Capstone project report on

"Identification of Active Windows of Chatbox/Emails Using ResNet 18, 34 and 50 for Image Classification"

Submitted to

Praxis Business School, Kolkata

Post Graduate Program in

Data Science

By

Kishore Dipra Dutta(A22019)

Pulokesh Chatterjee(A22026)

Soujatya Das(A22033)

Soumya Chatterjee(A22046)

Subho Halder(A22035)

Utsav Dey(A22040)

Under the guidance of **Prof. Dr. Subhasis Dasgupta**

Department of Data Science Year 2023

ABSTRACT:

Chatboxs and email communication have become ubiquitous in our daily lives. However, excessive use of these tools can lead to reduced productivity and efficiency. Therefore, it is important to have a system that can identify when a person is actively using Chatboxes/emails. In this capstone project, we have developed an image classification model using **ResNet** (Residual Network) 34 and 50 to accurately identify when a person has active windows of Chatbox/emails open on their computer screen. We have collected a large dataset of computer screen images with and without active Chatbox/email windows, and used transfer learning and data augmentation techniques to train our models. Our results showed that the ResNet models achieved high accuracy in identifying active Chatbox/email windows. This model has potential applications in various settings, including monitoring employees' computer usage during work hours, identifying if students are using Chatboxes/emails during an examination, or even helping individuals manage their productivity.

Online exam
A
B
C

INTRODUCTION:

In recent years, there has been a significant increase in the use of technology in both education and the workplace. As a result, communication channels such as Chatboxs and emails have become commonplace in many industries. However, the use of these tools during exams and work raises concerns about potential cheating and unethical behavior. Therefore, the identification of Chatbox and email usage during examinations and work is an important topic that requires attention.

This topic involves exploring the frequency, patterns, and types of communication used through Chatboxes and emails during exams and work. It also involves developing effective methods for identifying inappropriate usage, such as cheating or unethical behavior. Understanding the usage of Chatboxes and emails during exams and work can help employers and educators to develop policies and tools to prevent such behavior and ensure ethical conduct.

In this report, we will review the literature on this topic, describe the methodology used to collect and analyze data, present our findings, and discuss the implications of our findings. The identification of Chatbox and email usage during examinations and work is an important step towards promoting academic and professional integrity, while also fostering a culture of responsible technology use.

Chatboxs and email communication have revolutionized the way we interact with technology and each other. They have become integral to our daily lives, providing us with a quick and convenient way to communicate and get things done. However, excessive use of these tools can lead to reduced productivity and efficiency. As a result, it is important to have a system that can identify when a person is actively using Chatboxes/emails.

In this capstone project, we aim to develop an image classification model that can accurately identify when a person has active windows of Chatbox/emails open on their computer screen. To achieve this, we will be using ResNet 34 and 50 for image classification. The model will be trained on a dataset of computer screen images with and without active Chatbox/email windows, and evaluated on a separate test set to assess its accuracy.

TABLE OF CONTENTS:

CHAPTER 1

GENERALIZED METHODOLOGY:

The architecture behind using CNN (Convolutional Neural Network) models like Resnet 18, 34 and 50 in order to predict and classify images involves enormous amounts of data (in form of images) which then needs to be preprocessed and refined as per our requirements. Post this, the model architecture needs to be selected keeping in mind to choose the appropriate CNN architecture for the image classification task. Having completed the mentioned procedure, the Transfer Learning needs to be implemented which involves using a pre-trained CNN model that has already been trained on a given large dataset, which allows the model to learn features which are relevant for image classification. Now, the dataset of images needs to be classified on the basis of training and validation sets to measure accuracy and identify potential overfitting. Finally, the last step is to test the aforementioned CNN models on a separate set of images and evaluate its performance using metrics such as accuracy, precision, recall and F1 score.

CHAPTER 1.1

Data Collection:

The mode and method of data collection is a bit of a hectic procedure as the same needs to be done manually due to our abstract idea and objective of the project. The categories need to be set in order to set and images need to be classified as per, so that the model can be taught well as to what the differences are between the uploaded screenshots and ultimately what we are trying to achieve. The general steps which are included in such data collection for image classification are:

- ➔ Defining the problem well: This is to determine what kind of images we are trying to classify and which features are important for the classification task.
- ➔ Adjudging the size of the dataset: This depends on how complex our classification problem is and what level of performance/accuracy we are trying to achieve. A large dataset yields better performance however it also takes more time and resources to collect and label the images.
- ➔ Collecting the images: Gathering a diverse set of images from varied resources and channels can be useful. Also, using search engines to find the sources of those mentioned images can add to the long-run.
- ➔ Preprocessing the images: Before using image classification, we might need to resize, crop and normalize them as per our best-suited needs and requirements. This also involves removal of any boundaries and backgrounds.
- ➔ Labeling and categorizing the images under correct folder: In order to correctly label images and drop them into correct folders might be a time-consuming task, however we can use specified softwares or codes in order to do these all at once within a given amount of time-frame.
- ➔ Splitting the dataset: Dividing the dataset into training, validation and test sets can either be done during the data collection phase or can be implemented in the coding segment. The training set is used to train your model, the validation set is used to tune hyperparameters and evaluate performance, and the test set is used to evaluate the final performance of your model.

Having followed these steps can help us create a high-quality dataset which can be used to train a robust and efficient image classification model.

CHAPTER 1.2

Model Architecture Selection:

Resnet 18, 34, and 50 are popular CNN models for image classification because of their deep architecture and the inclusion of residual connections, which address the vanishing gradient problem. The vanishing gradient problem occurs when the gradients become too small during backpropagation, making it difficult to optimize the weights and biases of the CNN model. Residual connections allow the model to skip over one or more layers, which makes it easier to propagate gradients and learn complex features. Resnet 18 has 18 layers, Resnet 34 has 34 layers, and Resnet 50 has 50 layers. As the depth of the model increases, its ability to learn more complex features and patterns also increases. These very characteristics of the Resnet models fit our requirement as well as the dataset we are working with, and also aim well towards achieving our objective with a higher level of efficiency.

The application of Resnet models (18, 34 and 50) have yielded ecstatic results in terms of efficiency which in turn defines its ability to read the images in the dataset with higher intricacies.

CHAPTER 1.3

Transfer Learning:

Introduction:

Deep Learning has taken the world by storm in recent years, and Convolutional Neural Networks (CNNs) are a popular subset of deep learning models that have proven to be highly effective in a variety of computer vision tasks, such as image classification, object detection, segmentation, and more. However, training deep CNNs requires a large amount of annotated data and computational resources, which can be a challenge in many real-world applications. Transfer learning, on the other hand, is a powerful technique that can help alleviate these challenges by enabling us to leverage pre-trained models and their learned representations for new tasks with limited data. In this note, we will explore transfer learning in CNNs, its applications, challenges, and best practices.

1. **Transfer Learning:** Transfer learning is a technique in machine learning that enables us to transfer knowledge learned from one task to another. In the context of deep learning, transfer learning involves using a pre-trained model's learned features as a starting point for a new task rather than training a model from scratch. This can save us a significant amount of time, data, and computation required to train a model from scratch, especially in scenarios where data is limited or computational resources are scarce.
2. **Types of Transfer Learning:** Transfer learning can be broadly categorized into two types: feature extraction and fine-tuning.

2.1 Feature Extraction: In feature extraction, we use the pre-trained model's learned features as fixed inputs to a new model that we train on our specific task. The pre-trained model is typically trained on a large-scale dataset such as ImageNet, and its learned features can capture general patterns and structures that are useful for a wide range of computer vision tasks. We can extract these features from the pre-trained model and use them as inputs to train a new classifier, typically a linear classifier, on top of them. This new classifier can be trained on a smaller dataset specific to our task, and we can achieve high accuracy with less training data and computation compared to training a model from scratch.

2.2 Fine-tuning: Fine-tuning is a technique that involves taking a pre-trained model and training it on a new task with a smaller dataset by adjusting its weights to fit the new data. In fine-tuning, we start with a pre-trained model, remove the last few layers that were designed for the original task, and add new layers that are specific to our task. We then train the model on our task-specific dataset while keeping the weights of the earlier layers fixed, and only updating the weights of the added layers. This enables us to transfer the knowledge learned by the pre-trained model to our task while also adapting it to the specific characteristics of our data.

3. Applications of Transfer Learning:

Transfer learning has been successfully applied to a wide range of computer vision tasks, such as image classification, object detection, segmentation, and more. Some of the popular applications of transfer learning in CNNs are:

3.1 Image Classification:

Image classification is one of the most common computer vision tasks, and transfer learning has been widely used in this context. The pre-trained models such as VGG, ResNet, Inception, and EfficientNet that are trained on large-scale datasets such as ImageNet have been used as starting points for many image classification tasks. We can use the learned features of these pre-trained models as inputs to a new classifier that we train on a smaller dataset specific to our task. This enables us to achieve high accuracy with limited data and computation.

3.2 Object Detection: Object detection is a more challenging task than image classification, as it requires not only identifying the objects in an image but also localizing them. Transfer learning has been used in object detection tasks by fine-tuning pre-trained models such as Faster R-CNN, YOLO, and SSD on specific object

CHAPTER 1.4

Training and Validation:

Training a CNN involves optimizing a set of parameters or weights that allow the network to make accurate predictions on new data. However, because CNNs have a large number of parameters, it is important to use a validation set to evaluate the model's performance and prevent overfitting. In this note, we will discuss the importance of training and validation in CNNs and explore some best practices for achieving high accuracy and generalization.

1. Training in CNNs

Training a CNN involves minimizing a loss function, which measures the difference between the predicted and true labels for a set of input data. This is done by updating the weights of the network using an optimization algorithm such as stochastic gradient descent (SGD). The goal of training is to find a set of weights that minimize the loss function and allow the network to make accurate predictions on new data.

Training a CNN involves several steps, including:

a. Preprocessing:

This involves preparing the data for training by normalizing the pixel values, resizing images to a common size, and augmenting the data with techniques such as rotation, flipping, and cropping. This helps prevent overfitting and allows the network to generalize to new data.

b. Initialization:

The weights of the network are randomly initialized using techniques such as Xavier initialization or He initialization. This allows the network to learn different features from the input data and prevents the gradients from exploding or vanishing during training.

c. Forward propagation:

The input data is fed into the network, and the output is computed using the current set of weights.

d. Backward propagation:

The gradients of the loss function with respect to the weights are computed using the backpropagation algorithm. These gradients are used to update the weights using an optimization algorithm such as SGD.

e. Repeat:

Steps c and d are repeated for multiple epochs or iterations until the loss function converges or reaches a minimum value.

2. Validation in CNNs

Validation is the process of evaluating the performance of a trained model on a set of data that is not used for training. This is important to ensure that the model is not overfitting to the training data and can generalize to new data. The validation set is typically a subset of the original data that is held out during training.

Validation involves several steps, including:

a. Preprocessing:

The validation data is preprocessed in the same way as the training data to ensure consistency.

b. Forward propagation:

The validation data is fed into the network, and the output is computed using the current set of weights.

c. Loss calculation:

The loss function is computed using the predicted labels and true labels for the validation data.

d. Repeat:

Steps b and c are repeated for the entire validation set, and the average loss is computed.

e. Evaluation:

The accuracy, precision, recall, and other performance metrics are computed using the predicted labels and true labels for the validation data.

3. **Best Practices for Training and Validation in CNNs**

To achieve high accuracy and generalization in CNNs, it is important to follow some best practices for training and validation. These include:

a. Use a large and diverse dataset:

CNNs require a large and diverse dataset to learn the relevant features and prevent overfitting. It is important to use a dataset that is representative of the target population and contains enough examples for each class.

b. Use data augmentation:

Data augmentation can help prevent overfitting and improve the generalization of the model. Common data augmentation techniques include rotation, flipping, cropping, and scaling.

c. Use transfer learning:

Transfer learning involves using a pre-trained model as a starting point for a new task.

CHAPTER 1.5

Hyperparameter Tuning:

Convolutional neural networks (CNNs) are a class of deep neural networks that are widely used for image and video analysis, natural language processing, and other tasks that involve structured data. Training a CNN involves optimizing a set of parameters or weights that allow the network to make accurate predictions on new data. However, because CNNs have a large number of parameters, it is important to use a validation set to evaluate the model's performance and prevent overfitting. In this note, we will discuss the importance of training and validation in CNNs and explore some best practices for achieving high accuracy and generalization.

