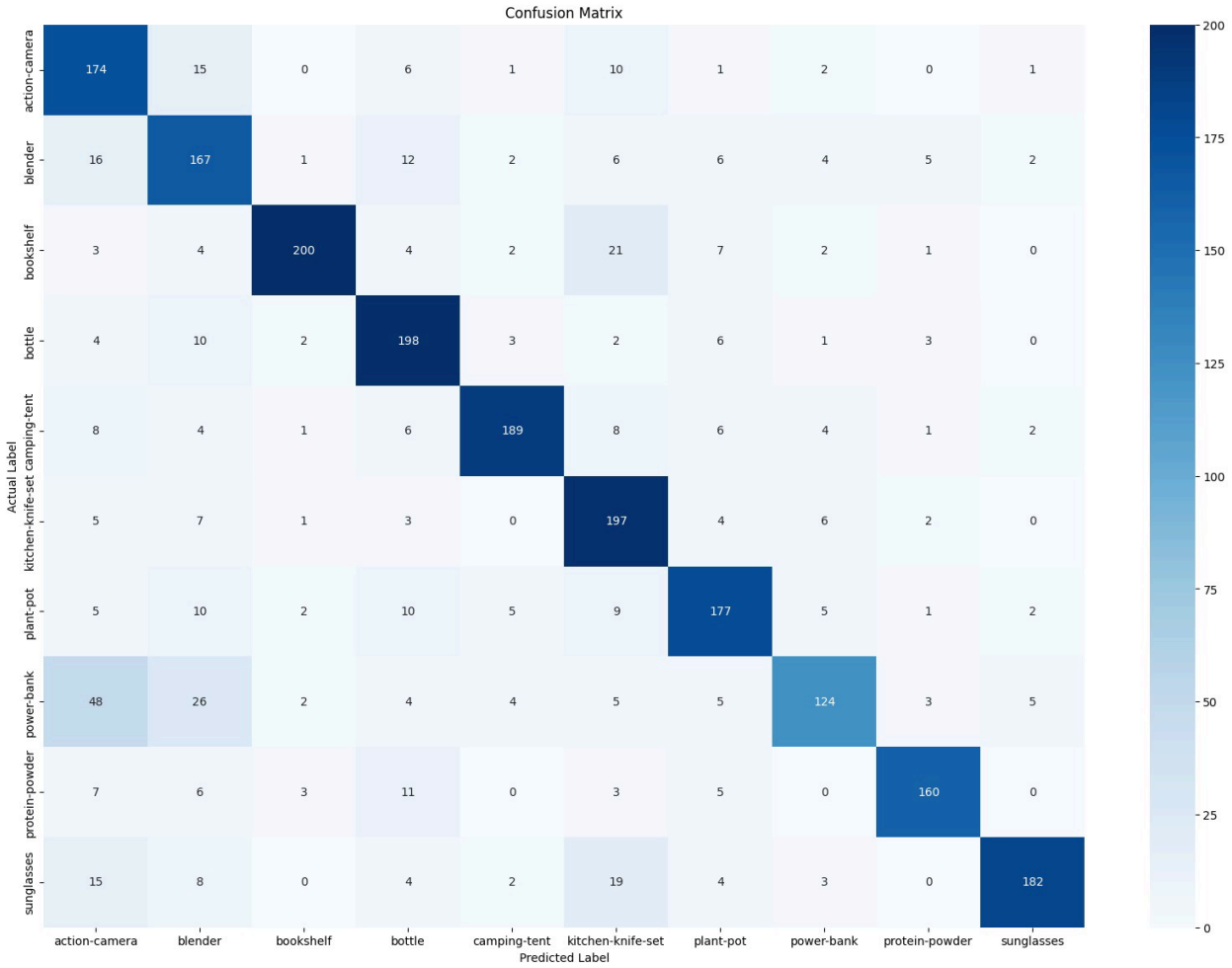
The Transformer part is added after the third stage of the ResNet, and it replaces the final stage of ResNet-18. It consists of a single Transformer block, which includes a positional encoding layer to inject information about the position of tokens in the sequence. The Transformer block employs self-attention mechanisms to capture global dependencies in the input sequence, enhancing the model's ability to understand relationships between different parts of the image.

