University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

Title: Human Action Recognition in Images Using Convolutional Neural Networks

Dataset:

The dataset we will be using is "Stanford 40 Action Dataset" contains images of humans performing 40 actions. In each image, there are bounding box of the person who is performing the action indicated by the filename of the image. There are 9532 images in total with 180-300 images per action class.

Link: Stanford 40 Actions

Methodology:

Deep Learning Techniques and Models:

- Convolutional Neural network: For image classification task we will use convolutional neural network as the primary model due to its strong performance. As it can effectively capture more complex patterns.
- Architecture: Using architecture like VGGNet, GoogleNet, ResNet, DenseNet to leverage these model's learned features and fine-tune them for action recognition task.

Preprocessing

- Loading: Divided the raw dataset into 40 classes and images are loaded from the specified file paths using the 'CustomImageDataset' class.
- Transformation: A series of transformation are applying to prepare the images for training.
- Resizing: Image are resized to 224*224 pixels, which is a standard input size.
- Normalization: Images are normalized using ([0.485, 0.456,0.406]) values for mean and ([0.229,0.224.0.225]) for standard deviation.

Model used:

1. **VGG Model:** The VGG that is Visual Geometry Group is a type of convolutional neural network which was developed by Visual Geometry Group at the University of Oxford. This model is popular due to its performance and simplicity.

VGG Implementation:

• Input Layer:

The model input image is of shape (224, 224, 3) with 3 color channels (RGB).

• Convolutional Blocks:

The model consists of 5 convolutional blocks. Each block contains:

Block 1:

- 2 convolutional layers with 64 filters each
- ReLU activation
- 1 max-pooling layer

Block 2:

- 2 convolutional layers with 128 filters each
- ReLU activation
- 1 max-pooling layer

Block 3:

- 3 convolutional layers with 256 filters each
- ReLU activation
- 1 max-pooling layer

Block 4:

- 3 convolutional layers with 512 filters each
- ReLU activation
- 1 max-pooling layer

Block 5:

- 3 convolutional layers with 512 filters each
- ReLU activation
- 1 max-pooling layer

• Fully Connected Layers:

The model has three fully connected (dense) layers:

- Two layers with 4096 units each and ReLU activation.
- Each of these layers is followed by a dropout layer with a probability of 0.5 to prevent overfitting.

• Output Layer:

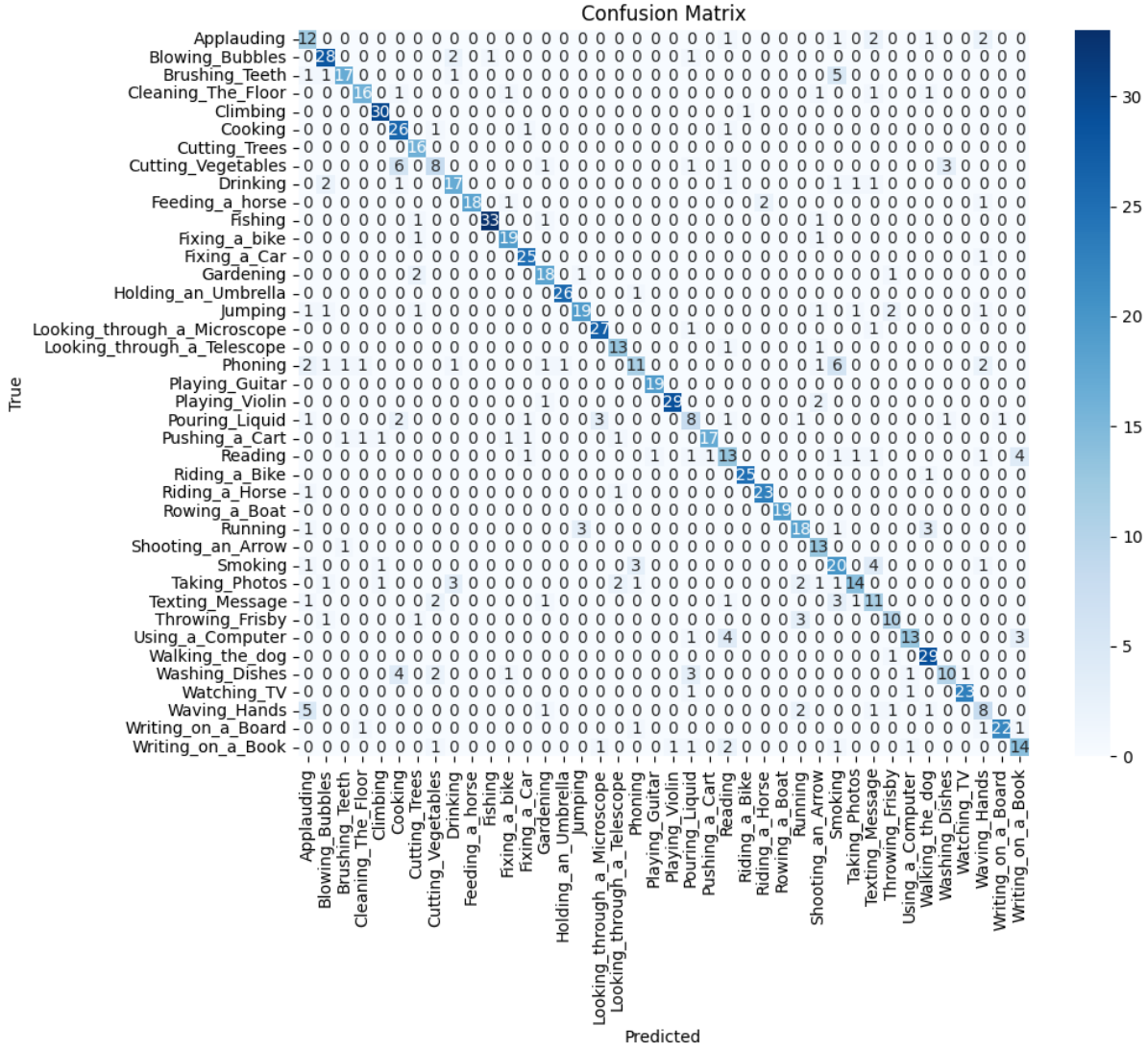
- The model has a final dense layer with `num_classes` units.

Evaluating Model:

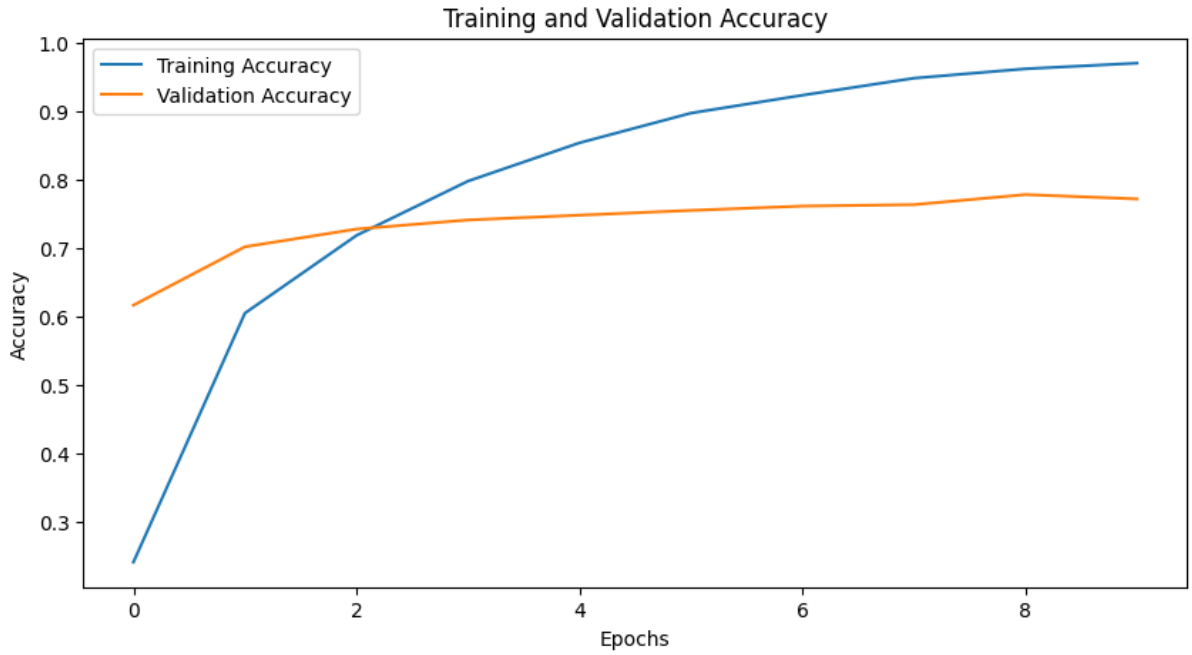
```
VGG Test Accuracy: 0.7725
Test Loss: 0.0874, Test Accuracy: 0.7725
```