1. Training in CNNs

Training a CNN involves minimizing a loss function, which measures the difference between the predicted and true labels for a set of input data. This is done by updating the weights of the network using an optimization algorithm such as stochastic gradient descent (SGD). The goal of training is to find a set of weights that minimize the loss function and allow the network to make accurate predictions on new data.

Training a CNN involves several steps, including:

a. Preprocessing:

This involves preparing the data for training by normalizing the pixel values, resizing images to a common size, and augmenting the data with techniques such as rotation, flipping, and cropping. This helps prevent overfitting and allows the network to generalize to new data.

b. Initialization:

The weights of the network are randomly initialized using techniques such as Xavier initialization or He initialization. This allows the network to learn different features from the input data and prevents the gradients from exploding or vanishing during training.

c. Forward propagation:

The input data is fed into the network, and the output is computed using the current set of weights.

d. Backward propagation:

The gradients of the loss function with respect to the weights are computed using the backpropagation algorithm. These gradients are used to update the weights using an optimization algorithm such as SGD.

e. Repeat:

Steps c and d are repeated for multiple epochs or iterations until the loss function converges or reaches a minimum value.

2. Validation in CNNs

Validation is the process of evaluating the performance of a trained model on a set of data that is not used for training. This is important to ensure that the model is not overfitting to the training data and can generalize to new data. The validation set is typically a subset of the original data that is held out during training.

Validation involves several steps, including:

a. Preprocessing:

The validation data is preprocessed in the same way as the training data to ensure consistency.

b. Forward propagation:

The validation data is fed into the network, and the output is computed using the current set of weights.

c. **Loss calculation:**

The loss function is computed using the predicted labels and true labels for the validation data.

d. **Repeat:**

Steps b and c are repeated for the entire validation set, and the average loss is computed.

e. **Evaluation:**

The accuracy, precision, recall, and other performance metrics are computed using the predicted labels and true labels for the validation data.

3. Best Practices for Training and Validation in CNNs

To achieve high accuracy and generalization in CNNs, it is important to follow some best practices for training and validation. These include:

a. Use a large and diverse dataset: CNNs require a large and diverse dataset to learn the relevant features and prevent overfitting. It is important to use a dataset that is representative of the target population and contains enough examples for each class.

b. Use data augmentation: Data augmentation can help prevent overfitting and improve the generalization of the model. Common data augmentation techniques include rotation, flipping, cropping, and scaling.

c. Use transfer learning: Transfer learning involves using a pre-trained model as a starting point for a new task.

CHAPTER 1.6

Testing and Evaluation:

Testing and evaluation in Convolutional Neural Networks (CNNs) is crucial to assess the performance of the model on a given task. The process involves training the model on a training dataset, validating it on a validation dataset, and finally testing it on a separate testing dataset.

Here are the main steps involved in testing and evaluating a CNN:

1. **Data Preparation:** Collect the data required for the task, preprocess it, and split it into training, validation, and testing sets.
2. **Model Training:** Train the CNN model on the training dataset using an appropriate optimization algorithm, such as stochastic

gradient descent, and a suitable loss function, such as cross-entropy.

3. **Model Validation:** Evaluate the performance of the trained model on the validation dataset. This helps in monitoring the training process and avoiding overfitting. Early stopping is often used to stop training when validation accuracy stops improving.
4. **Model Testing:** Evaluate the final performance of the trained model on the testing dataset. This provides an estimate of how well the model will perform on unseen data.
5. **Metrics Calculation:** Calculate various evaluation metrics, such as accuracy, precision, recall, and F1 score, to measure the performance of the model.
6. **Model Improvement:** If the model performance is not satisfactory, iterate through the process of data preparation, model training, validation, and testing to improve the model.

It is essential to perform these steps carefully and objectively, to ensure that the model performs and classifies well to unseen data.

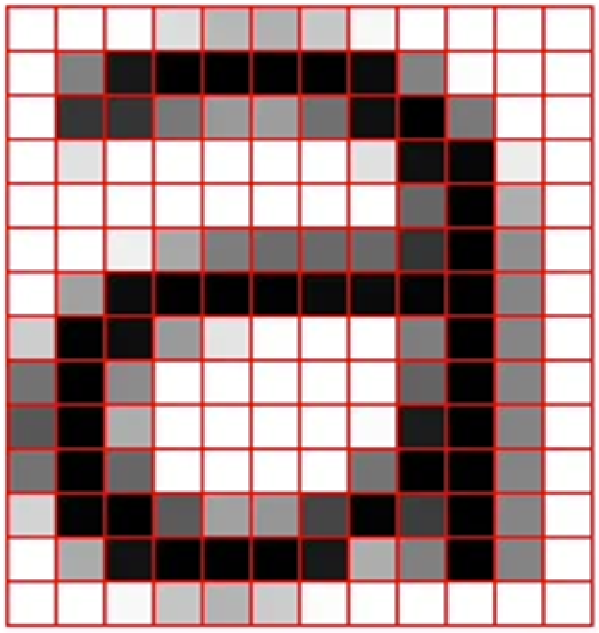
CHAPTER 1.7

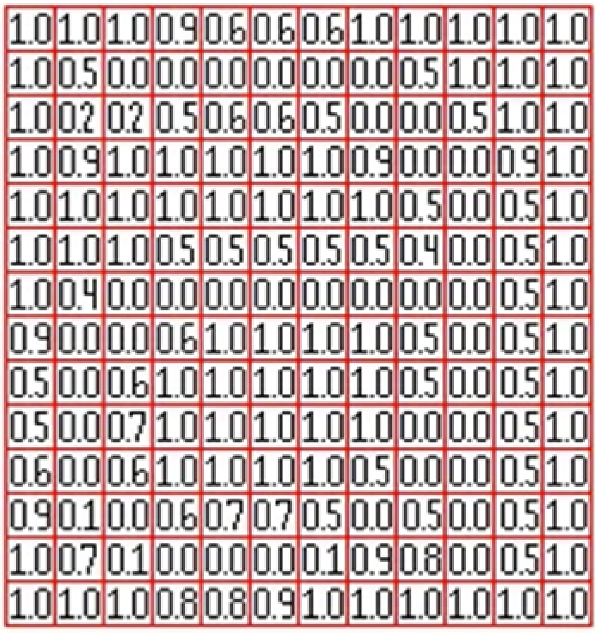
Popup Notification Generation for forbidden objects on screen:

CHAPTER 2

CNN (CONVOLUTIONAL NEURAL NETWORK):

A convolutional neural network (CNN) is a deep learning architecture that is commonly used for image and video analysis tasks. It is designed to automatically learn features from raw input data by using convolutional filters to identify spatial patterns in the input. A digital image is a binary representation of visual data. It contains a series of pixels arranged in a grid-like fashion that contains pixel values to denote how bright and what color each pixel should be.





1.0	1.0	1.0	0.9	0.6	0.6	0.6	1.0	1.0	1.0	1.0	1.0
1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	1.0	1.0
1.0	0.2	0.2	0.5	0.6	0.6	0.5	0.0	0.0	0.5	1.0	1.0
1.0	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.0	0.0	0.9	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
1.0	1.0	1.0	0.5	0.5	0.5	0.5	0.5	0.4	0.0	0.5	1.0
1.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0
0.9	0.0	0.0	0.6	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
0.5	0.0	0.6	1.0	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
0.5	0.0	0.7	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.5	1.0
0.6	0.0	0.6	1.0	1.0	1.0	1.0	0.5	0.0	0.0	0.5	1.0
0.9	0.1	0.0	0.6	0.7	0.7	0.5	0.0	0.5	0.0	0.5	1.0
1.0	0.7	0.1	0.0	0.0	0.0	0.1	0.9	0.8	0.0	0.5	1.0
1.0	1.0	1.0	0.8	0.8	0.9	1.0	1.0	1.0	1.0	1.0	1.0

Figure explanation: Representation of image as a grid of pixels.

The working architecture of a CNN typically consists of several layers, including convolutional layers, pooling layers, and fully connected layers. The input to the network is an image or a set of images, and the output is a probability distribution over a set of classes.

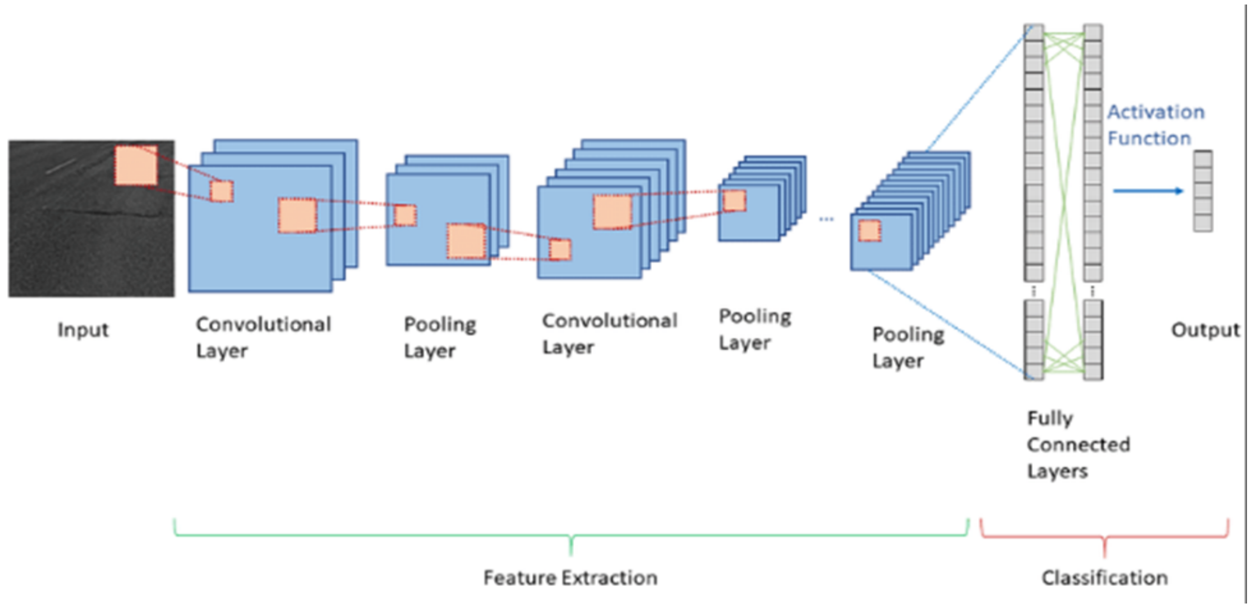


Figure explanation: Working architecture of CNN.

Each and every layer of a CNN is important towards proper working and architecture of the model due to its vastly layered components and stages

which are explained below:

Input layer: This layer takes the raw input image(s) and applies any necessary preprocessing, such as resizing or normalization.

- Convolutional Layer: This is the core building block of the CNN. It carries the main portion of the network's computational load. The output of the convolutional layer is a set of feature maps, where each map represents the activation of a particular filter. The formula for the convolution operation is:

  feature_map = activation (convolution (input, filter) + bias),

where input is the input image, filter is the convolutional filter, bias is a bias term, convolution is the convolution operation, and activation is an activation function such as ReLU or sigmoid. This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels. During the forward pass, the kernel slides across the height and width of the image-producing the image representation of that receptive region. This produces a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride. If we have an input of size W * W * D and D_{out} number of kernels with a spatial size of F with stride S and amount of padding P, then the size of output volume can be determined by the following formula:

$W_{out} = \{(W-F+2P)/S\} + 1$

This will yield an output volume of size $W_{out} * W_{out} * D_{out}$.