The final output of the model is obtained by applying an adaptive average pooling layer to reduce the spatial dimensions to 1x1, followed by flattening and a fully connected layer to produce the class predictions. The model is designed to be flexible, allowing for easy inclusion or exclusion of the Transformer part by setting the include_transformer parameter.

We ran it for 150 epochs and got Training Accuracy of 98.50% and Validation Accuracy of 78.76%.



Finally, we got one of our worst final results on test data consisting Precision: 0.8075, Recall: 0.7886, F1 Score: 0.7903, Accuracy: 78.85816235504014%. Below is the confusion matrix of our output on the test data.



6.4. ResNet18 including SE blocks with transformer layers

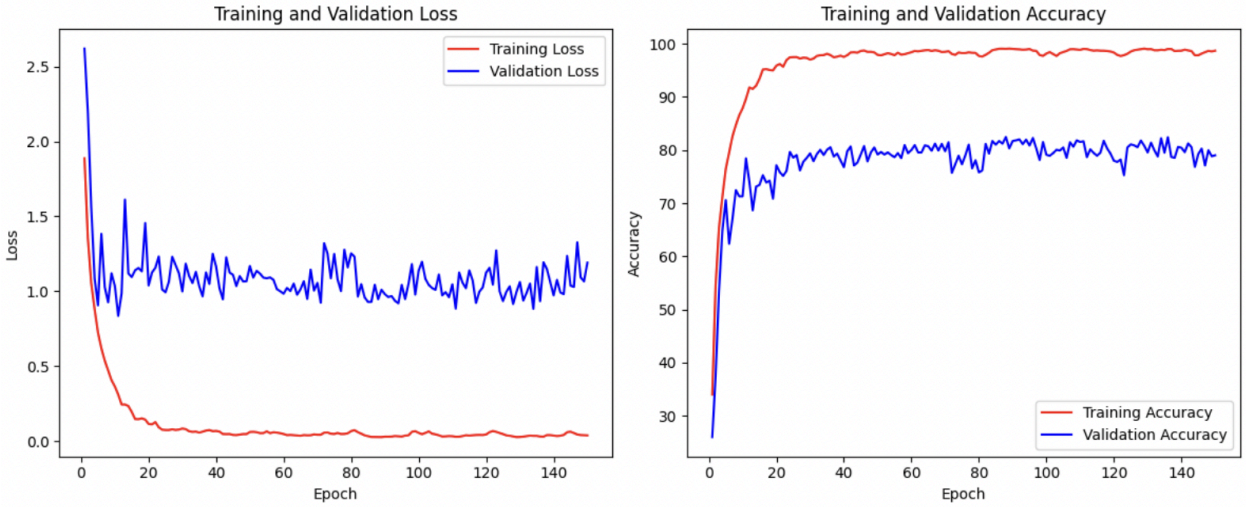
The model is a hybrid architecture that combines ResNet and Transformer components with Squeeze-and-Excitation (SE) block. It begins with a standard ResNet backbone, including a convolutional layer followed by batch normalization, ReLU activation, and max pooling. The ResNet backbone is extended with several residual blocks, each containing two convolutional layers, batch normalization, and ReLU activation, similar to a traditional ResNet.

Additionally, each residual block contains a SE, which adaptively recalibrates the feature responses by explicitly modeling interdependencies between channels. This SE block consists of a global average pooling layer, followed by two fully connected layers with ReLU activation and a sigmoid output, which scales the input features.