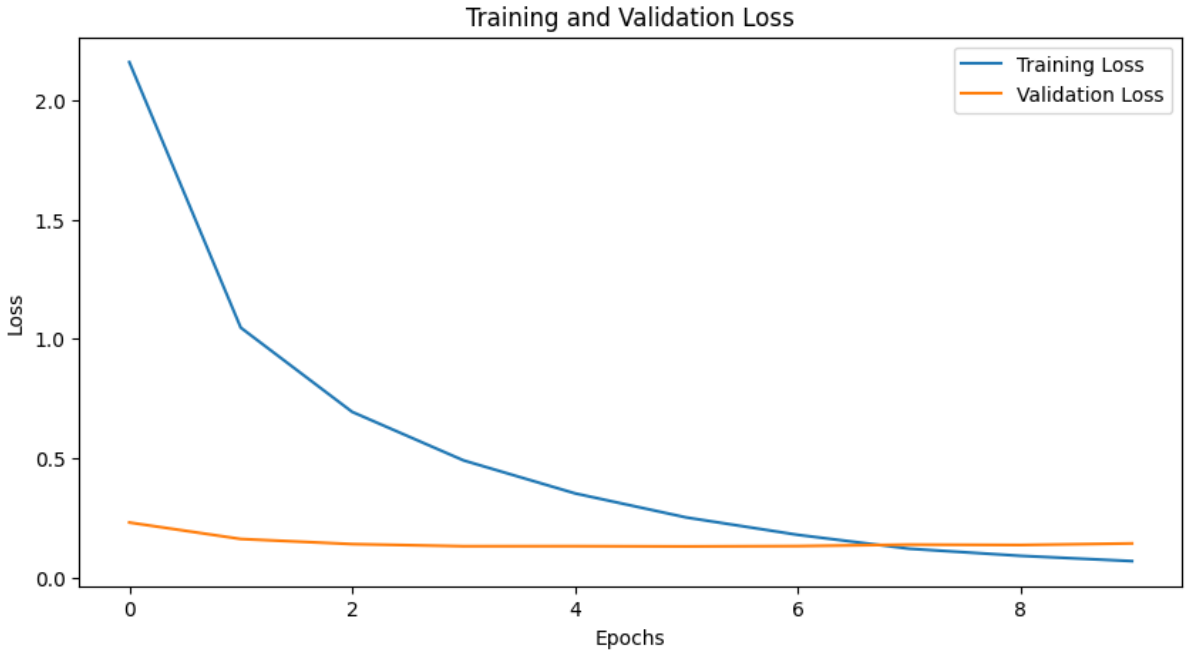
```
Precision: 0.7810, Recall: 0.7725, F1 Score: 0.7702
```

Confusion Matrix:



Plotting Graph:


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epochs
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
2
4
6
8


Training and Validation Loss
Training Loss
Validation Loss
Loss
Epochs
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8

2. **ResNet Model:** The Resnet model that is Residual network, learns residual function with reference to the layer inputs, instead of learning unreferenced functions. This model helps to solve the vanishing/exploding gradient. The model uses skip connection which connects activation of a function by skipping some layers in between. This forms a residual block. And Resent is formed by stacking this residual block.

ResNet Implementation:

• Input Layer:

The model input image is of shape (224, 224, 3) with 3 color channels (RGB).

• Initial Convolutional Layer:

- Convolutional layer with 64 filters, kernel size 7x7, stride 2, padding 3.
- Batch normalization.
- ReLU activation.
- Max-pooling layer with kernel size 3x3, stride 2, padding 1.

• Residual Blocks:

Bottleneck Block:

- Consists of three convolutional layers.
- First layer: 1x1 convolution with `out_channels` filters.
- Second layer: 3x3 convolution with `out_channels` filters.
- Third layer: 1x1 convolution with `out_channels * 4` filters.
- Each convolutional layer is followed by batch normalization and ReLU activation.
- If the input and output dimensions differ, a downsample layer is used to match them.
- The block's final output is the sum of the input (identity) and the output of the convolutional layers, followed by ReLU activation.

Residual Layers: The ResNet architecture consists of four main layers, each containing multiple residual blocks. Each layer outputs feature maps of different dimensions.

Layer 1:

- Contains 3 bottleneck blocks.
- Each block has 64 output channels.
- The first block can have a stride of 1 or more depending on the downsample parameter.

Layer 2:

- Contains 4 bottleneck blocks.
- Each block has 128 output channels.
- The first block uses a stride of 2 for spatial downsampling.

Layer 3:

- Contains 6 bottleneck blocks.
- Each block has 256 output channels.
- The first block uses a stride of 2 for spatial downsampling.

Layer 4:

- Contains 3 bottleneck blocks.
- Each block has 512 output channels.
- The first block uses a stride of 2 for spatial downsampling.

• Adaptive Average Pooling Layer:

- The model includes an adaptive average pooling layer that outputs a fixed size of (1, 1).

• Fully Connected Layer:

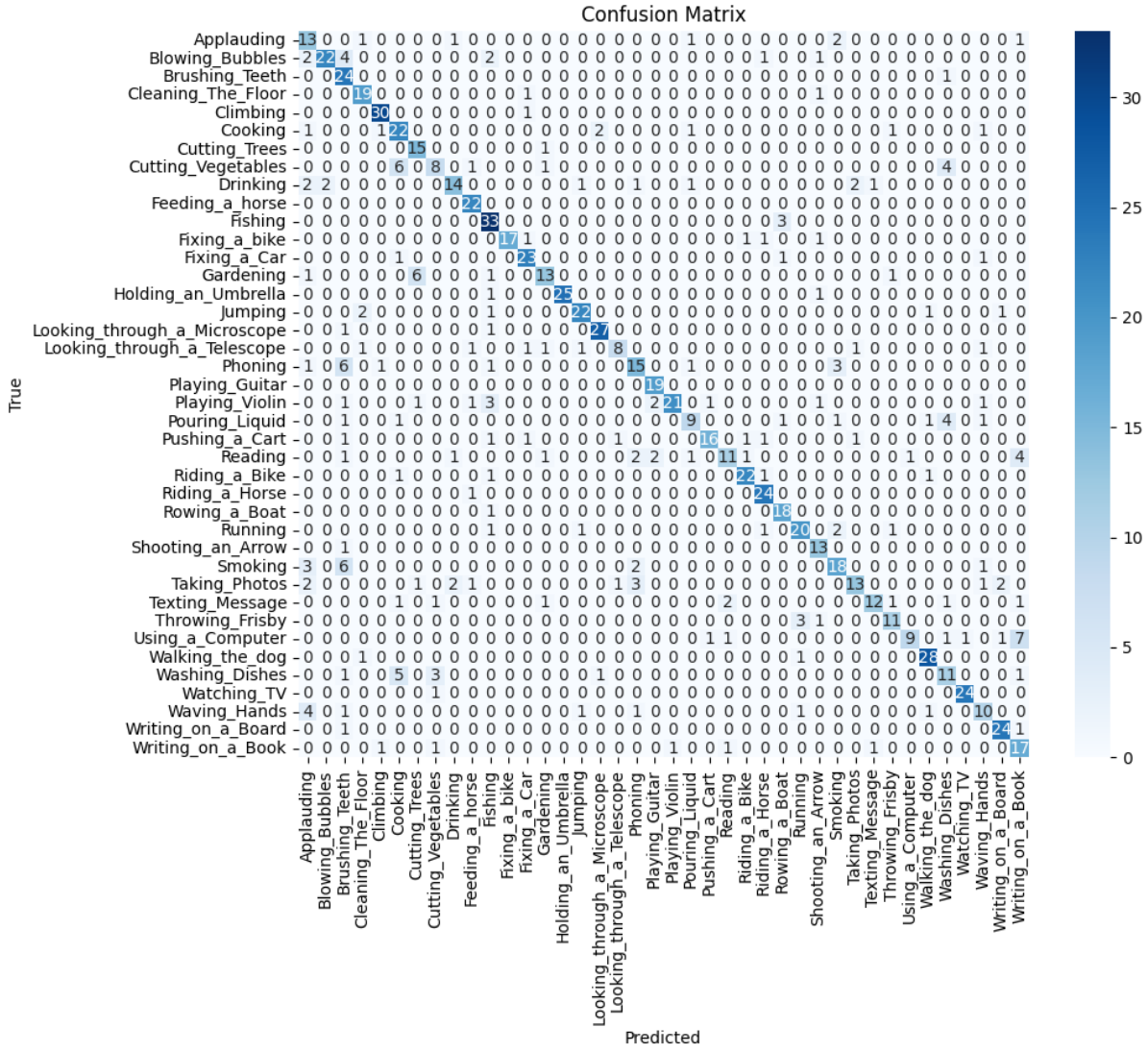
- The model has one fully connected (dense) layer.
- This layer has `512 * block.expansion` units (2048 units when using the Bottleneck block with expansion factor 4) which maps to `num_classes`

Evaluating Model:

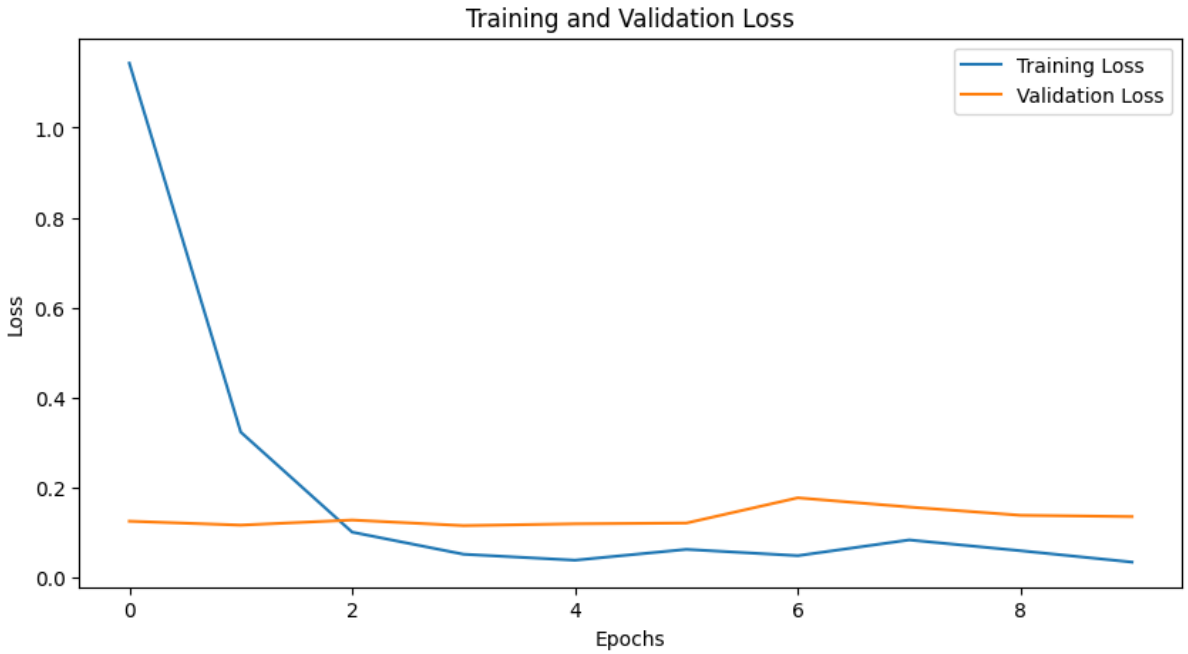
```
ResNet Test Accuracy: 0.7568
Test Loss: 0.0923, Test Accuracy: 0.7568
```