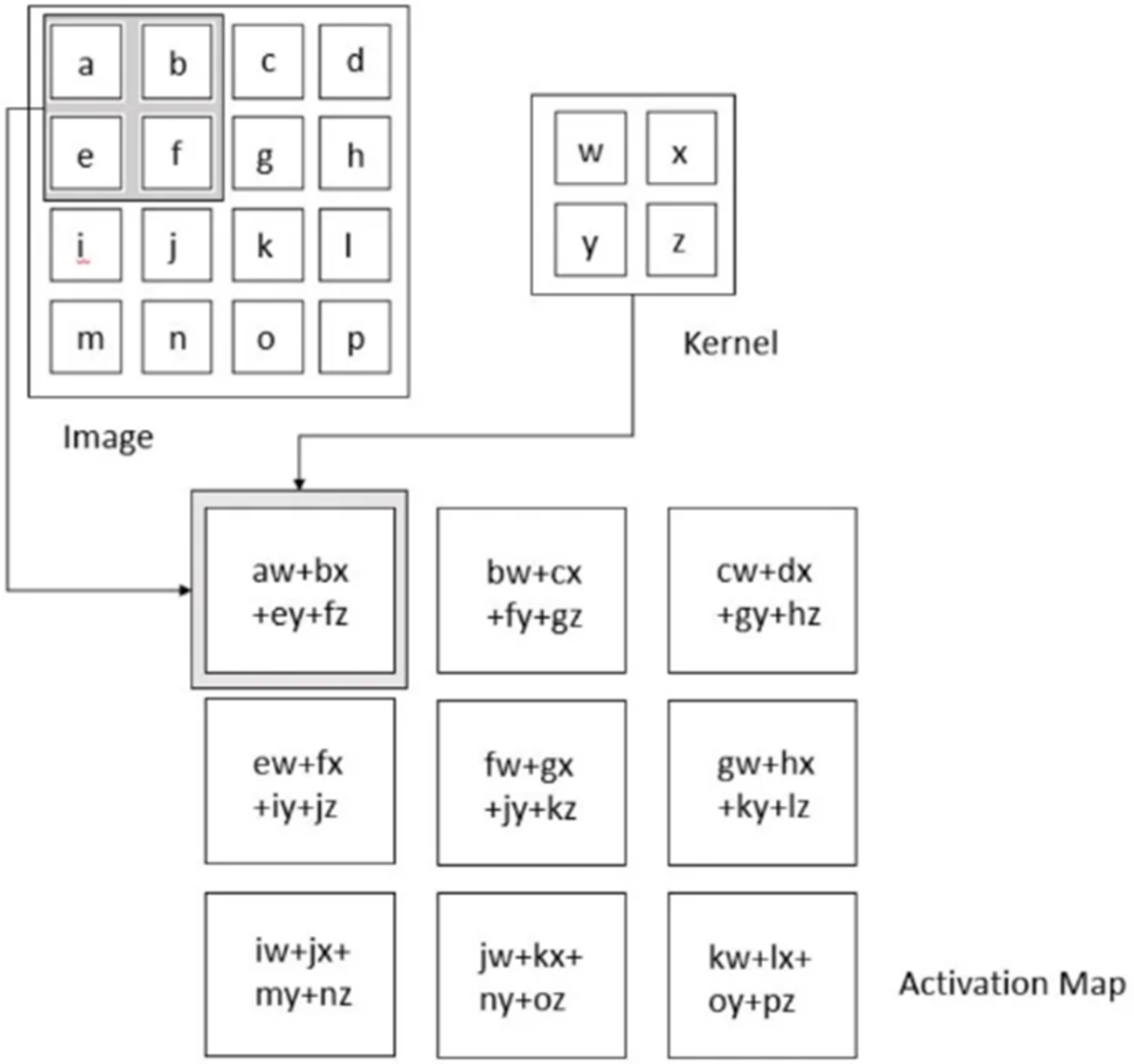


Figure Explanation: Convolution Operation.

- Pooling Layer: The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually. There are several pooling functions such as the average of the rectangular neighborhood, L2 norm of the rectangular neighborhood, and a weighted average based on the distance from the central pixel. However, the most popular process is max pooling, which reports the maximum output from the neighborhood. If we have an activation map of size W x W x D, a pooling kernel of spatial size F, and stride S, then the size of output volume can be determined by the following formula:

$$W_{out} = \{(W - F)/S\} + 1$$

which will yield an output volume of size $W_{out} * W_{out} * D$.

- Fully connected layer: This layer takes the flattened output of the previous layer and applies a set of weights to generate a probability distribution over the set of classes. The formula for the fully connected layer is:

  output = activation (dot (input, weights) + bias)

  where input is the flattened output of the previous layer, weights are the learnable weights of the layer, bias is a bias term, dot is the dot product operation, and activation is an activation function such as softmax or sigmoid.

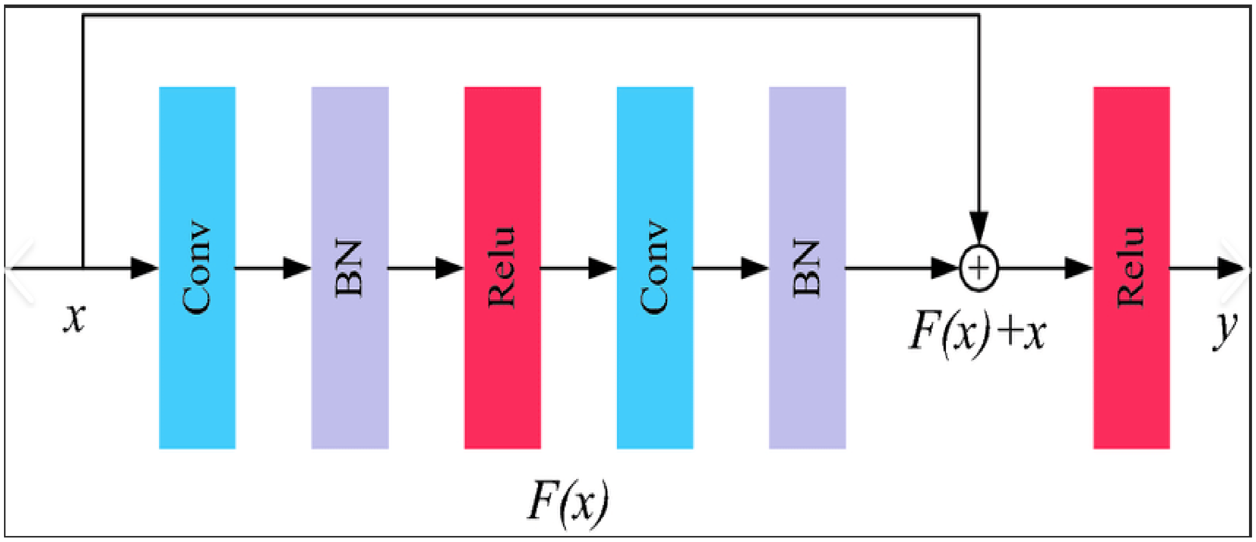
CHAPTER 2.1

RESNET(RESIDUAL NETWORK):

This is a deep neural network architecture which was introduced in order to make image-related tasks more efficient and robust in nature, without losing accuracy. The tendency to add so many layers by deep learning practitioners is to extract important features from complex images. This was the very first architecture to introduce skip connections which allows the network to skip over layers and preserve the required information. A shortcut connection is added that bypasses one or more layers and adds the output of those layers to the output of the shortcut connection. This creates a "residual" connection that carries information directly from the input to the output of the block.

The ResNet architecture and models have been widely used for image classification, object recognition, semantic segmentation that has achieved recognition and has been standardized for such purposes worldwide.

ResNet has a basic residual block as shown in the below diagram.



One of the most important features brought about by ResNet is Vanishing Gradient minimization. The vanishing gradient problem is encountered while training artificial neural networks that involve gradient based learning and backpropagation. We know that in backpropagation, we use gradients to update the weights in a network. But sometimes what happens is that gradient becomes vanishingly small, effectively preventing the weights to change values. This leads the network to stop training as the same values are propagated over and over again and no useful work is done. The ResNet architecture solves the problem of vanishing gradients in deep neural networks by introducing skip connections, also known as residual connections, that allow the network to skip over layers.

As our project concentrates on working with a huge amount of complex image dataset, we have selected the best and most efficient image classification architecture - ***ResNet***.

The several versions of ResNet namely ResNet-18, ResNet-34, ResNet-50, ResNet-101 and ResNet-152 are currently being used on basis of the complexity of the images and the number of layers required to identify the objects, however we have identified and selected ResNet models - 18, 34 and 50 to be best suited for our purpose and the same has been driven well throughout the project.

CHAPTER 2.2

MODELS TO BE USED:

ResNet 18:

The ResNet-18 architecture consists of 18 layers, including 16 convolutional layers and 2 fully connected layers. The first layer is a 7x7 convolutional layer with a stride of 2, followed by a max pooling layer with a stride of 2. The next layers consist of convolutional layers with 3x3 filters and varying numbers of filters per layer. The final layer is a fully connected layer that produces the output classification. The architecture is designed to improve the training of very deep neural networks by introducing residual blocks, which allow the network to skip over layers and preserve information.

The ResNet-18 architecture can be expressed mathematically as follows:

Input: A 3-channel RGB image with a height and width of 224 pixels.

The model architecture is trained using a loss function such as cross-entropy loss, and the weights of the network are optimized using backpropagation with a stochastic gradient descent optimizer or a variant such as Adam.

In our project, we used ResNet 18 to classify images of computer screens with and without active Chatbox/email windows. We found that the shallower architecture of ResNet 18 was not that effective in capturing the relevant features and patterns in our dataset, resulting in low accuracy in identifying active Chatbox/email windows.

We also found that ResNet 18 was not as effective as compared to ResNet 34 and 50 in accurately identifying active Chatbox/email windows in images of computer screens. This may be due to the limited depth of ResNet 18, which may have made it difficult for the model to capture fine-grained details and complex patterns in the images.

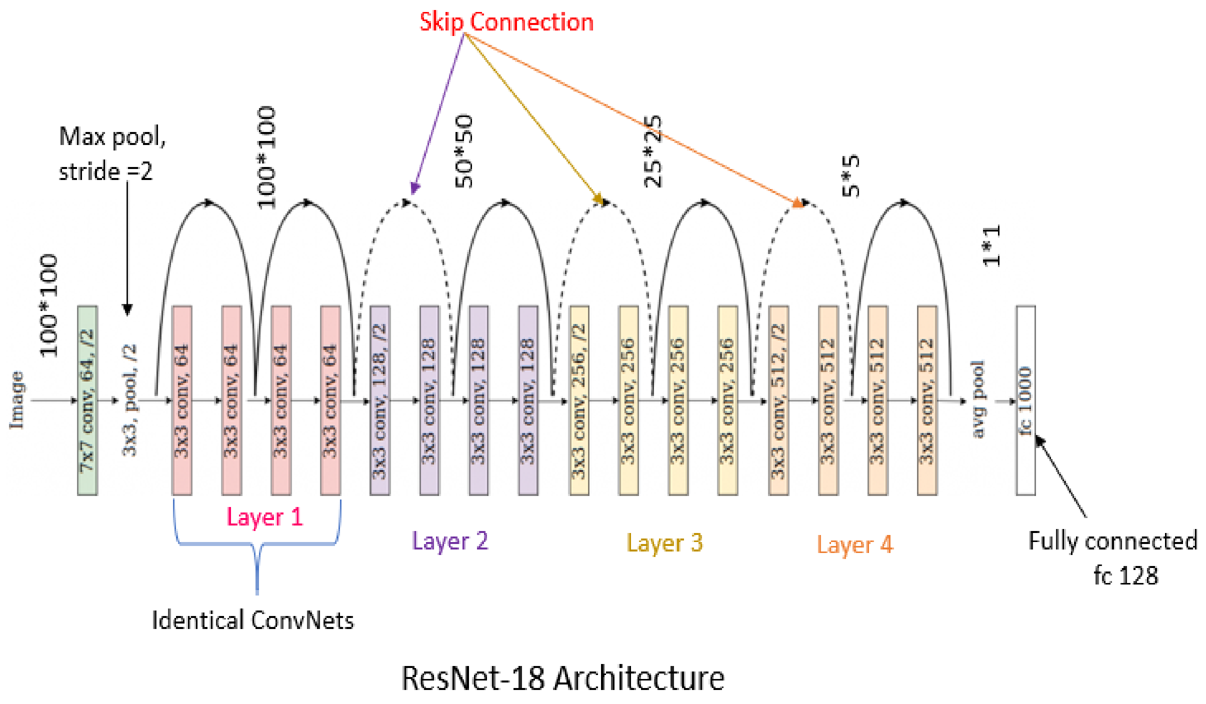


Figure Explanation: Resnet 18 Working Architecture

ResNet 34:

ResNet 34 has 34 layers and utilizes skip connections, also known as residual connections, which enable information to flow more easily through the network. This allows the model to avoid the vanishing gradient problem, which can occur when deep neural networks are trained. The skip connections also help to preserve the information from earlier layers of the network, allowing the model to better capture and retain important information throughout the training process.