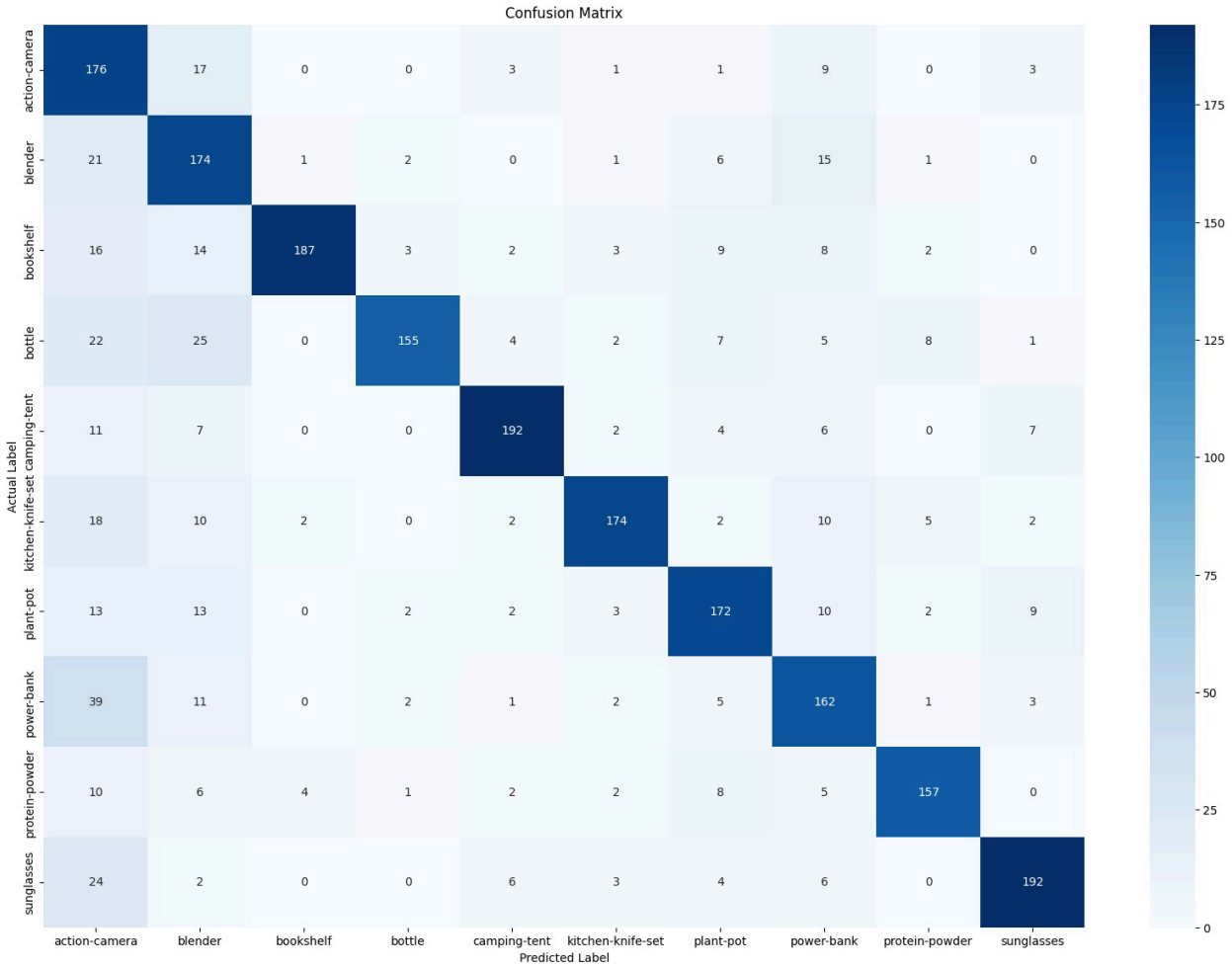
The model's main innovation lies in the inclusion of a Transformer-based component after the ResNet backbone. This Transformer block consists of a TransformerEncoder with multiple TransformerEncoderLayers, each containing a self-attention mechanism and two fully connected layers. The self-attention mechanism allows the model to capture long-range dependencies in the input feature maps.

Finally, the model ends with a global average pooling layer, which reduces the spatial dimensions of the feature maps to a single value per channel, followed by a fully connected layer (fc) that outputs the final class predictions. Overall, this model combines the strengths of ResNet's feature extraction capabilities with the attention mechanism of Transformers for improved performance on tasks requiring long-range dependencies and adaptively recalibrated features.

We ran it for 150 epochs and got Training Accuracy of 98.70% and Validation Accuracy of 79.03%.