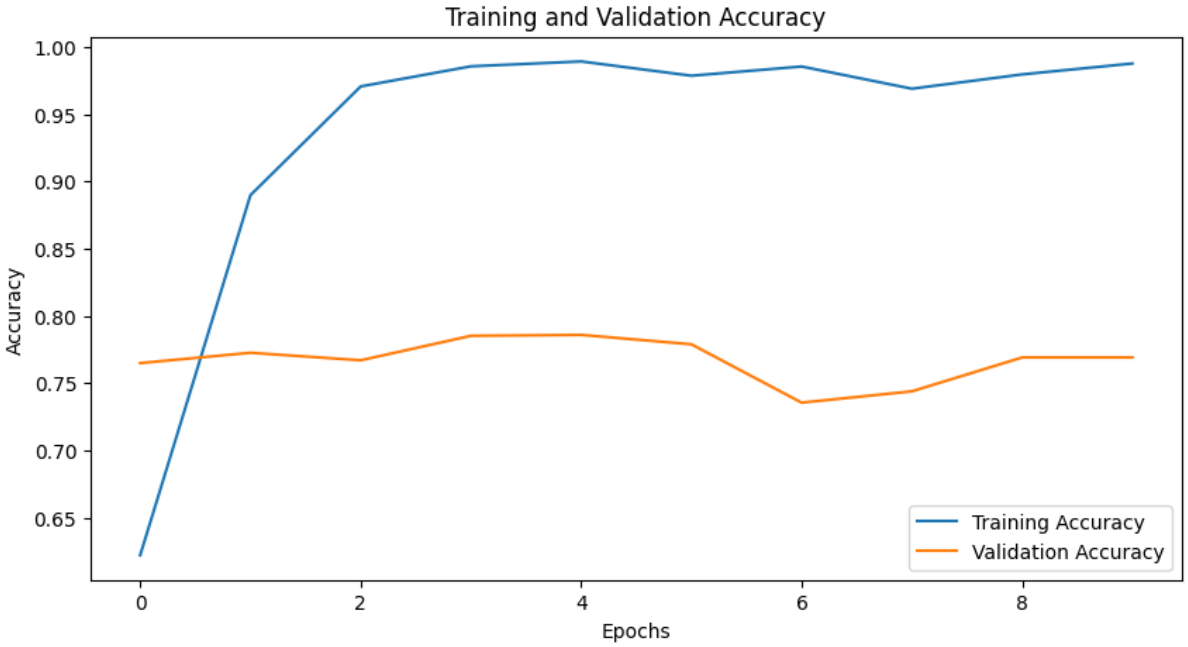
```
Precision: 0.7754, Recall: 0.7568, F1 Score: 0.7529
```

Confusion Matrix



Plotting Graph:


Training and Validation Loss
Training Loss
Validation Loss
Loss
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
Epochs


Training and Validation Accuracy
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
Training Accuracy
Validation Accuracy
0
2
4
6
8
Epochs

3. **GoogleNet Model:** GoogleNet model was proposed in the research paper "Going Deeper with Convolutions" in 2014 by Google. This model uses 1*1 convolution and global average pooling which enables deeper architecture. This model is based on inception architecture.

GoogleNet Implementation:

• Input Layer:

The model input image is of shape (224, 224, 3) with 3 color channels (RGB).

• Initial Convolutional Layers:

- **Conv1:** Convolutional layer with 64 filters, kernel size 7x7, stride 2, padding 3.
- **MaxPool1:** Max-pooling layer with kernel size 3x3, stride 2.
- **Conv2:** Convolutional layer with 64 filters, kernel size 1x1.
- **Conv3:** Convolutional layer with 192 filters, kernel size 3x3, padding 1.
- **MaxPool2:** Max-pooling layer with kernel size 3x3, stride 2.

• Inception Modules:

The model uses Inception modules which consist of multiple branches with convolutional filters of different sizes, concatenated along the channel dimension.

Inception Module Structure:

- **Branch 1:** 1x1 convolutional layer.
- **Branch 2:** 1x1 convolutional layer followed by a 3x3 convolutional layer.
- **Branch 3:** 1x1 convolutional layer followed by a 5x5 convolutional layer.
- **Branch 4:** 3x3 max-pooling layer followed by a 1x1 convolutional layer.

Inception Blocks:

- **Inception3a:** Takes input with 192 channels, produces an output with 256 channels.
  - 1x1 conv: 64 channels
  - 3x3 conv: 128 channels (96 reduced to 128)
  - 5x5 conv: 32 channels (16 reduced to 32)
  - Pool: 32 channels
- **Inception3b:** Takes input with 256 channels, produces an output with 480 channels.
  - 1x1 conv: 128 channels
  - 3x3 conv: 192 channels (128 reduced to 192)
  - 5x5 conv: 96 channels (32 reduced to 96)
  - Pool: 64 channels
- **Inception4a:** Takes input with 480 channels, produces an output with 512 channels.
  - 1x1 conv: 192 channels
  - 3x3 conv: 208 channels (96 reduced to 208)
  - 5x5 conv: 48 channels (16 reduced to 48)
  - Pool: 64 channels
- **Inception4b:** Takes input with 512 channels, produces an output with 512 channels.
  - 1x1 conv: 160 channels
  - 3x3 conv: 224 channels (112 reduced to 224)

    - 5x5 conv: 64 channels (24 reduced to 64)
    - Pool: 64 channels
- **Inception4c:** Takes input with 512 channels, produces an output with 512 channels.
    - 1x1 conv: 128 channels
    - 3x3 conv: 256 channels (128 reduced to 256)
    - 5x5 conv: 64 channels (24 reduced to 64)
    - Pool: 64 channels
- **Inception4d:** Takes input with 512 channels, produces an output with 528 channels.
    - 1x1 conv: 112 channels
    - 3x3 conv: 288 channels (144 reduced to 288)
    - 5x5 conv: 64 channels (32 reduced to 64)
    - Pool: 64 channels
- **Inception4e:** Takes input with 528 channels, produces an output with 832 channels.
    - 1x1 conv: 256 channels
    - 3x3 conv: 320 channels (160 reduced to 320)
    - 5x5 conv: 128 channels (32 reduced to 128)
    - Pool: 128 channels
- **Inception5a:** Takes input with 832 channels, produces an output with 832 channels.
    - 1x1 conv: 256 channels
    - 3x3 conv: 320 channels (160 reduced to 320)
    - 5x5 conv: 128 channels (32 reduced to 128)
    - Pool: 128 channels
- **Inception5b:** Takes input with 832 channels, produces an output with 1024 channels.
    - 1x1 conv: 384 channels
    - 3x3 conv: 384 channels (192 reduced to 384)
    - 5x5 conv: 128 channels (48 reduced to 128)
    - Pool: 128 channels

• Pooling and Dropout Layers:

- **MaxPool3:** Max-pooling layer with kernel size 3x3, stride 2.
- **MaxPool4:** Max-pooling layer with kernel size 2x2, stride 2.
- **AvgPool:** Adaptive average pooling layer that outputs a fixed size of (1, 1).
- **Dropout:** Dropout layer with a probability of 0.4 to prevent overfitting.

• Fully Connected Layer:

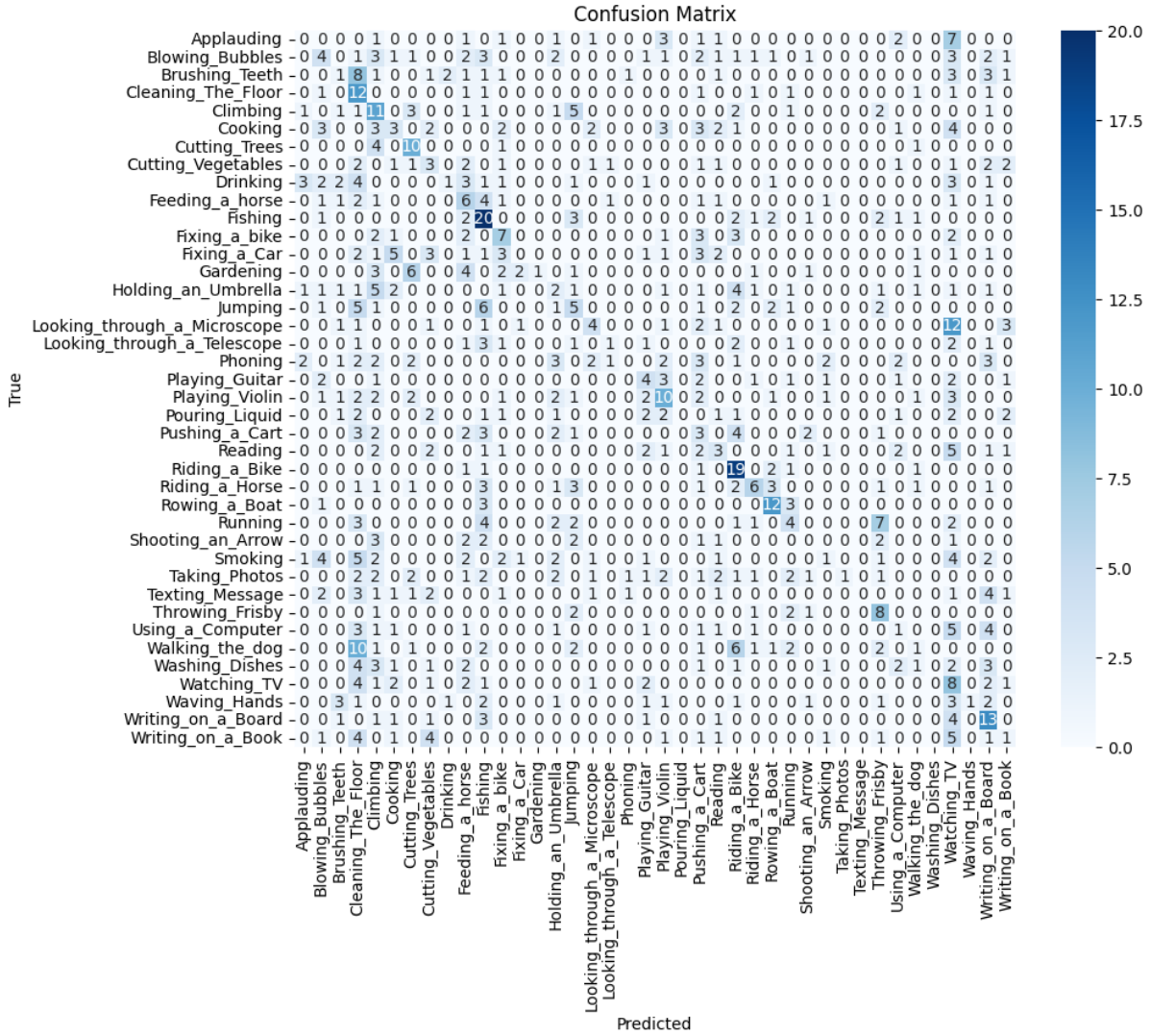
- The model has one fully connected (dense) layer with 1024 units, which maps to num_classes (in this case, 40).