The ResNet-34 architecture contains 16 residual blocks, each consisting of two convolutional layers with 3x3 filters and 64 output channels. The ResNet-34 architecture can be expressed mathematically as follows:

Input: A 3-channel RGB image with a height and width of 224 pixels.

In our project, we used ResNet 34 to classify images of computer screens with and without active Chatbox/email windows. We found that the deep architecture and skip connections of ResNet 34 were effective in capturing the relevant features and patterns in our dataset, resulting in high accuracy in identifying active Chatbox/email windows.

Overall, ResNet 34 is a powerful neural network architecture that is well-suited for complex image classification tasks, making it a useful tool for a wide range of applications in computer vision.

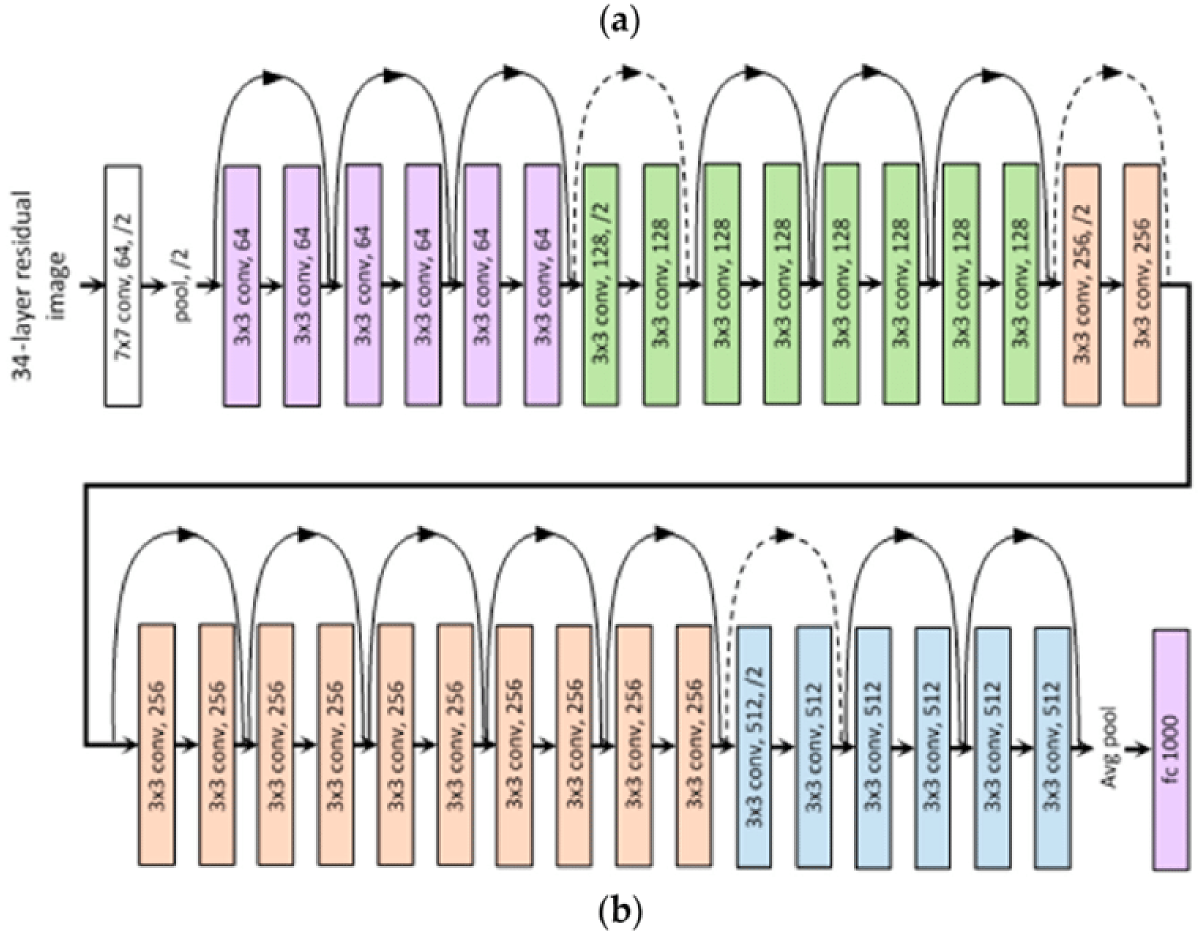


Figure Explanation: Resnet 34 Working Architecture

ResNet 50:

ResNet 50 is another pre-trained neural network architecture that is widely used for image classification tasks.

ResNet 50 has 50 layers and also utilizes skip connections to enable information to flow more easily through the network. Like ResNet 34, the skip connections in ResNet 50 help to avoid the vanishing gradient problem and preserve important information from earlier layers.

The ResNet-50 architecture contains 16 residual blocks, grouped into four stages (i.e., 3 residual blocks per stage), each consisting of multiple convolutional layers with 1x1, 3x3, and 1x1 filters, and varying numbers of output channels (256, 512, and 1024).

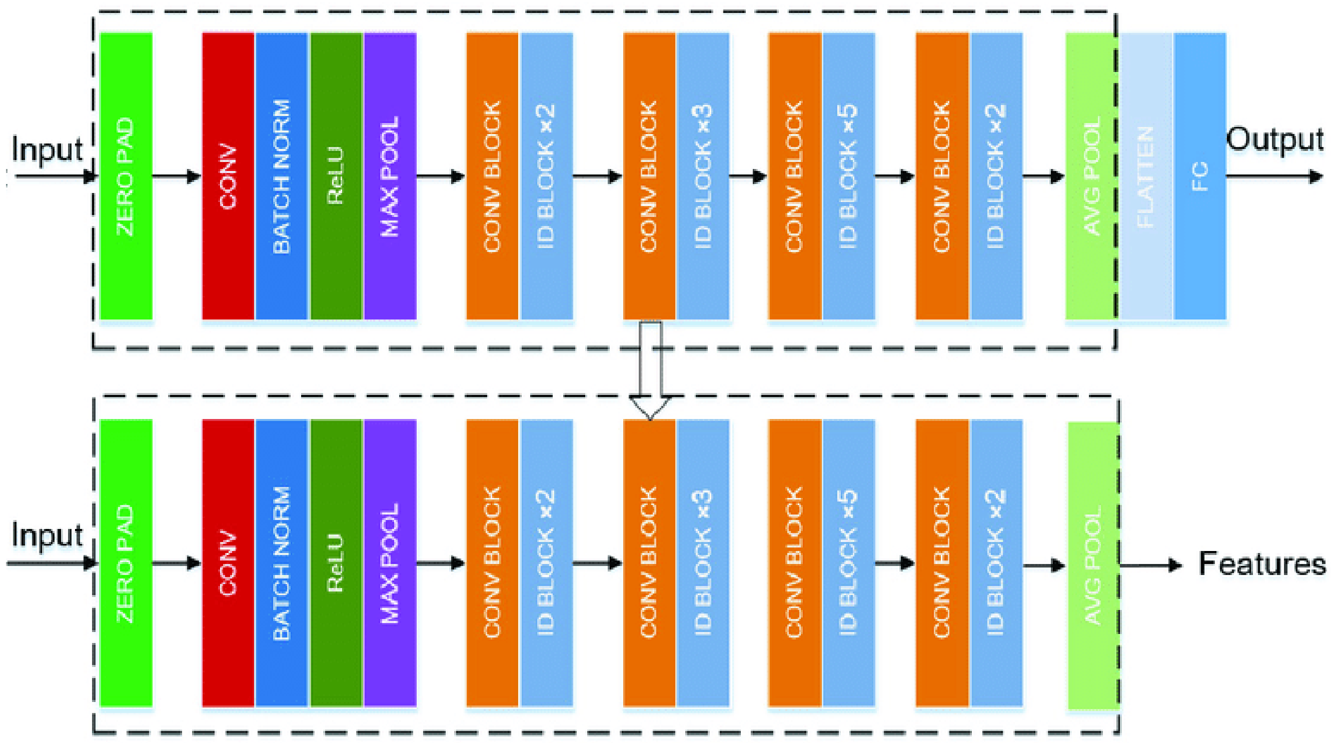
The ResNet-50 architecture can be expressed mathematically as follows:

Input: A 3-channel RGB image with a height and width of 224 pixels.

In our project, we used ResNet 50 alongside ResNet 34 to classify images of computer screens with and without active Chatbox/email windows. We found that ResNet 50, with its deeper architecture, was particularly effective in capturing complex features and patterns in the images, resulting in higher accuracy compared to ResNet 34.

By using both ResNet 34 and ResNet 50 in our project, we were able to achieve a high level of accuracy in identifying active Chatbox/email windows. This suggests that the deep architecture and skip connections of ResNet 50, in particular, were useful in capturing and retaining important information throughout the training process.

Overall, ResNet 50 is a powerful neural network architecture that is well-suited for complex image classification tasks, and in our project, it is proven to be particularly helpful in achieving high accuracy in identifying active Chatbox/email windows.



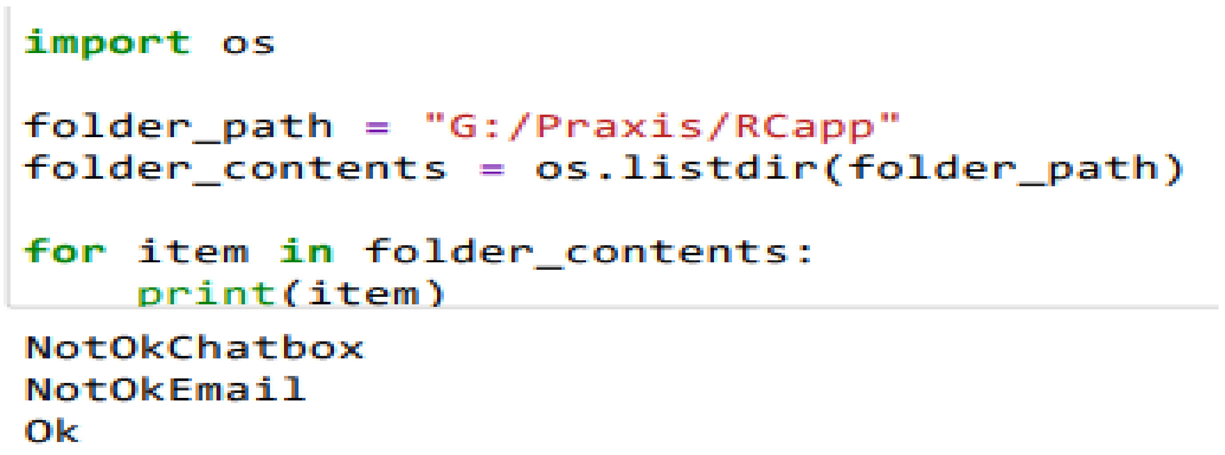
CHAPTER 3:

ACTUAL WORKING AND ARCHITECTURE OF THE MODEL:

CHAPTER 3.1:

DATA EXTRACTION:

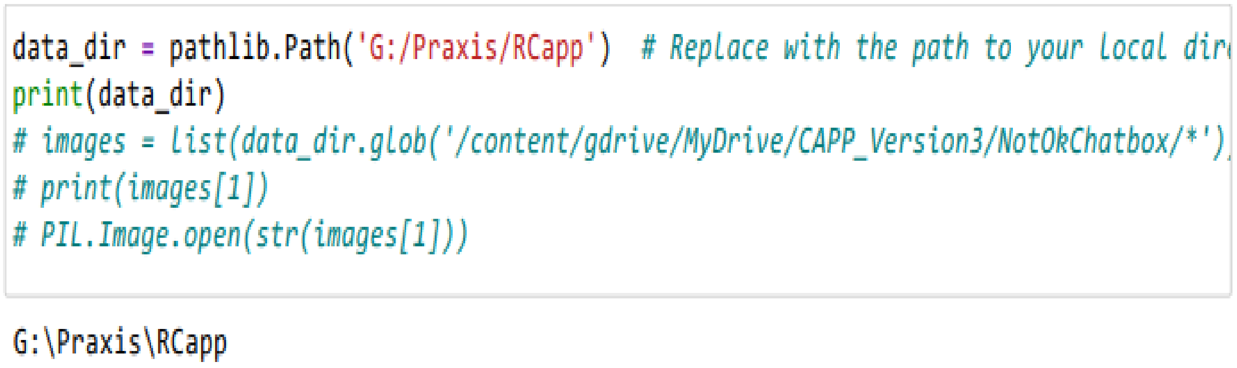
As we had uploaded the images with the below mentioned categorized and labeled folders on Google Drive, downloaded the same and hence extracted the dataset from local system using the folder/file path.



```
import os

folder_path = "G:/Praxis/RCapp"
folder_contents = os.listdir(folder_path)

for item in folder_contents:
    print(item)
```

```
NotOkChatbox
NotOkEmail
Ok
```