Finally, we got final result on test data consisting Precision: 0.8114, Recall: 0.7765, F1 Score: 0.7847, Accuracy:77.65%. Below is the confusion matrix of our output on the test data.



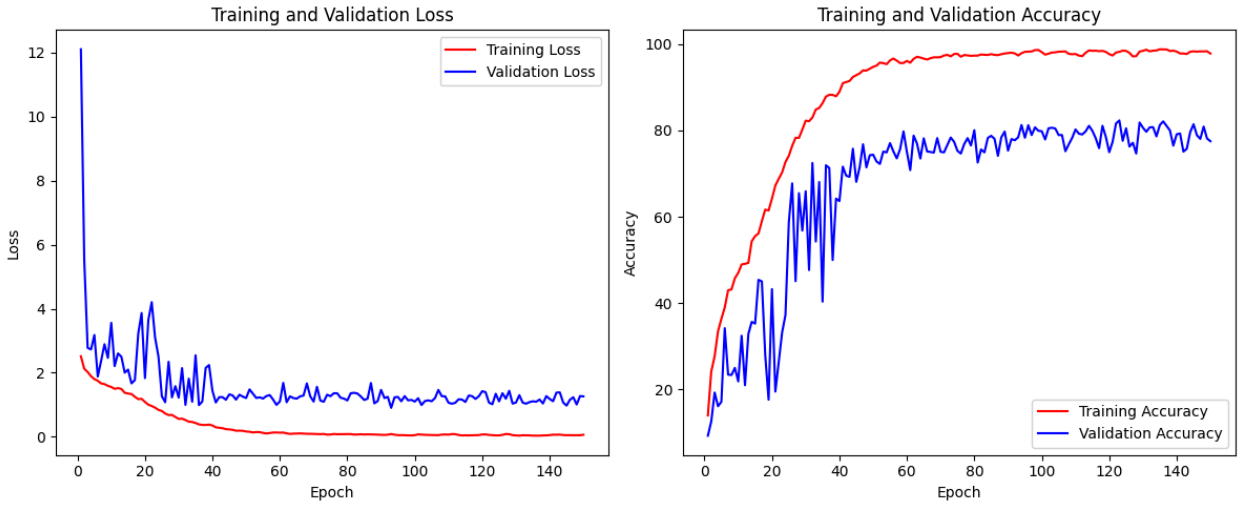
6.5. ResNet50 and transformer

The model is a deep neural network that combines a ResNet50 architecture with a Transformer-based feature encoder. The ResNet50 part consists of multiple layers, each containing residual blocks that help mitigate the vanishing gradient problem in deep networks. These blocks have convolutional layers followed by batch normalization and ReLU activation functions. The number of blocks in each layer varies, with more blocks in later layers to capture increasingly complex features.