Evaluating Model:

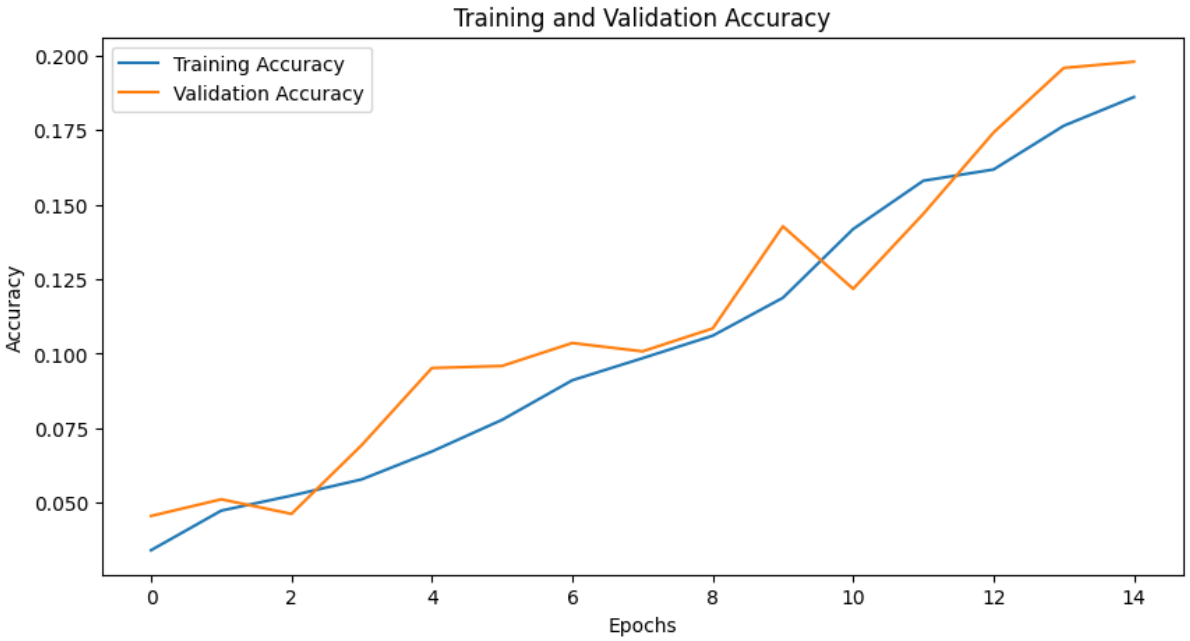
```
GoogleNet Test Accuracy: 0.1960
Test Loss: 0.2978, Test Accuracy: 0.1960
```

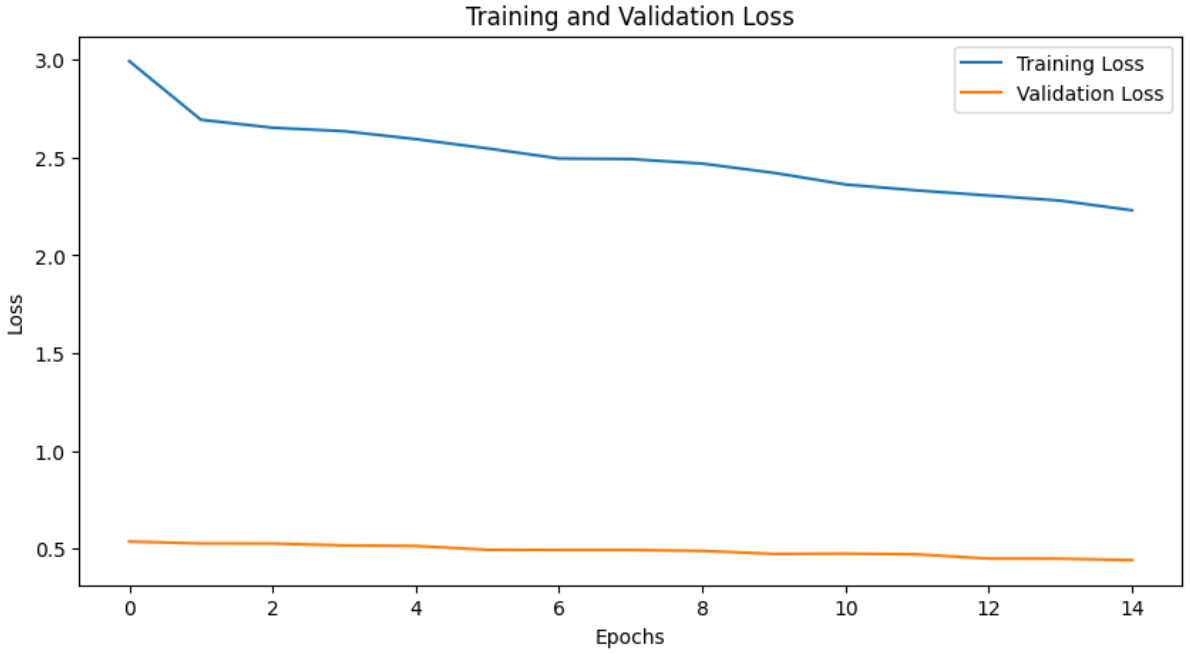
```
Precision: 0.2204, Recall: 0.1960, F1 Score: 0.1597
```

Confusion Matrix



Plotting Graph:


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0
2
4
6
8
10
12
14
Epochs


Training and Validation Loss
Training Loss
Validation Loss
Loss
3.0
2.5
2.0
1.5
1.0
0.5
0
2
4
6
8
10
12
14
Epochs

4. **DenseNet Model:** DenseNet is a type of convolutional neural network which utlises dense connections between layers, through dense blocks which connects all layers directly with each other. In this model concatenation is used where each layer receives collective knowledge from all preceding layers.

DenseNet Implementation:

• Input Layer:

The model input image is of shape (224, 224, 3) with 3 color channels (RGB).

• Initial Convolutional Layer:

- **Conv1:** Convolutional layer with 64 filters, kernel size 7x7, stride 2, padding 3.
- **BN1:** Batch normalization layer.
- **ReLU1:** ReLU activation.
- **MaxPool1:** Max-pooling layer with kernel size 3x3, stride 2, padding 1.

• Dense Blocks and Transition Layers:

The model consists of multiple dense blocks, each followed by a transition layer (except the last block).

Dense Block Structure:

- **Bottleneck Layer:** Each bottleneck layer in a dense block contains:
  - Batch normalization.
  - 1x1 convolutional layer to reduce the number of input channels.
  - Batch normalization.
  - 3x3 convolutional layer.
  - Concatenation of input and output feature maps.

Transition Layer Structure:

- Batch normalization.
- 1x1 convolutional layer to reduce the number of channels.
- Average pooling layer.

Dense Blocks:

- **Dense Block 1:** Contains 6 bottleneck layers. Input channels: 64, Growth rate: 32.
  - Output channels: 256 (64 + 6 * 32).
- **Transition 1:** Reduces the number of channels from 256 to 128.
- **Dense Block 2:** Contains 12 bottleneck layers. Input channels: 128, Growth rate: 32.
  - Output channels: 512 (128 + 12 * 32).
- **Transition 2:** Reduces the number of channels from 512 to 256.
- **Dense Block 3:** Contains 24 bottleneck layers. Input channels: 256, Growth rate: 32.
  - Output channels: 1024 (256 + 24 * 32).
- **Transition 3:** Reduces the number of channels from 1024 to 512.
- **Dense Block 4:** Contains 16 bottleneck layers. Input channels: 512, Growth rate: 32.