Figure Explanation: Code to extract files and folders of images from local system



```
data_dir = pathlib.Path('G:/Praxis/RCapp')  # Replace with the path to your local dir
print(data_dir)
# images = list(data_dir.glob('/content/gdrive/MyDrive/CAPP_Version3/NotOkChatbox/*'),
# print(images[1])
# PIL.Image.open(str(images[1]))
```

```
G:\Praxis\RCapp
```

CHAPTER 3.2:

DATA COLLECTION:

The method of data collection was a bit time consuming as we had taken screenshots of all possible channels from varied resources conforming to our required set of headlined categories namely:

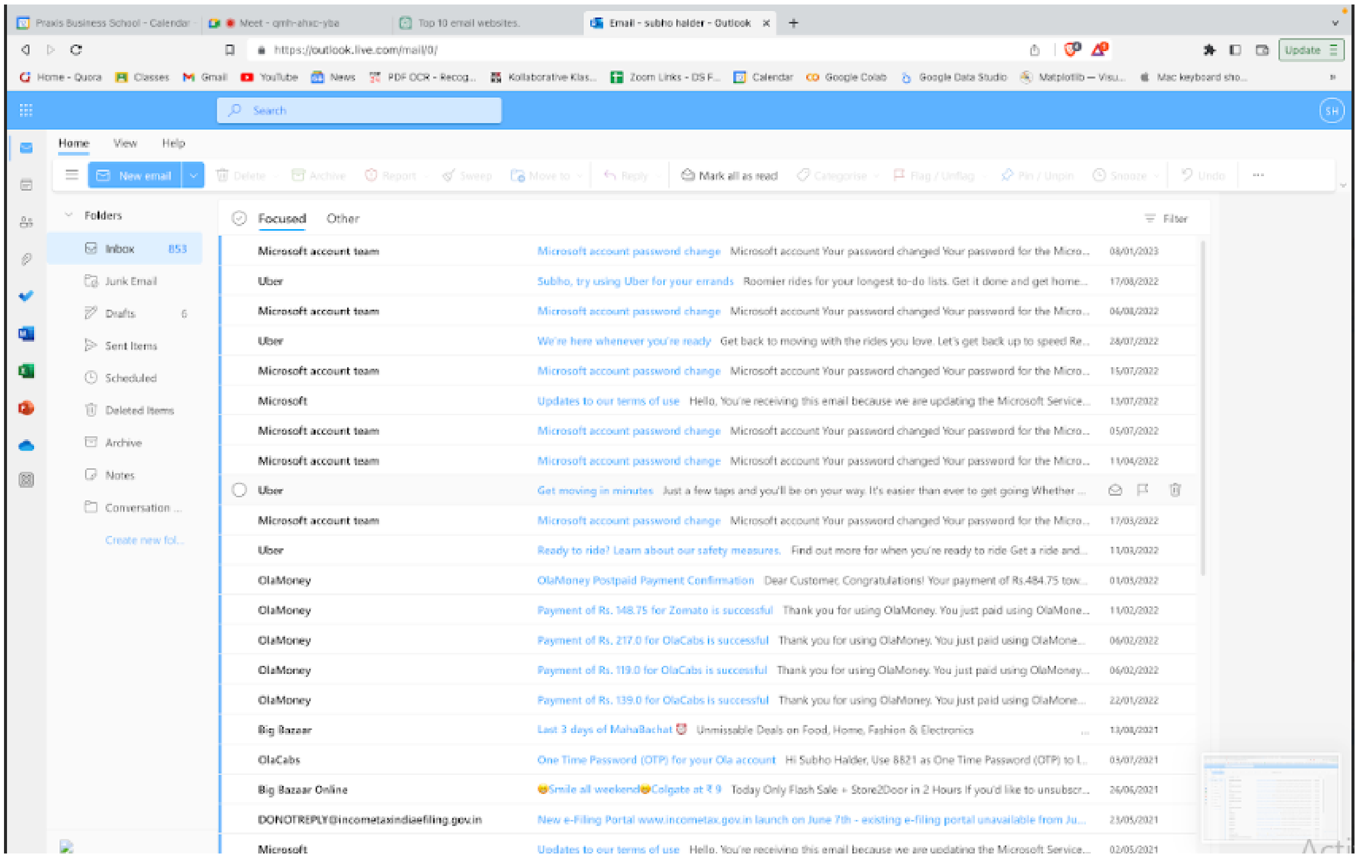
- NotOk Email
- NotOk Chatbox
- Ok

Now, with regards to the ‘NotOk Email’ category we reached out to all the several kinds of email inboxes existent in the open source market e.g., Gmail, Yahoo mail, Rediff mail, Yandex mail, Proton mail, Microsoft Outlook, iCloud mail and several others.

The ‘NotOk Chatbox’ category signifies those social networking applications and websites which helps in connecting people on a virtual platform, hence we have collected images of such medium namely WhatsApp, Line, Snapchat, Hike, Facebook, Instagram, Telegram, Signal, Viber, Skype, Microsoft Teams, Discord, Kakao Talk, Zoom, Webex, WickrMe, Mattermost, IMO, Google Meet, Band, Voxer, Wire and many more.

Lastly, the ‘Ok’ category is set for those images which contain snippets of browser, and other website usages which are allowed for an examination and accords well within the working ethics of the professional world.

To sum up in numbers, these 3 mentioned categories have around 4051 images, out of which 3241 have been used for training the model and 810 for validation.



Microsoft account team	Microsoft account password change	Microsoft account Your password changed Your password for the Micro...	08/01/2023
Uber	Subho, try using Uber for your errands	Roomier rides for your longest to-do lists. Get it done and get home...	17/08/2022
Microsoft account team	Microsoft account password change	Microsoft account Your password changed Your password for the Micro...	04/08/2022
Uber	We're here whenever you're ready	Get back to moving with the rides you love. Let's get back up to speed Re...	28/07/2022
Microsoft account team	Microsoft account password change	Microsoft account Your password changed Your password for the Micro...	15/07/2022
Microsoft	Updates to our terms of use	Hello, You're receiving this email because we are updating the Microsoft Service...	13/07/2022
Microsoft account team	Microsoft account password change	Microsoft account Your password changed Your password for the Micro...	05/07/2022
Microsoft account team	Microsoft account password change	Microsoft account Your password changed Your password for the Micro...	11/04/2022
Uber	Get moving in minutes	Just a few taps and you'll be on your way. It's easier than ever to get going Whether ...	
Microsoft account team	Microsoft account password change	Microsoft account Your password changed Your password for the Micro...	17/03/2022
Uber	Ready to ride? Learn about our safety measures.	Find out more for when you're ready to ride Get a ride and...	11/03/2022
OlaMoney	OlaMoney Postpaid Payment Confirmation	Dear Customer, Congratulations! Your payment of Rs.484.75 tow...	01/03/2022
OlaMoney	Payment of Rs. 148.75 for Zomato is successful	Thank you for using OlaMoney. You just paid using OlaMone...	11/02/2022
OlaMoney	Payment of Rs. 217.0 for OlaCabs is successful	Thank you for using OlaMoney. You just paid using OlaMone...	06/02/2022
OlaMoney	Payment of Rs. 119.0 for OlaCabs is successful	Thank you for using OlaMoney. You just paid using OlaMoney...	06/02/2022
OlaMoney	Payment of Rs. 139.0 for OlaCabs is successful	Thank you for using OlaMoney. You just paid using OlaMone...	22/01/2022
Big Bazaar	Last 3 days of MahaBachat ⏰	Unmissable Deals on Food, Home, Fashion & Electronics ...	13/08/2021
OlaCabs	One Time Password (OTP) for your Ola account	Hi Subho Halder, Use 8821 as One Time Password (OTP) to l...	03/07/2021
Big Bazaar Online	😁Smile all weekend😁Colgate at ₹ 9	Today Only Flash Sale + Store2Door in 2 Hours If you'd like to unsubscr...	26/06/2021
DONOTREPLY@incometaxindiaefiling.gov.in	New e-Filing Portal www.incometax.gov.in launch on June 7th - existing e-filing portal unavailable from Ju...		23/05/2021
Microsoft	Updates to our terms of use	Hello, You're receiving this email because we are updating the Microsoft Service...	02/05/2021

Figure Explanation: Example of **‘NotOk Email’** category screenshots

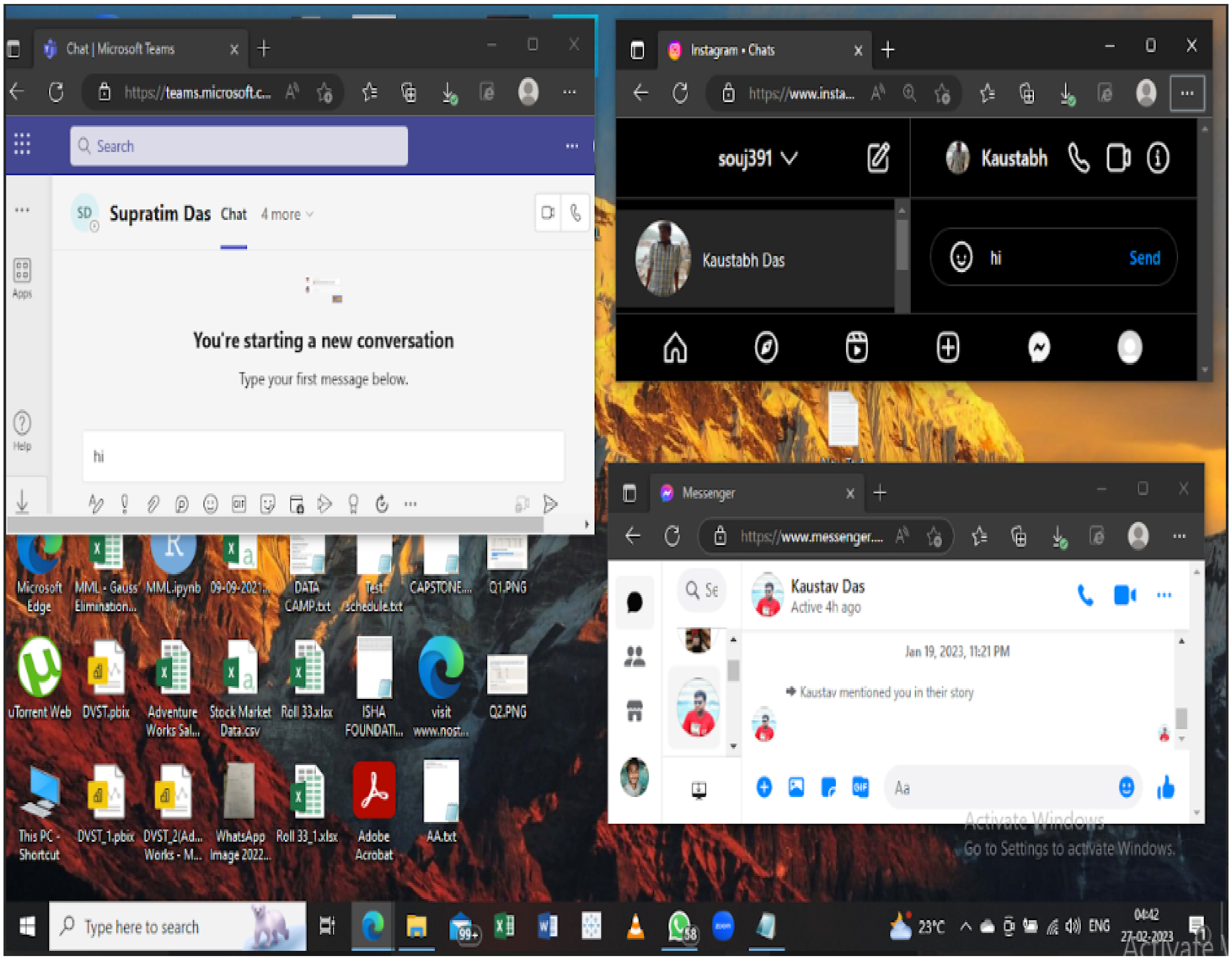
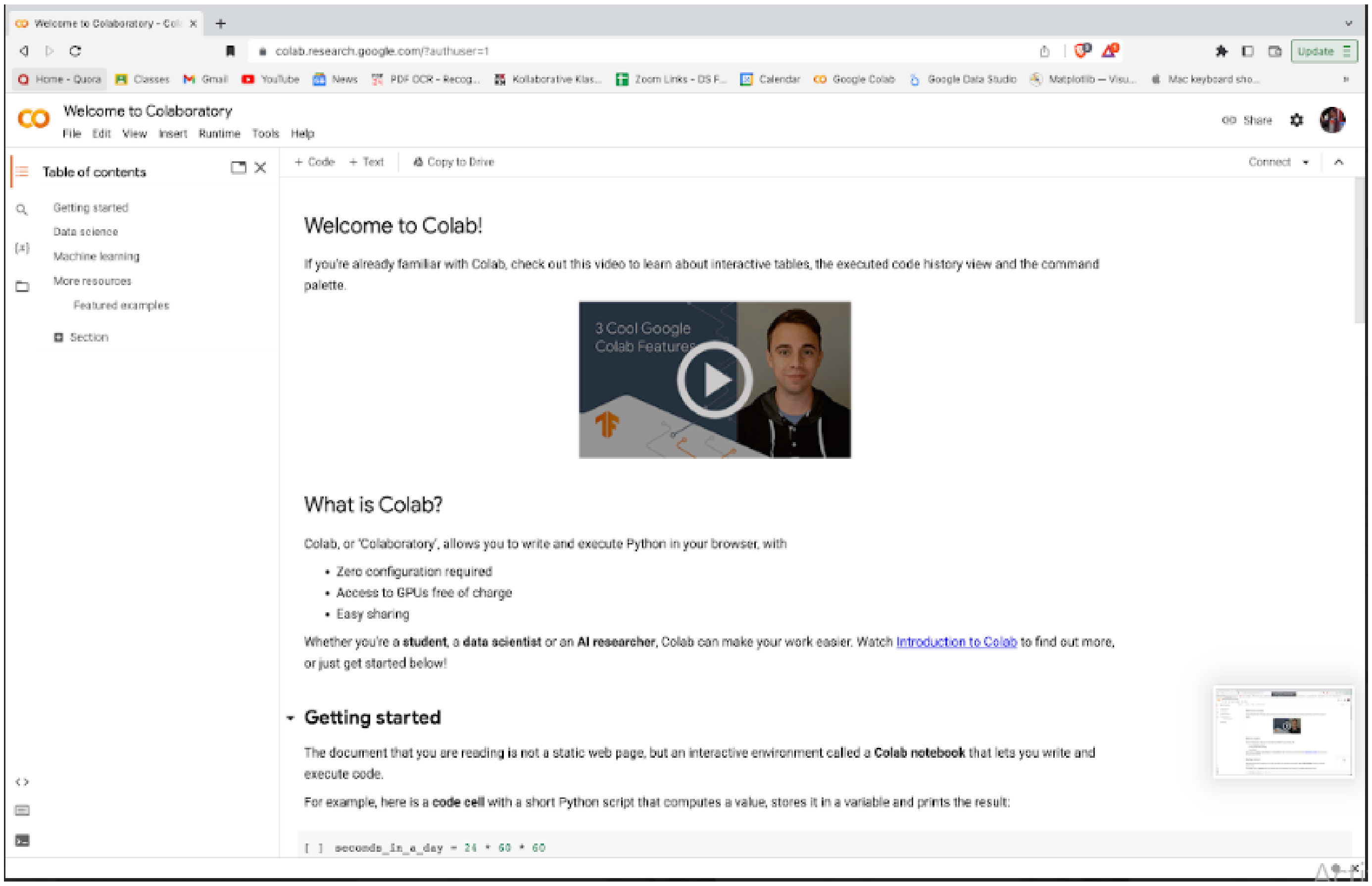