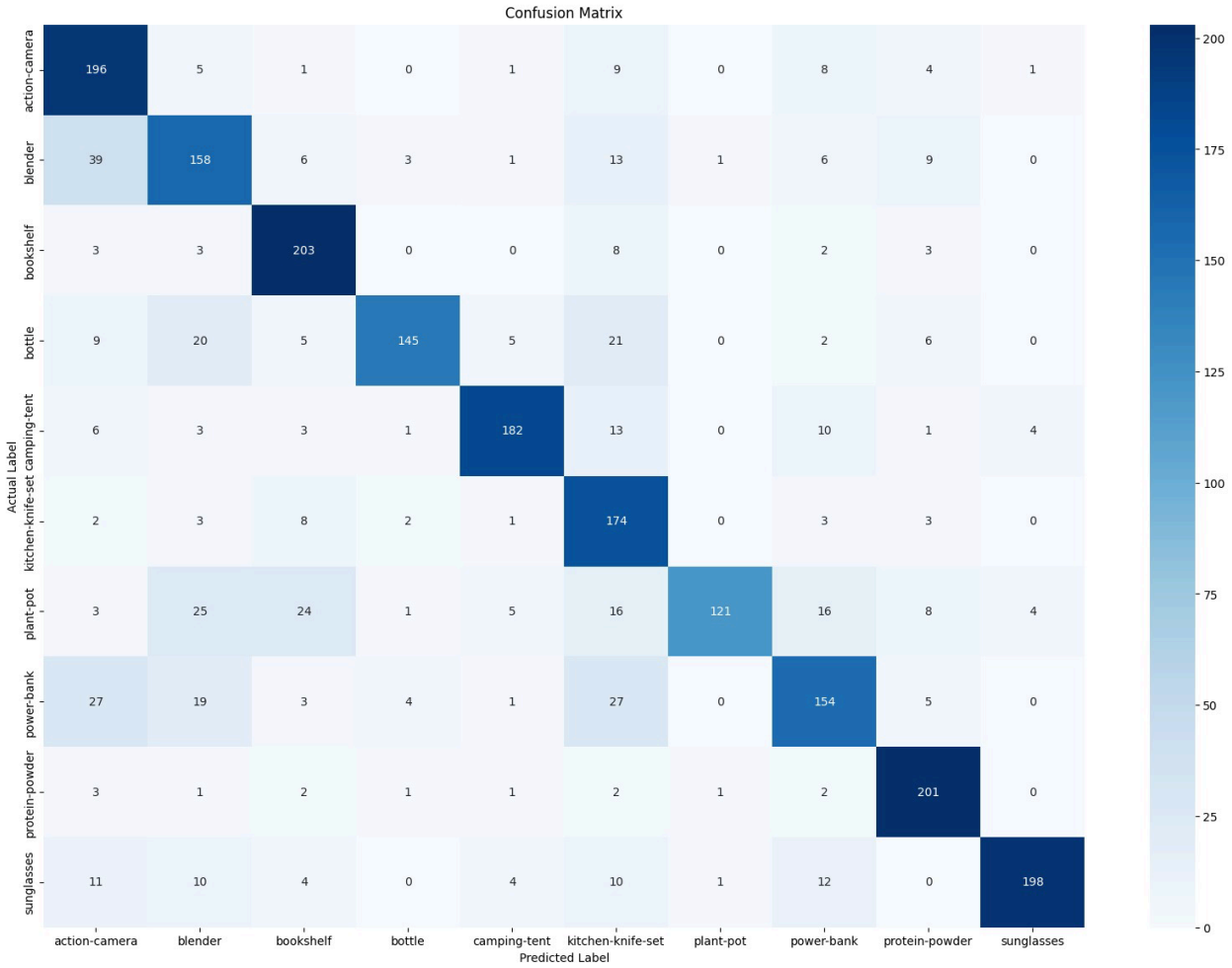
The Transformer part, is used to encode spatial information in the input image. It consists of a stack of TransformerEncoderLayers, each containing multi-head self-attention mechanisms followed by feedforward neural networks. Positional encoding is applied to the input before passing it through the Transformer to preserve positional information.

The model's final output is a linear layer that maps the extracted features to the number of output classes, suitable for classification tasks. The combination of ResNet and Transformer allows the model to capture both local and global features effectively, making it suitable for tasks where spatial information is crucial, such as image classification or segmentation.

We ran it for 150 epochs and got Training Accuracy: 97.82% and Validation Accuracy: 77.51%.



Finally, we got final result on test data consisting Precision: 0.8008, Recall: 0.7725, F1 Score: 0.7726, Accuracy:77.25%. Below is the confusion matrix of our output on the test data.