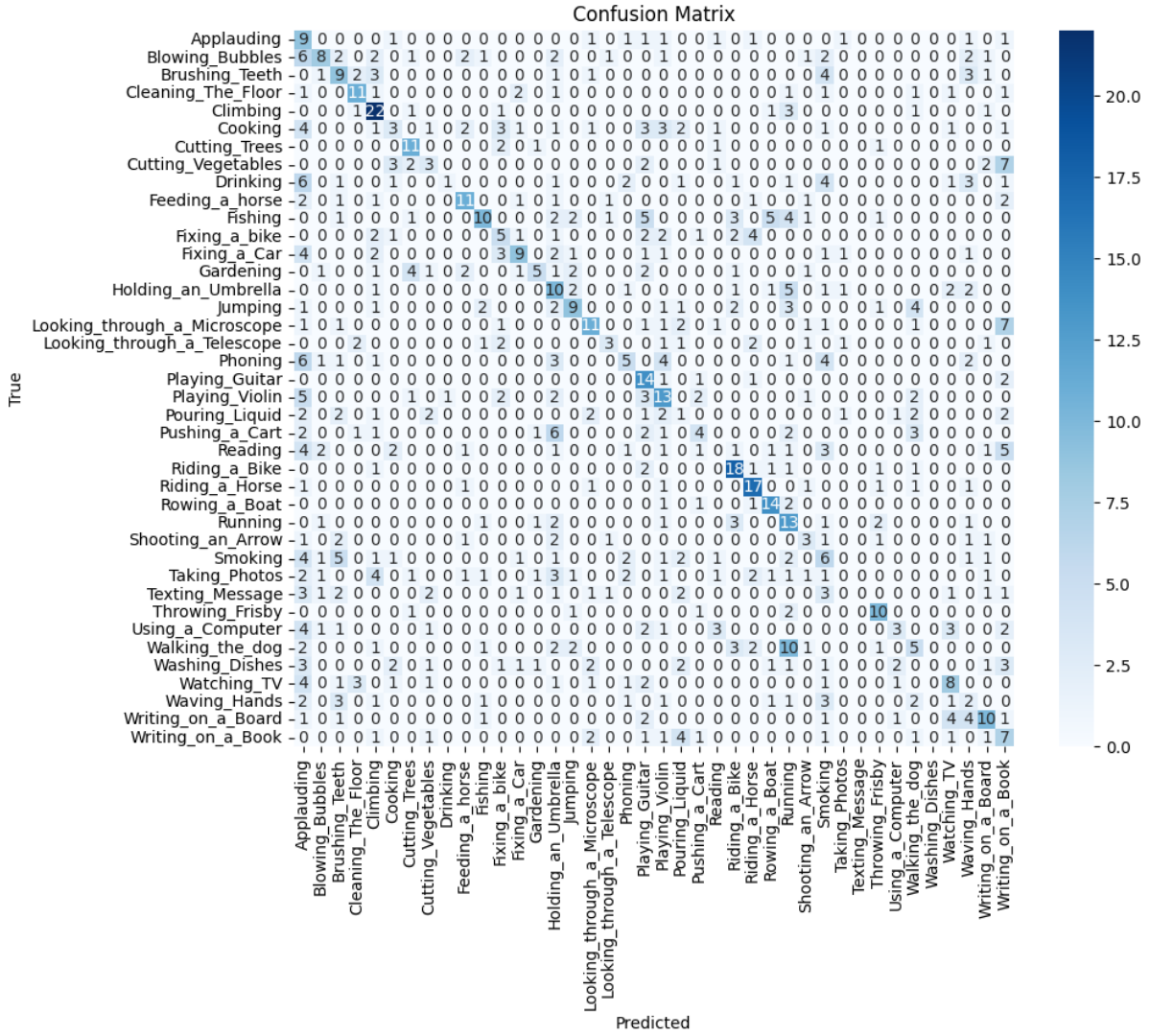
- Output channels: 1024 (512 + 16 * 32).

Evaluating Model:

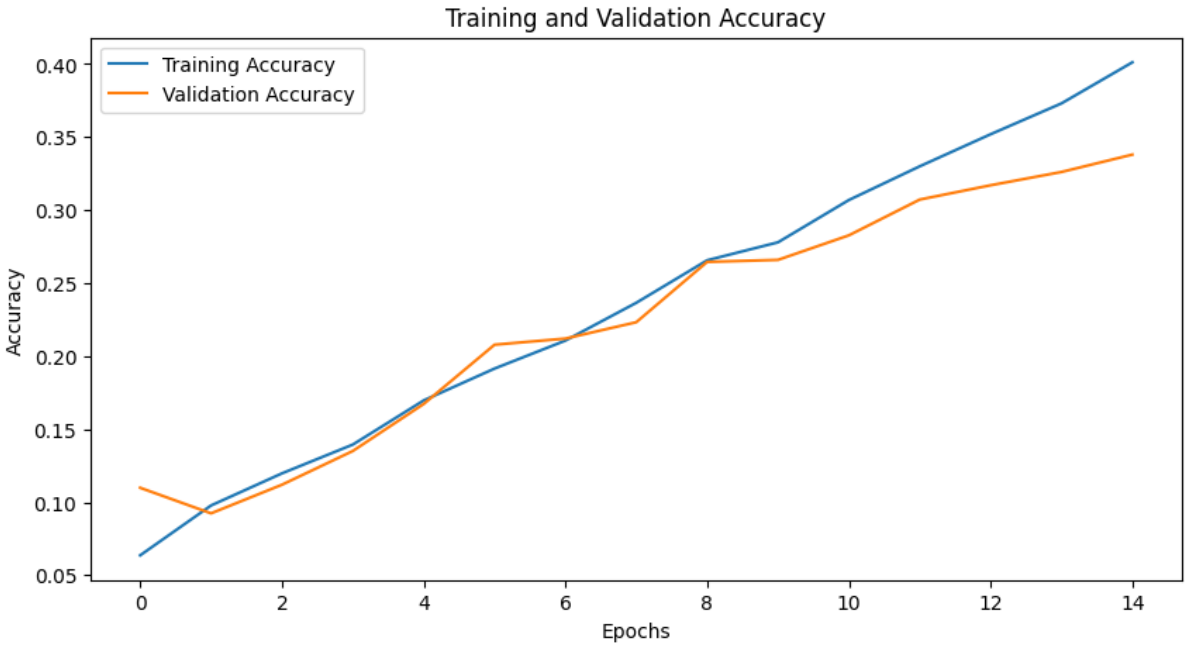
```
DenseNet Test Accuracy: 0.3176
Test Loss: 0.2473, Test Accuracy: 0.3176
```

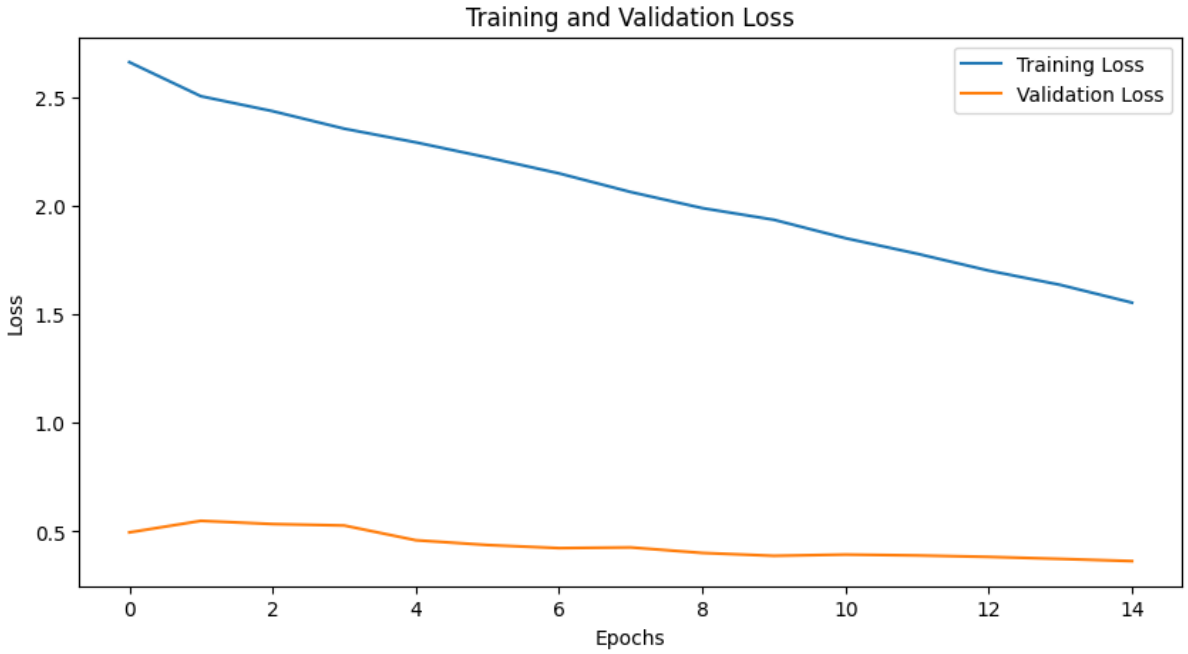
```
Precision: 0.3217, Recall: 0.3176, F1 Score: 0.2976
```

Confusion Matrix



Plotting Graph:





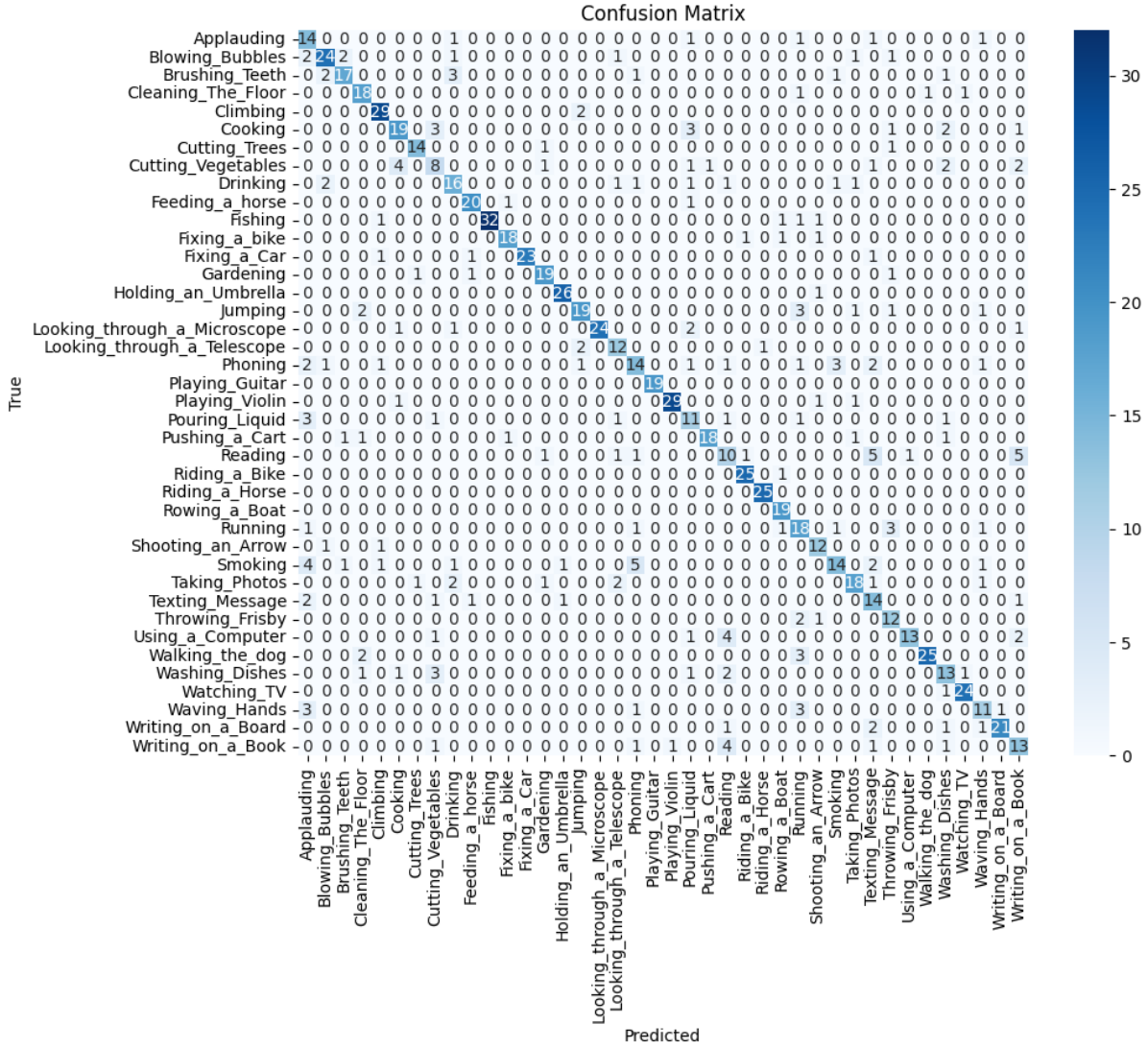
Pretrained GoogleNet:

Evaluating Model:

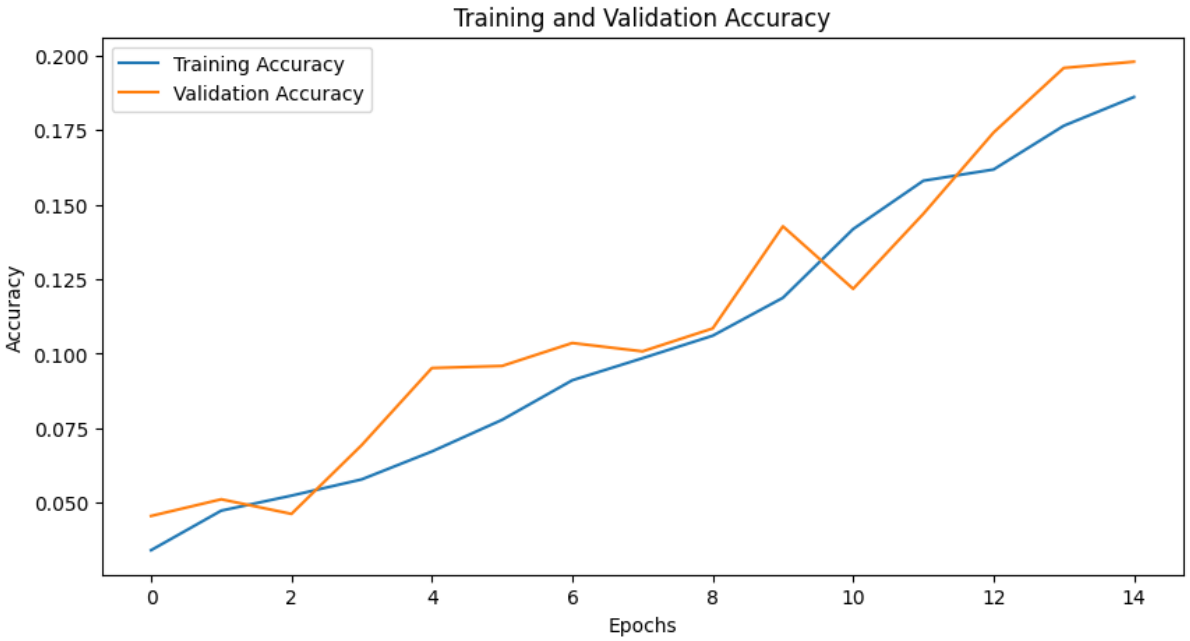
```
Pretrained GoogleNet Test Accuracy: 0.7652
Test Loss: 0.0788, Test Accuracy: 0.7652
```

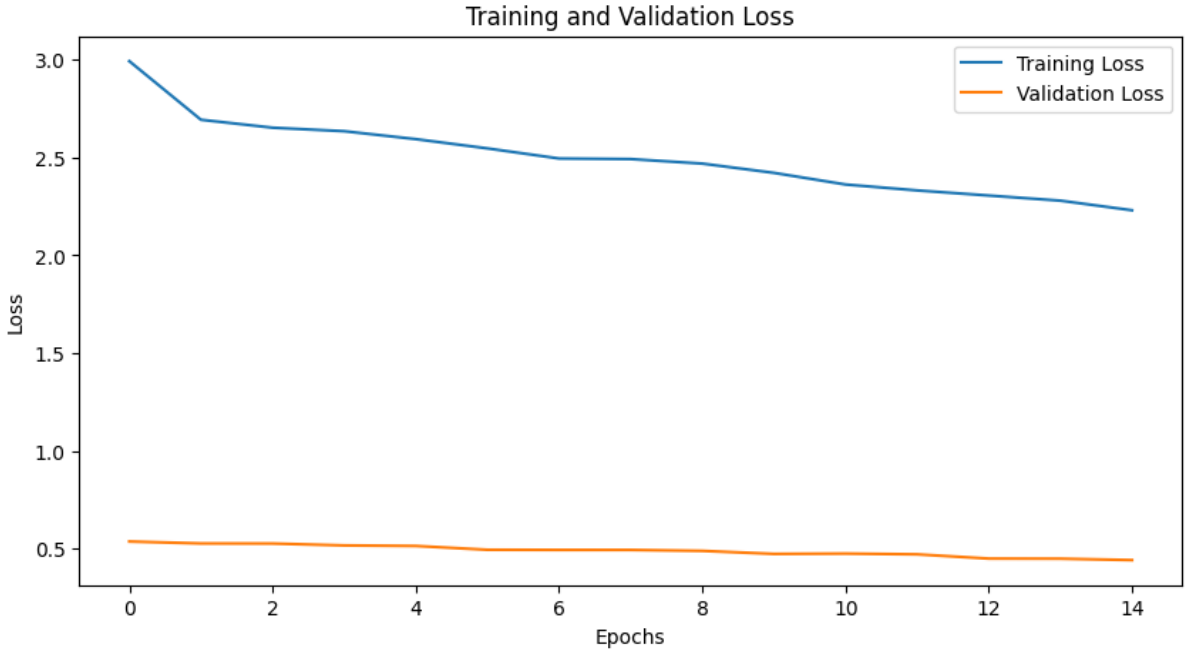
```
Precision: 0.7814, Recall: 0.7652, F1 Score: 0.7680
```

Confusion Matrix



Plotting Graph:





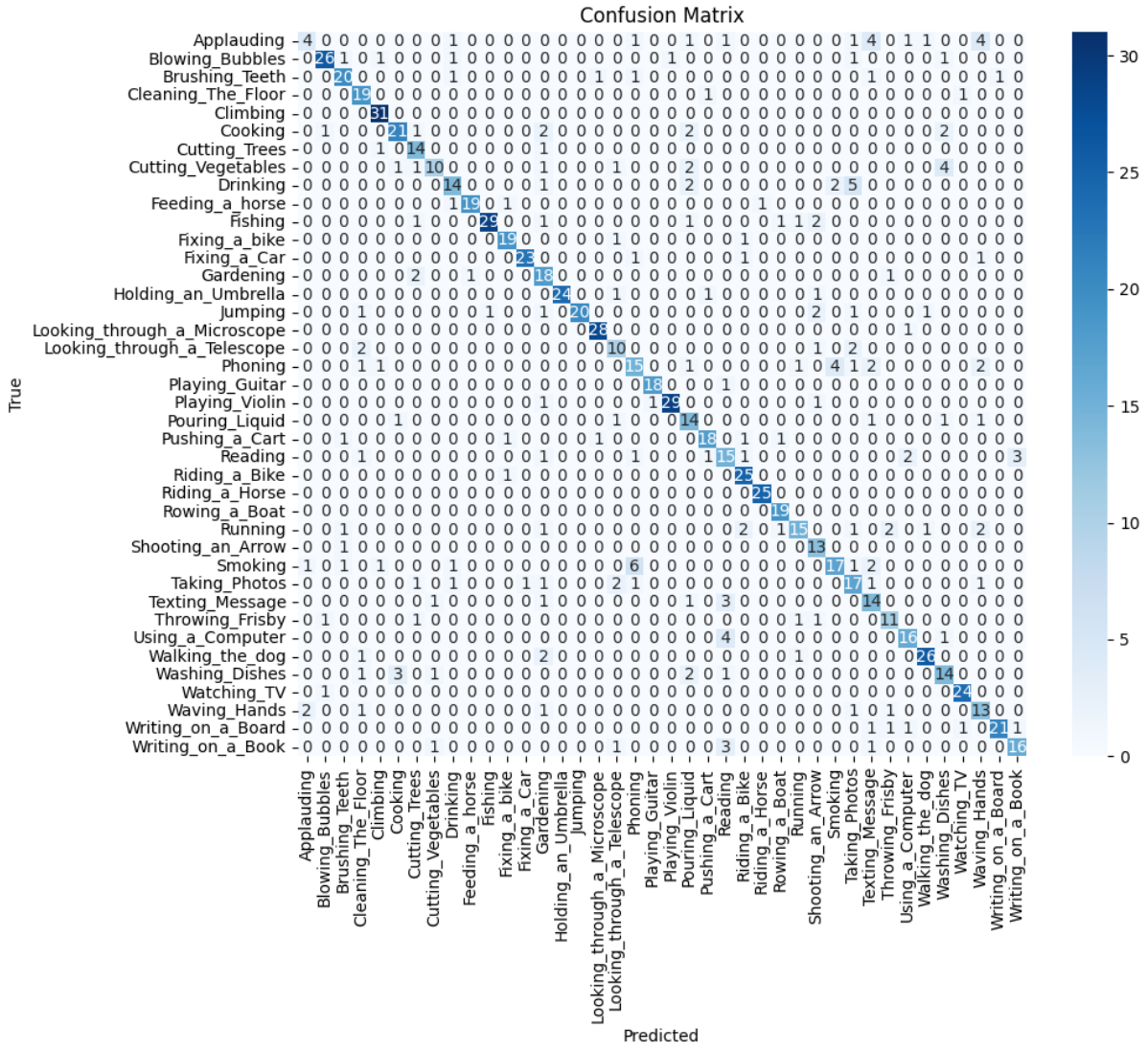
Pretrained DenseNet:

Evaluating Model:

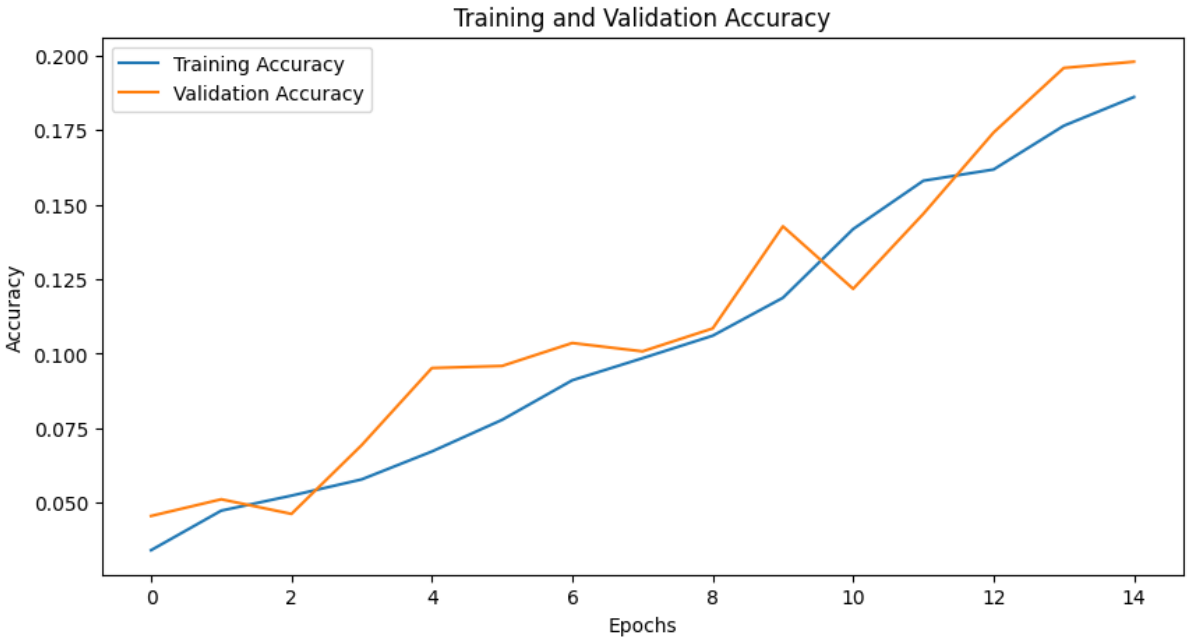
```
Pretrained DenseNet Test Accuracy: 0.7799
Test Loss: 0.0874, Test Accuracy: 0.7799
```

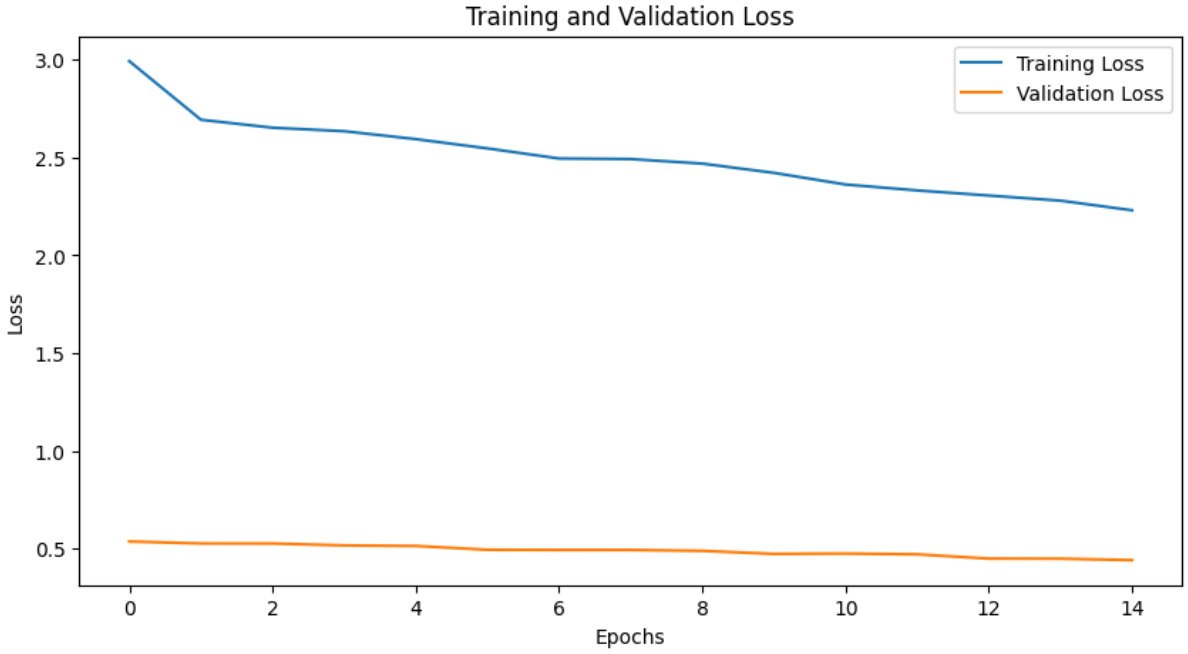
```
Precision: 0.7934, Recall: 0.7799, F1 Score: 0.7790
```

Confusion Matrix



Plotting Graph:





Deployment:

- **Data Preparation:** Images of various human actions are loaded from specified directory which is labelled and pre-processed. This data is split into training, validation and test sets.
- **Model Used:** ResNet model is used and trained using the training data and evaluated on validation data over epochs.
- **Visualization and Interaction:** Streamlit is used for interactive web application which displays training progress which includes loss and accuracy epochs.This application will classify randomly selected images from the training set and display the predicted class.

Result:

Human Action Recognition using CNN Models - ResNet

Epoch [1/8], Train Loss: 1.5924, Train Acc: 0.4195, Val Loss: 0.3202, Val Acc: 0.4255

Epoch [2/8], Train Loss: 1.0118, Train Acc: 0.5973, Val Loss: 0.2803, Val Acc: 0.4997

Epoch [3/8], Train Loss: 0.6954, Train Acc: 0.7226, Val Loss: 0.2572, Val Acc: 0.5556

Epoch [4/8], Train Loss: 0.4140, Train Acc: 0.8279, Val Loss: 0.2746, Val Acc: 0.5297

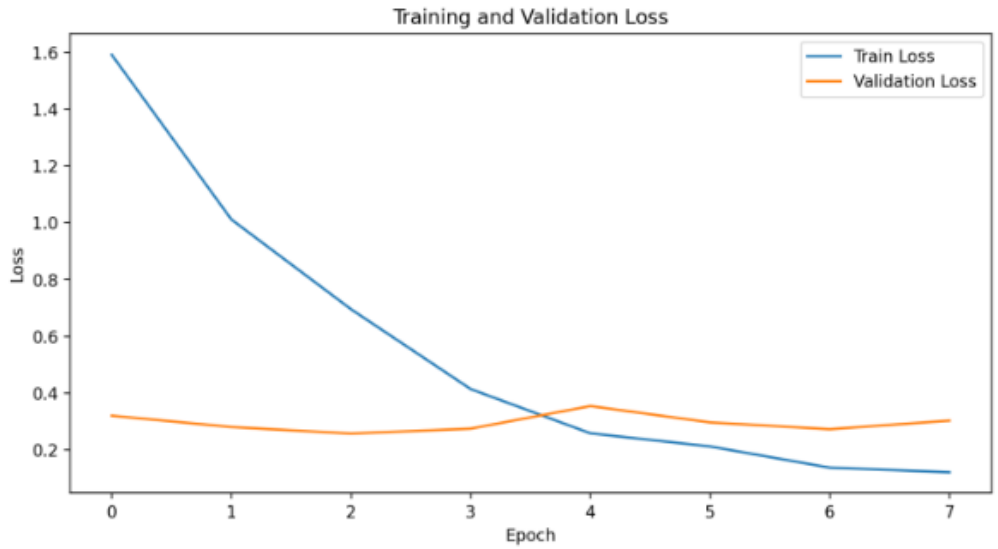
Epoch [5/8], Train Loss: 0.2581, Train Acc: 0.8941, Val Loss: 0.3542, Val Acc: 0.5171