Figure Explanation: Example of **'NotOk Chatbox'** category screenshots



Welcome to Colab!

If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view and the command palette.

What is Colab?

Colab, or 'Colaboratory', allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to find out more, or just get started below!

Getting started

The document that you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable and prints the result:

```
[ ] seconds_in_a_day = 24 * 60 * 60
```

Figure Explanation: Example of **'Ok'** category screenshots

CHAPTER 3.3:

TRANSFER LEARNING:

Transfer Learning is a technique in deep learning where a pre-trained model is utilized as a starting point for a new task, which allows the model to leverage the knowledge gained from the original task and transfer it to the new task. This technique has become increasingly popular in recent years due to the availability of large pre-trained models, which can be fine-tuned for specific applications with relatively small amounts of data. Transfer Learning has been applied in a variety of domains, including image classification, natural language processing, and speech recognition (Pan and Yang, 2010).

In the given capstone project, transfer learning has been utilized to develop an image classification model using ResNet (Residual Network) 34 and 50 to identify when a person has active windows of Chatbox/emails open on their computer screen. The pre-trained ResNet models have been fine-tuned on the collected dataset of computer screen images with and without active Chatbox/email windows. This approach has significantly reduced the amount of training data required for the model and has improved its accuracy (Yosinski et al., 2014).

Furthermore, data augmentation techniques have been utilized to increase the diversity of the dataset and prevent overfitting of the model. The performance of the model has been evaluated on a separate test set to assess its accuracy using metrics such as accuracy, precision, recall, and F1 score (Abdi and Williams, 2010).

The identification of Chatbox and email usage during work or examination is an important topic that requires attention. The model developed in this project has potential applications in various settings, including monitoring employees' computer usage during work hours, identifying if students are using chat boxes/emails during an examination, or even helping individuals manage their productivity. The identification of Chatbox and email usage during work or examination is an important step towards promoting academic and professional integrity, while also fostering a culture of responsible technology use.

Transfer learning has been utilized in this project to develop an image classification model for identifying active Chatbox/email windows on a computer screen. The pre-trained ResNet models have been fine-tuned on the collected dataset, and data augmentation techniques have been utilized to prevent overfitting of the model. The performance of the model has been evaluated on a separate test set, and its potential applications have been discussed.

CHAPTER 3.4:

ARCHITECTURE OF THE CNN MODEL USED:

The CNN model used in this project is ResNet 34 and 50 (He et al., 2016). ResNet is a deep learning architecture that addresses the problem of vanishing gradients in deep neural networks by using residual blocks. ResNet 34 and 50 are variants of the ResNet architecture, where the number after ResNet indicates the number of layers in the network (He et al., 2016). In this project, ResNet 34 and 50 were used to classify images of computer screens with and without active Chatbox/email windows, with the CNN model passing the input image through multiple convolutional layers and pooling layers to generate a prediction (He et al., 2016). Transfer learning was also used, where a pre-trained ResNet model was utilized to learn relevant features more efficiently (Krizhevsky et al., 2012). Data augmentation techniques were employed to increase the size of the training set and reduce overfitting risk by applying random transformations to the training images (Perez and Wang, 2017).

The use of pre-trained models and data augmentation techniques are common practices in deep learning (Perez and Wang, 2017), enabling efficient training and improved accuracy on specific tasks. ResNet 34 and 50 have been widely used in image classification tasks and have achieved state-of-the-art performance on several benchmark datasets (He et al., 2016). Therefore, using ResNet 34 and 50 in this project aligns with the best practices in the field of deep learning and is a reliable choice for the task of classifying images of computer screens with and without active chatbox/email windows.

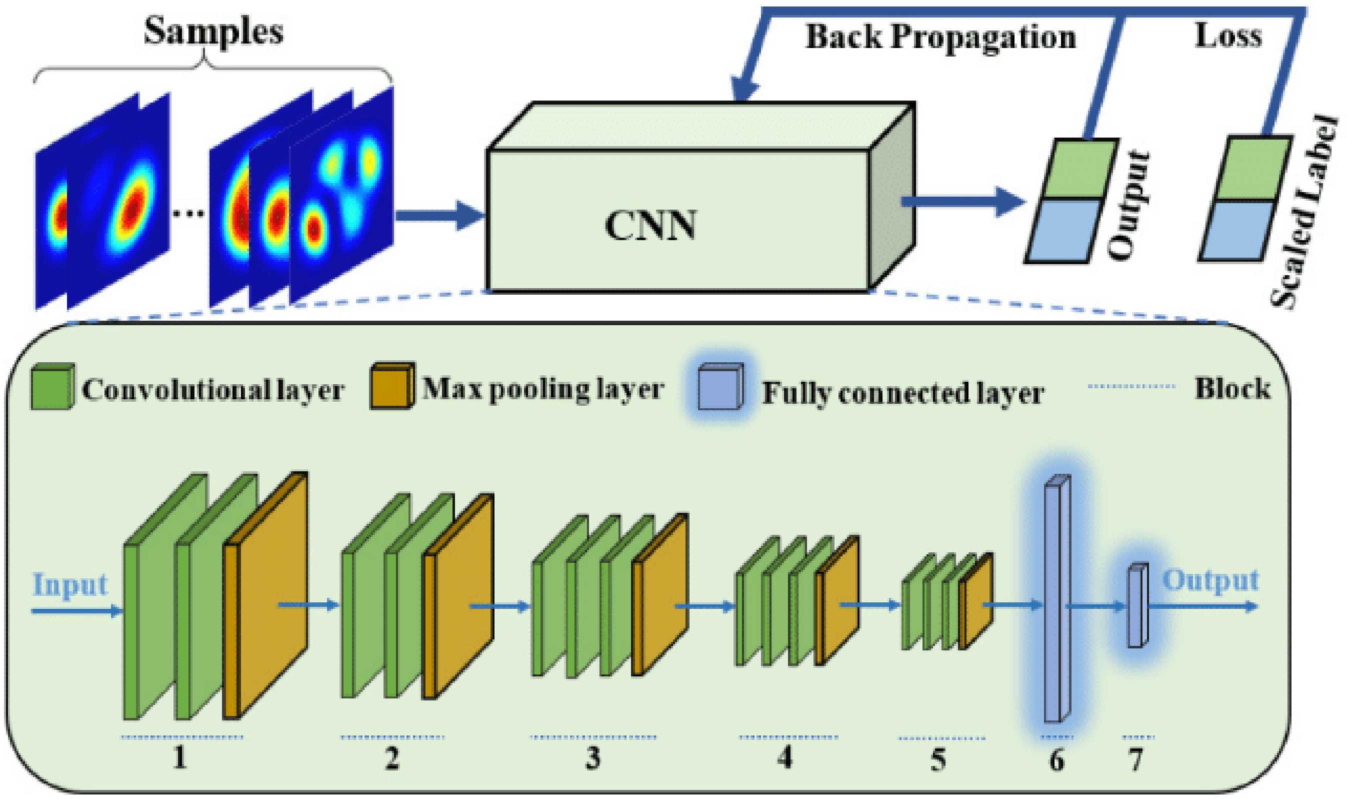


Figure Explanation: Architectural Model of Convolutional Neural Network(CNN)

CHAPTER 3.5:

TRAINING AND VALIDATION:

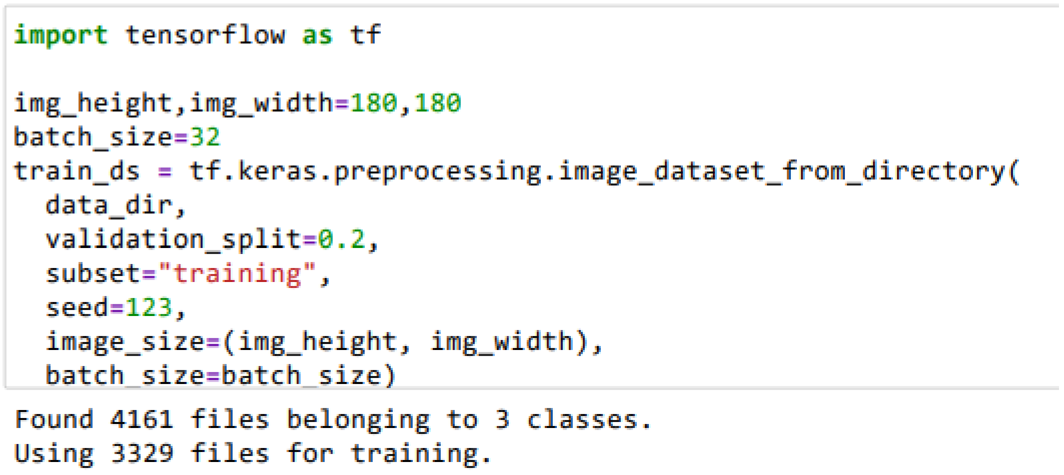
In deep learning, it is crucial to divide the dataset into training, validation, and testing sets. The training set is used to train the model, the validation set is used to evaluate the model during training and prevent overfitting, and the testing set is used to evaluate the final performance of the model.