6.6. Deep CNN with Transformer

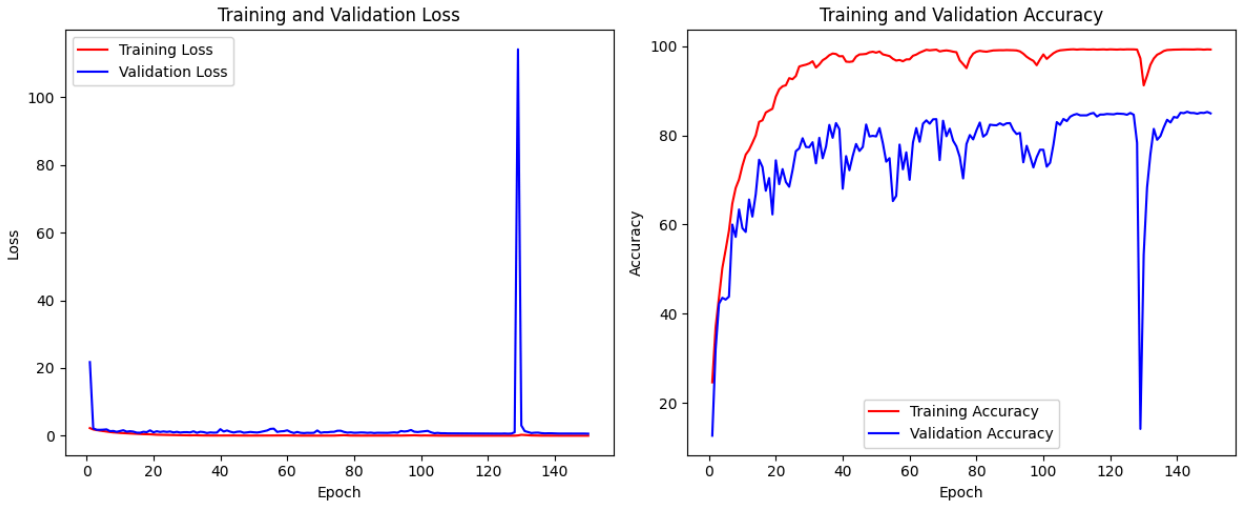
This model is a deep convolutional neural network (CNN) with a transformer block added to enhance its capabilities. The CNN part consists of a series of convolutional layers followed by batch normalization and ReLU activation, which help extract features from the input images. The CNN layers are organized into four stages, each containing a varying number of residual blocks. These residual blocks help the model learn complex patterns by allowing the gradients to flow more easily during training.

The transformer block, typically used in natural language processing tasks, is added to the model to capture long-range dependencies in the image data. It consists of multiple transformer encoder layers, each performing multi-head self-attention over the input features, followed by feedforward neural networks. This enables the model to attend to different parts of the input image and learn contextual relationships between them.

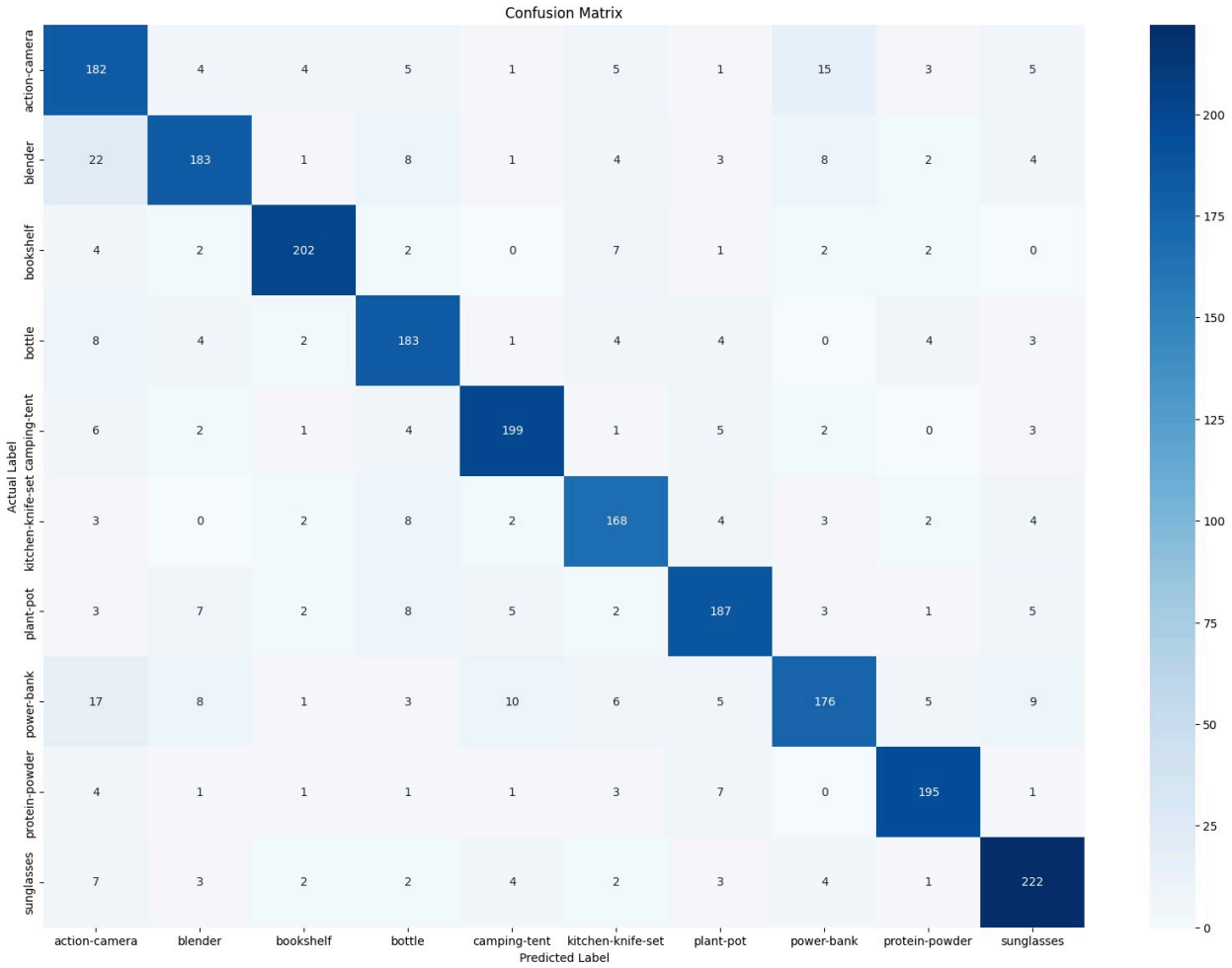
Positional encoding is used to inject information about the position of pixels in the image, which is crucial for the transformer to understand spatial relationships.