Epoch [6/8], Train Loss: 0.2119, Train Acc: 0.9148, Val Loss: 0.2966, Val Acc: 0.5836

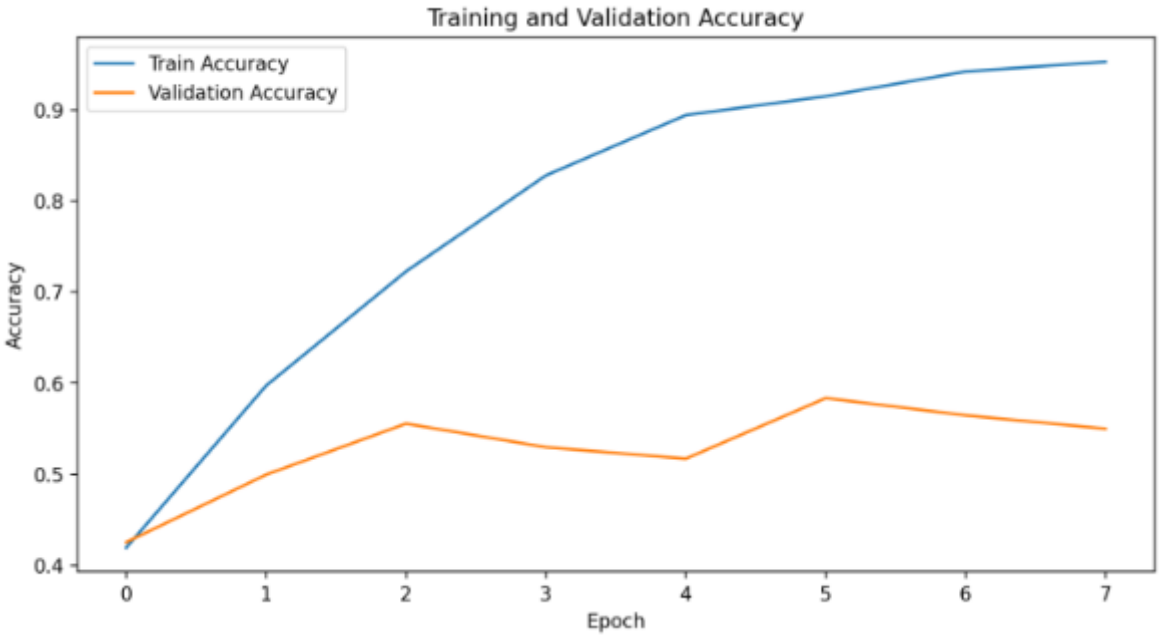
Epoch [7/8], Train Loss: 0.1371, Train Acc: 0.9419, Val Loss: 0.2727, Val Acc: 0.5647

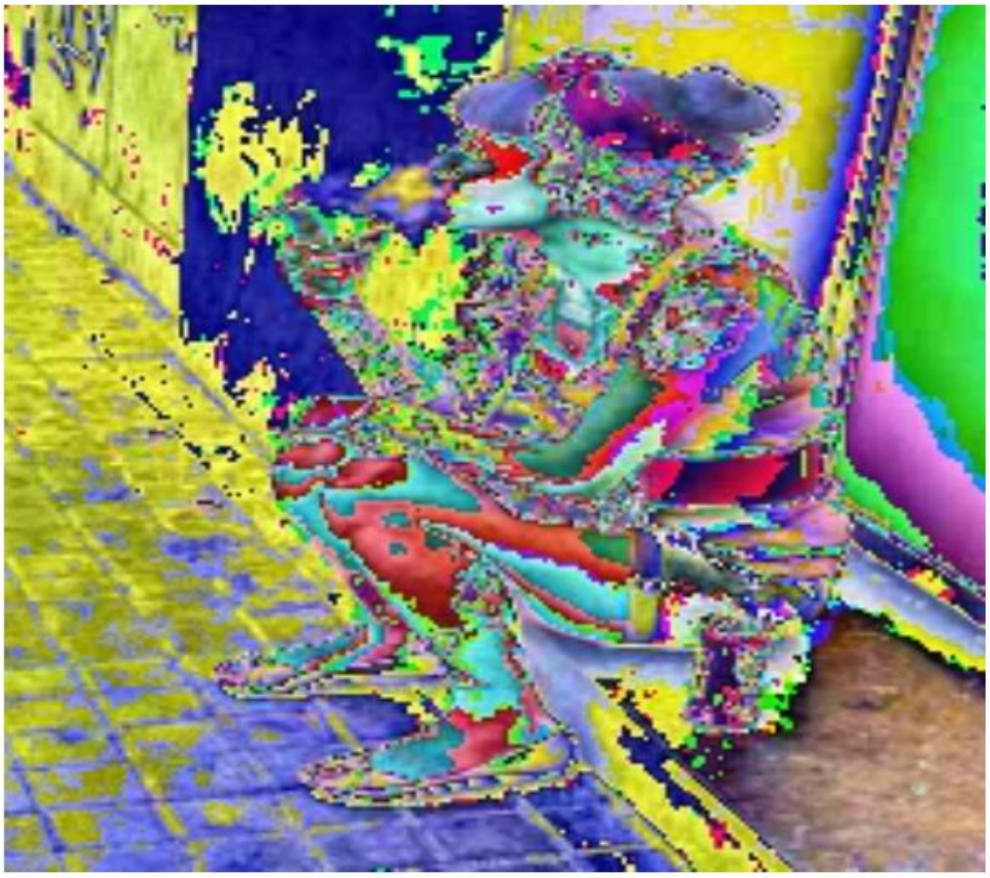
Epoch [8/8], Train Loss: 0.1210, Train Acc: 0.9527, Val Loss: 0.3029, Val Acc: 0.5500

Plotting Training and Validation Loss:



Training and Validation Accuracy:





Randomly Selected Image - Predicted Class: Smoking

Refresh

Real World application:

1. Surveillance and Security: It will help to identify unusual activities or behaviours in various places. This will detect actions like fights, thefts or suspicious behaviour to alert security personnel.

2. Healthcare: It will help to observe patient movement to detect falls or abnormal behaviour in hospitals or elderly care facilities.

3. Sports and Fitness: This will provide feedback on exercise routines and detect incorrect postures or movements.

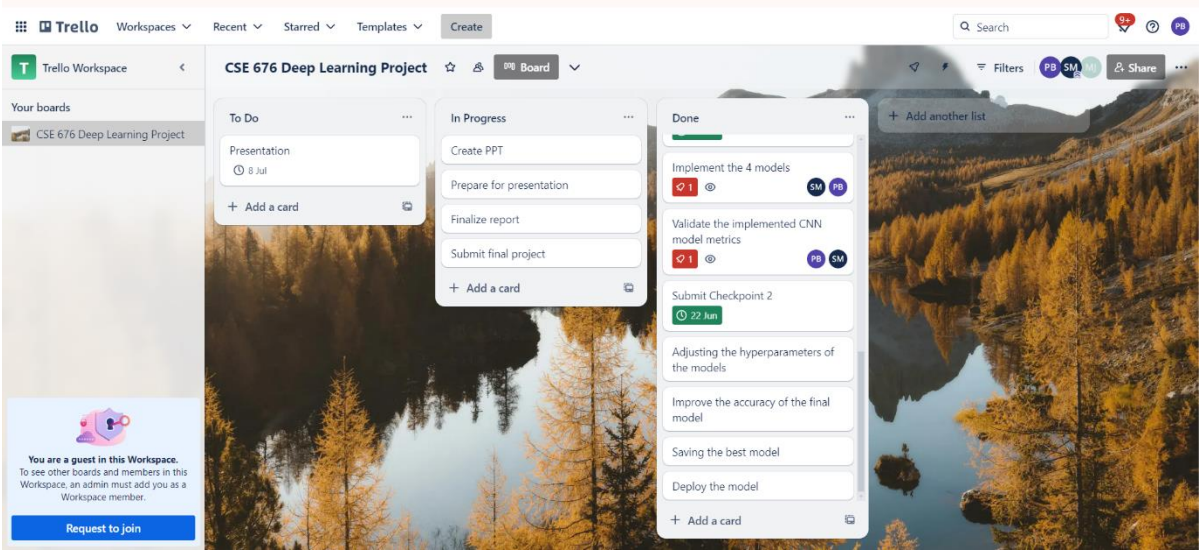
4. Human-Computer Interaction: It will enable touchless interaction with devices and application through gesture recognition.

5. Robotics: This will allow robots to understand and respond to human actions in collaborative environment.

6. Smart Homes: Triggering smart home devices based on recognized actions, such as turning off lights when leaving a room. Also, it will detect unusual behavior to alert family members or emergency services.

7. Education: Observing student activities and engagement to provide feedback to teachers.

Trello Screenshot:



References:

- https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/
- https://www.geeksforgeeks.org/understanding-googlenet-model-cnn-architecture/
- Referred CSE – 676 Assignment 1Part 2 of sriveera and pbodke
- https://medium.com/@ilaslanduzgun/create-vgg-from-scratch-in-pytorch-aa194c269b55
- http://vision.stanford.edu/Datasets/40actions.html
- https://pytorch.org/docs/stable/generated/torch.nn.AdaptiveAvgPool2d.html

- https://pytorch.org/docs/stable/generated/torch.nn.functional.binary_cross_entropy_with_logits.html
- https://medium.com/@sharma.tanish096/detailed-explanation-of-residual-network-resnet50-cnn-model-106e0ab9fa9e
- https://paperswithcode.com/method/googlenet#:~:text=GoogLeNet%20is%20a%20type%20of,filter%20sizes%20in%20each%20block.
- https://pytorch.org/hub/pytorch_vision_densenet/
- https://streamlit.io/