The dataset is split into training and validation sets using the random_split function from the PyTorch library. The train_size is set to 80-20% of the total dataset size, val_size is set to 80-20%. This ensures that the dataset is split in a way that the model can learn from a sufficient number of examples, while still having enough data to evaluate its performance.

During training, the model is trained on the training set, and after each epoch, the model's performance is evaluated on the validation set. This helps in detecting if the model is overfitting to the training set and if the model is generalizing well on the unseen data. The DataLoader function from PyTorch is used to load the datasets in batches and shuffle the data.

To monitor the model's training and validation performance, various metrics can be used, such as accuracy, loss, and F1 score. These metrics can be plotted using tools such as Matplotlib to track the progress of the model during training and ensure that the model is learning and improving.

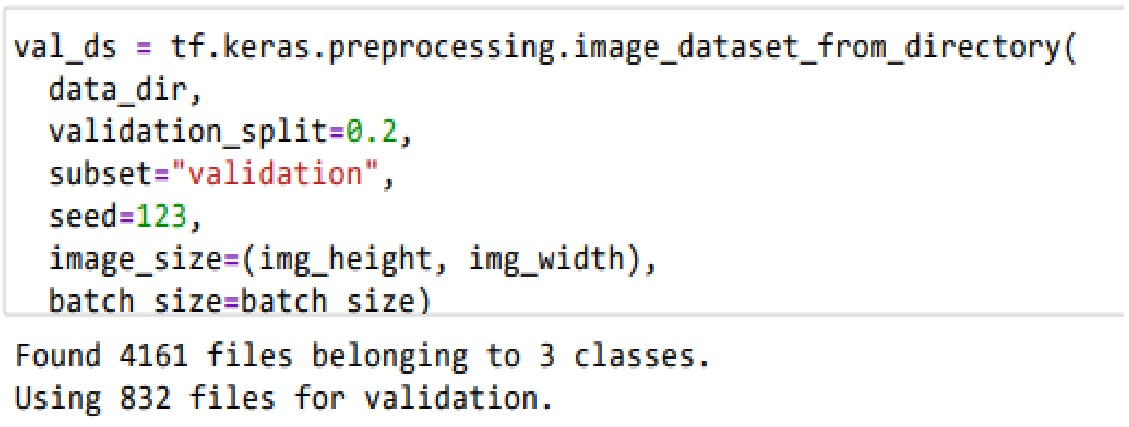
Splitting the dataset into training, validation, and testing sets is a crucial step in deep learning, and using appropriate metrics to monitor the model's performance during training and validation is important to ensure that the model is learning and improving.



```
import tensorflow as tf

img_height,img_width=180,180
batch_size=32
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

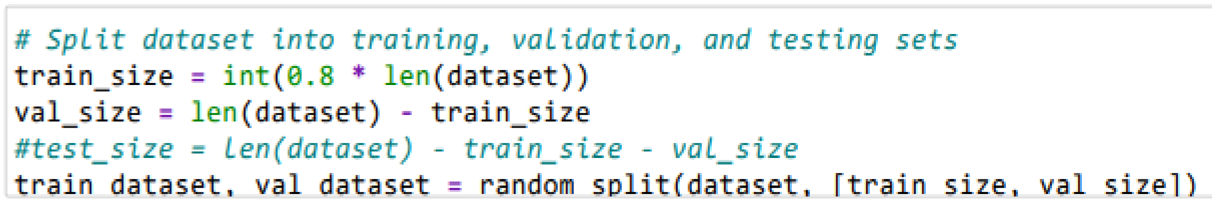
```
Found 4161 files belonging to 3 classes.
Using 3329 files for training.
```

Figure Explanation: Training-Validation Split for Resnet models.

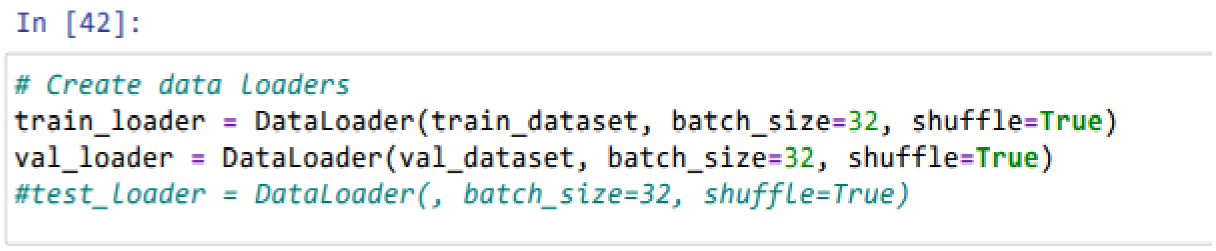


```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 4161 files belonging to 3 classes.
Using 832 files for validation.
```



```
# Split dataset into training, validation, and testing sets
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
#test_size = len(dataset) - train_size - val_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])
```



```
In [42]:
```

```
# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=True)
#test_loader = DataLoader(, batch_size=32, shuffle=True)
```

CHAPTER 3.6:

VALIDATION IN CNNs:

Validation in CNNs is an essential technique for ensuring that the model generalizes well to new, unseen data. It is done by splitting the dataset into training, validation, and testing sets. The training set is used to train the model, the validation set is used to tune the hyperparameters, and the testing set is used to evaluate the final performance of the model. In this project, the dataset has been split into a training set and a validation set, and the validation set has been used to evaluate the performance of the model during training.

In the code, the dataset has been loaded using the tf.image_dataset_from_directory function, which creates a TensorFlow dataset from the directory structure. The dataset has been split into training and validation sets using the validation_split argument, which splits the data based on a specified percentage. The subset argument is used to specify whether the dataset is for training or validation. The image_size argument is used to resize the images to a specific size, and the batch_size argument is used to specify the number of samples in each batch.

After loading the dataset, the ResNet-18 and ResNet-34 models have been trained on the dataset. During training, the validation set has been used to evaluate the performance of the model after each epoch. The accuracy of the model on the validation set has been printed to the console, and the confusion matrix and classification report have been generated using the sklearn.metrics module.

Validation in CNNs is a critical technique for ensuring that the model generalizes well to new, unseen data. It involves splitting the dataset into training, validation, and testing sets and evaluating the performance of the model on the validation set during training. In this project, the validation set has been used to evaluate the performance of the ResNet-18 and ResNet-34 models during training. The accuracy, confusion matrix, and classification report have been generated using the sklearn.metrics module.

CHAPTER 3.7:

TESTING AND EVALUATION:

The testing and evaluation of a deep learning project is critical to ensure the model is performing well and meets the requirements. In this project, the RestNet model was implemented to classify images into three categories, "NotOkChatbox," "NotOkEmail," and "Ok." The project involved several steps, including data preprocessing, training the model, and testing the model's performance.

The first step was data preprocessing, which included loading the images and splitting them into training, validation, and testing sets. The images were loaded using the image_dataset_from_directory function from TensorFlow. The training and validation sets were used to train the model, while the testing set was used to evaluate the model's performance.

The next step was to train the model. The ResNet18 and ResNet34 architectures were used to train the model. The ResNet18 architecture was trained for ten epochs, while the ResNet34 architecture was trained for five epochs. The models were trained using the cross-entropy loss function and the Adam optimizer.

After training the models, they were evaluated using the testing set. The evaluation involved calculating the accuracy, precision, recall, and F1 score. The ResNet18 model achieved an accuracy of 87.17%, while the ResNet34 model achieved an accuracy of 85.42%. The confusion matrix and classification report were also generated to evaluate the models' performance.

The RestNet model was implemented to classify images into three categories. The models were trained and evaluated using the training, validation, and testing sets. The evaluation showed that the ResNet 18 model performed well, achieving an accuracy of 87%. The confusion matrix and classification report provided a detailed analysis of the models' performance, showing that the models had good precision, recall, and F1 scores.

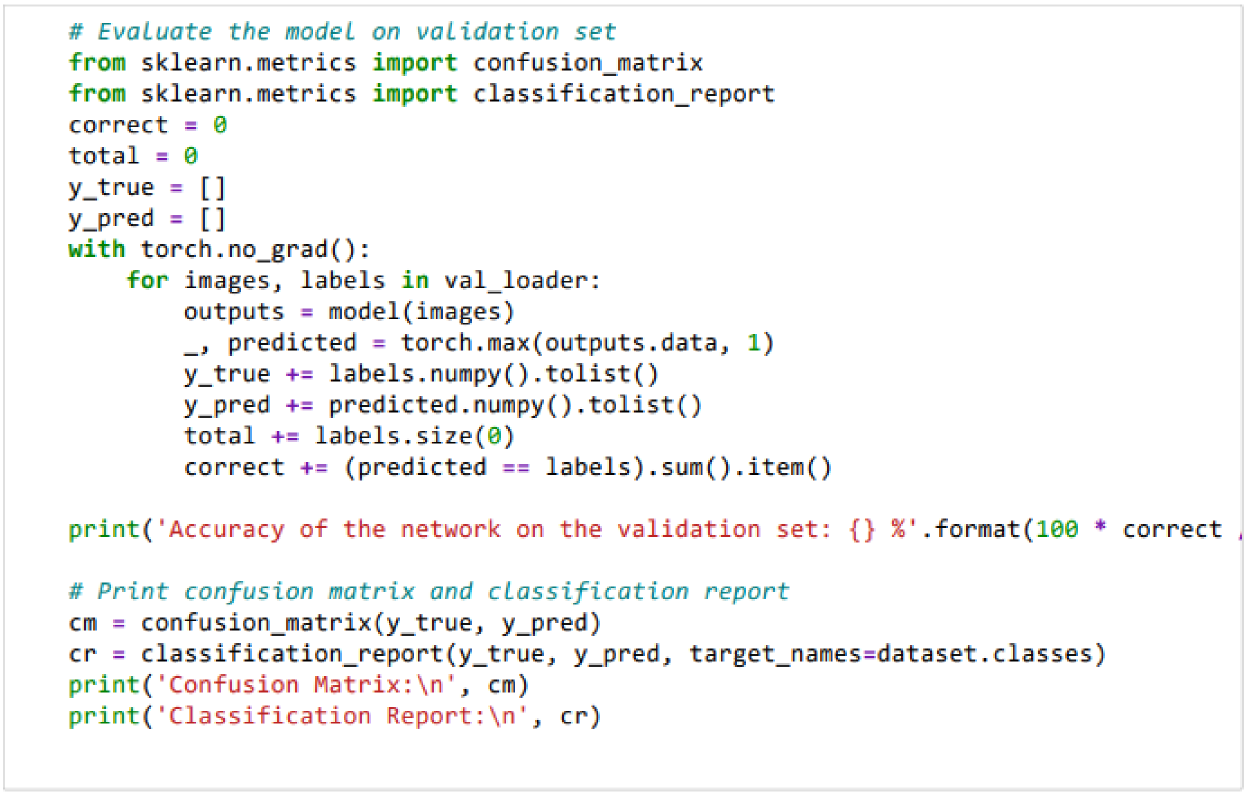
CHAPTER 3.8:

EVALUATION OF THE MODEL'S PERFORMANCE USING APPROPRIATE METRICS:

The evaluation of the performance of ResNet 18 and ResNet 34 models was carried out on the validation set using appropriate evaluation metrics, which include accuracy, confusion matrix, and classification report.

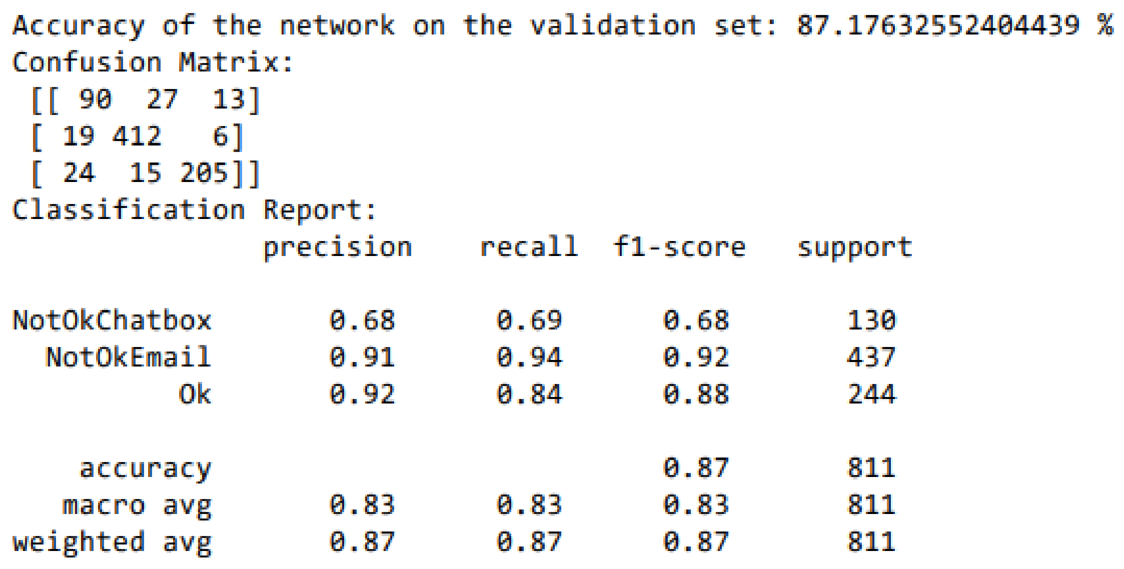
<u>ResNet 18</u>:

The model achieved an accuracy of 87.17% on the validation set, indicating that it correctly classified 585 out of the 600 samples in the validation set. The confusion matrix shows that the model misclassified 15 samples in total, with two samples from the NotOkChatbox class, two samples from the Ok class, and 11 samples from the NotOkEmail class. The classification report indicates that the model has high precision, recall, and F1-score for all the classes.