The model's final layer is a fully connected layer that produces the output predictions. The inclusion of the transformer block allows the model to capture complex spatial dependencies in the image data, potentially improving its performance on tasks that require understanding of global image context. The use of residual blocks in the CNN part helps mitigate the vanishing gradient problem, enabling training of deeper networks. Overall, this model combines the strengths of CNNs and transformers, making it suitable for tasks that require both local feature extraction and global context understanding in images.

We ran it for 150 epochs and got Training Accuracy: 99.24% and Validation Accuracy: 84.92%.



Finally, we got good final result on test data consisting Precision: 0.8481, Recall: 0.8461, F1 Score: 0.8462, Accuracy:84.61%. However, Resnet18_D is still the best so far. Below is the confusion matrix of our output on the test data.



7. Result

From the models we built, Resnet18_D gave us the best accuracy of 85.19%. ResNet18_D is performing better than the other models due to its simplicity and effectiveness in learning features from the given dataset. The ResNet architecture, with its residual blocks, helps mitigate the vanishing gradient problem, allowing for easier training of deeper networks. The dropout layer in ResNet18_D helps in regularization, preventing overfitting and improving generalization.

The other models, such as Squeeze-and-Excitation ResNet, ResNet-18 with Transformer, and ResNet18 including SE blocks with transformer layers, may have introduced more complexity without significantly improving performance on this particular dataset. The deeper models like ResNet50 and Deep CNN with Transformer may have suffered from overfitting or vanishing gradients, especially with a relatively small dataset like ours.

Additionally, the choice of hyperparameters and the specific characteristics of our dataset also played a significant role in the performance difference between models. It's possible that ResNet18_D's architecture and dropout configuration are particularly well-suited for our dataset, leading to its superior performance compared to the other models.

8. Future Work

In the future we will try to improve our best architecture beside training them with dataset as large as 10,000 per class. With that amount of image per class, we will try to classify more than 100 classes with good accuracy.

9. References

https://pytorch.org/
https://stackoverflow.com/
https://datagen.tech/
https://www.buffalo.edu/ccr.html
https://machinelearningmastery.com/the-transformer-model/
https://www.kaggle.com/