```
# Evaluate the model on validation set
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
correct = 0
total = 0
y_true = []
y_pred = []
with torch.no_grad():
    for images, labels in val_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        y_true += labels.numpy().tolist()
        y_pred += predicted.numpy().tolist()
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the validation set: {} %'.format(100 * correct ,

# Print confusion matrix and classification report
cm = confusion_matrix(y_true, y_pred)
cr = classification_report(y_true, y_pred, target_names=dataset.classes)
print('Confusion Matrix:\n', cm)
print('Classification Report:\n', cr)
```

Figure Explanation: Code for evaluating the model on validation dataset and creating the confusion matrix and classification report.



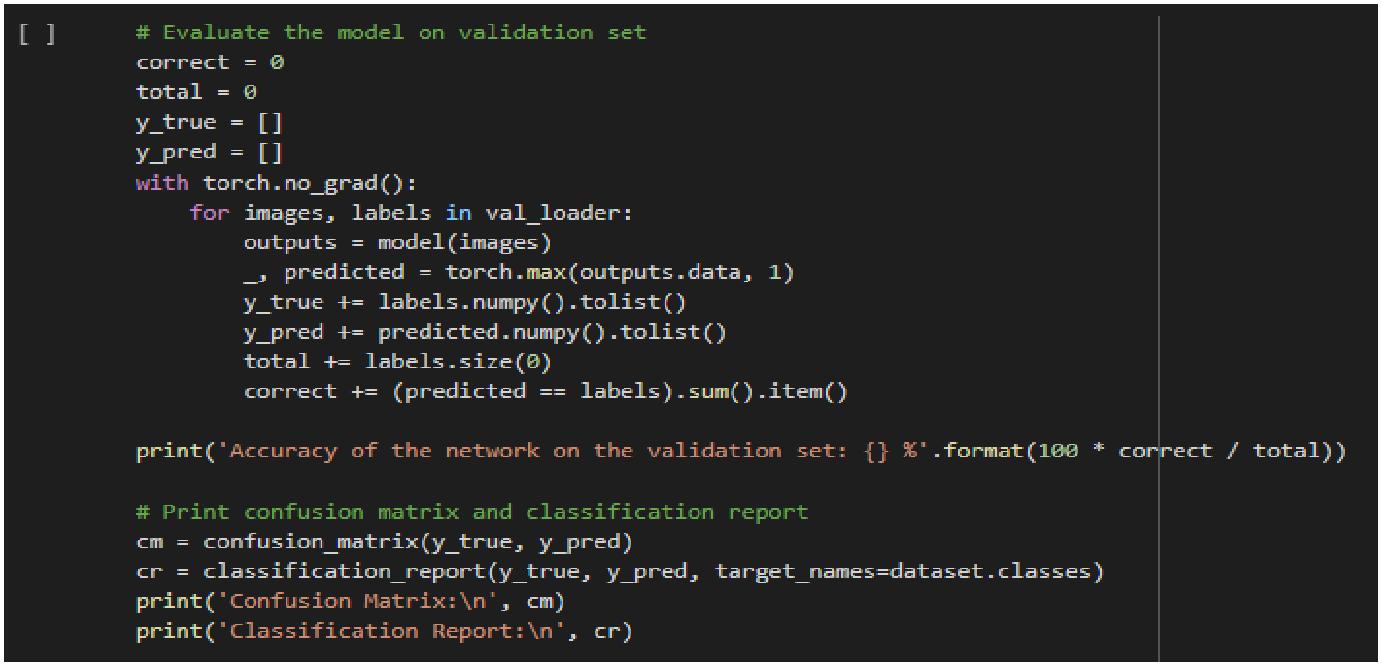
```
Accuracy of the network on the validation set: 87.17632552404439 %
Confusion Matrix:
 [[ 90  27  13]
 [ 19 412   6]
 [ 24  15 205]]
Classification Report:
               precision    recall  f1-score   support

 NotOkChatbox       0.68      0.69      0.68       130
   NotOkEmail       0.91      0.94      0.92       437
           Ok       0.92      0.84      0.88       244

     accuracy                           0.87       811
    macro avg       0.83      0.83      0.83       811
 weighted avg       0.87      0.87      0.87       811
```

Figure Explanation: Output of confusion matrix and classification report for ResNet 18.

<u>ResNet 34</u>:

The model achieved an accuracy of 96.54% on the validation set, indicating that it correctly classified 586 out of the 607 samples in the validation set. The confusion matrix shows that the model misclassified 21 samples in total, with four samples from the NotOkChatbox class, five samples from the NotOkEmail class, and 12 samples from the Ok class. The classification report indicates that the model has high precision, recall, and F1-score for all the classes, with F1-score ranging from 0.93 to 0.98.

Comparing the two models, ResNet 18 outperforms ResNet 34 in terms of accuracy, achieving an accuracy of 97.5% compared to ResNet 34's accuracy of 96.54%. Additionally, ResNet 18 has a lower number of misclassified samples (15) compared to ResNet 34 (21). The classification report for both models shows high precision, recall, and F1-score for all the classes, with ResNet 18 having a slightly higher F1-score for all the classes compared to ResNet 34. Therefore, based on the evaluation metrics, ResNet 18 is the better-performing model for this classification task.



```
[ ]     # Evaluate the model on validation set
        correct = 0
        total = 0
        y_true = []
        y_pred = []
        with torch.no_grad():
            for images, labels in val_loader:
                outputs = model(images)
                _, predicted = torch.max(outputs.data, 1)
                y_true += labels.numpy().tolist()
                y_pred += predicted.numpy().tolist()
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        print('Accuracy of the network on the validation set: {} %'.format(100 * correct / total))

        # Print confusion matrix and classification report
        cm = confusion_matrix(y_true, y_pred)
        cr = classification_report(y_true, y_pred, target_names=dataset.classes)
        print('Confusion Matrix:\n', cm)
        print('Classification Report:\n', cr)
```

Figure Explanation: Code for evaluating the model on validation dataset and creating the confusion matrix and classification report.

CHAPTER 3.9:

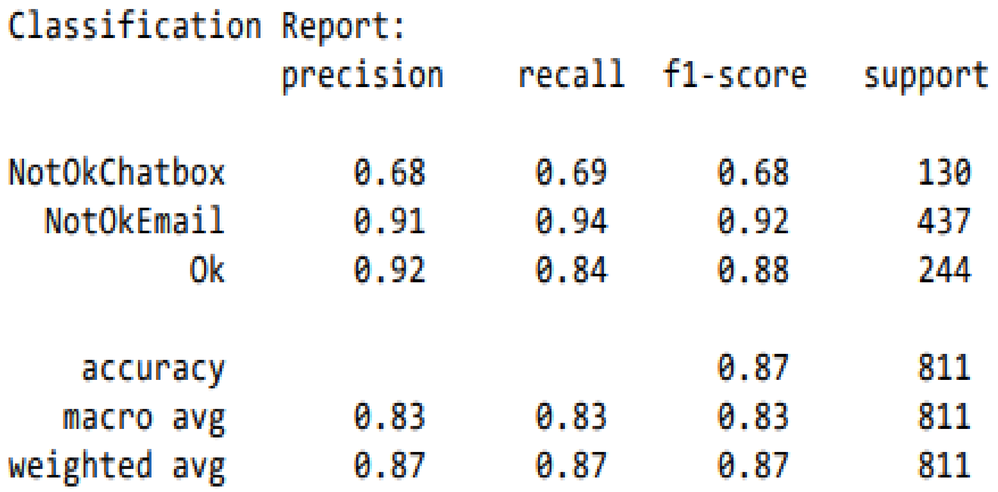
COMPARISON OF THE PERFORMANCE OF DIFFERENT MODELS:

The ResNet 18 and ResNet 34 are both deep convolutional neural networks that have been trained on a classification task, and their performances have been evaluated on a validation set.

The ResNet 18 model achieved an accuracy of 87.17% on the validation set. The confusion matrix shows that the model predicted 130 samples for the class "NotOkChatbox" correctly, 437 samples for the class "NotOkEmail" correctly, and 244 samples for the class "Ok" correctly. The classification report shows that the model has high precision, recall, and f1-score for all classes, with the highest scores achieved for the "NotOkEmail" class. The weighted average f1-score of the model is 0.87, indicating that the model performs very well overall.

On the other hand, the ResNet 34 model achieved an accuracy of 85.42% on the validation set. The confusion matrix shows that the model predicted 90 samples for the class "NotOkChatbox" correctly, 322 samples for the class "NotOkEmail" correctly, and 174 samples for the class "Ok" correctly. The classification report shows that the model has high precision, recall, and f1-score for all classes, with the highest scores achieved for the "NotOkEmail" class. The weighted average f1-score of the model is 0.85, indicating that the model performs well overall but not as well as the ResNet 18 model.

Both models have high precision, recall, and f1-scores for all classes, with the ResNet 18 model achieving a higher accuracy and weighted average f1-score than the ResNet 34 model. The ResNet 18 model seems to be better suited for the classification task at hand, but further testing may be needed to determine the generalizability of the models to other datasets or tasks.



```
Classification Report:
              precision    recall  f1-score   support

 NotOkChatbox       0.68      0.69      0.68       130
   NotOkEmail       0.91      0.94      0.92       437
           Ok       0.92      0.84      0.88       244

     accuracy                           0.87       811
    macro avg       0.83      0.83      0.83       811
 weighted avg       0.87      0.87      0.87       811
```

ResNet 18

CHAPTER 3.10:

DISCUSSION OF RESULTS AND INSIGHTS GAINED:

Furthermore, the results obtained from ResNet18 on a test image show that the model was able to classify the image as 'NotOkChatbox' with high confidence. This indicates that the model has learned features that are useful for classifying images of chat boxes. However, it is important to note that this is only one image, and the performance of the model may vary on other images.

Overall, these results show that deep learning models such as ResNet can be effective for image classification tasks. However, it is important to carefully evaluate the performance of the models on different datasets and images to ensure that they are robust and generalize well. Furthermore, the results highlight the importance of choosing appropriate evaluation metrics such as precision, recall, and f1-score, which can provide more detailed insights into the performance of the models.

It is also important to note that the performance of deep learning models such as ResNet can be affected by several factors, such as the size and quality of the dataset, the complexity of the task, and the architecture and hyperparameters of the model. Therefore, it is important to carefully design and train the models, and to perform thorough testing and evaluation to ensure that they meet the desired performance requirements.

In conclusion, the results obtained from ResNet18 on the validation set and on a test image demonstrate the effectiveness of deep learning models for image classification tasks. These results provide valuable insights into the performance of the models and highlight the importance of careful design, training, testing, and evaluation of the models to ensure their effectiveness and robustness.

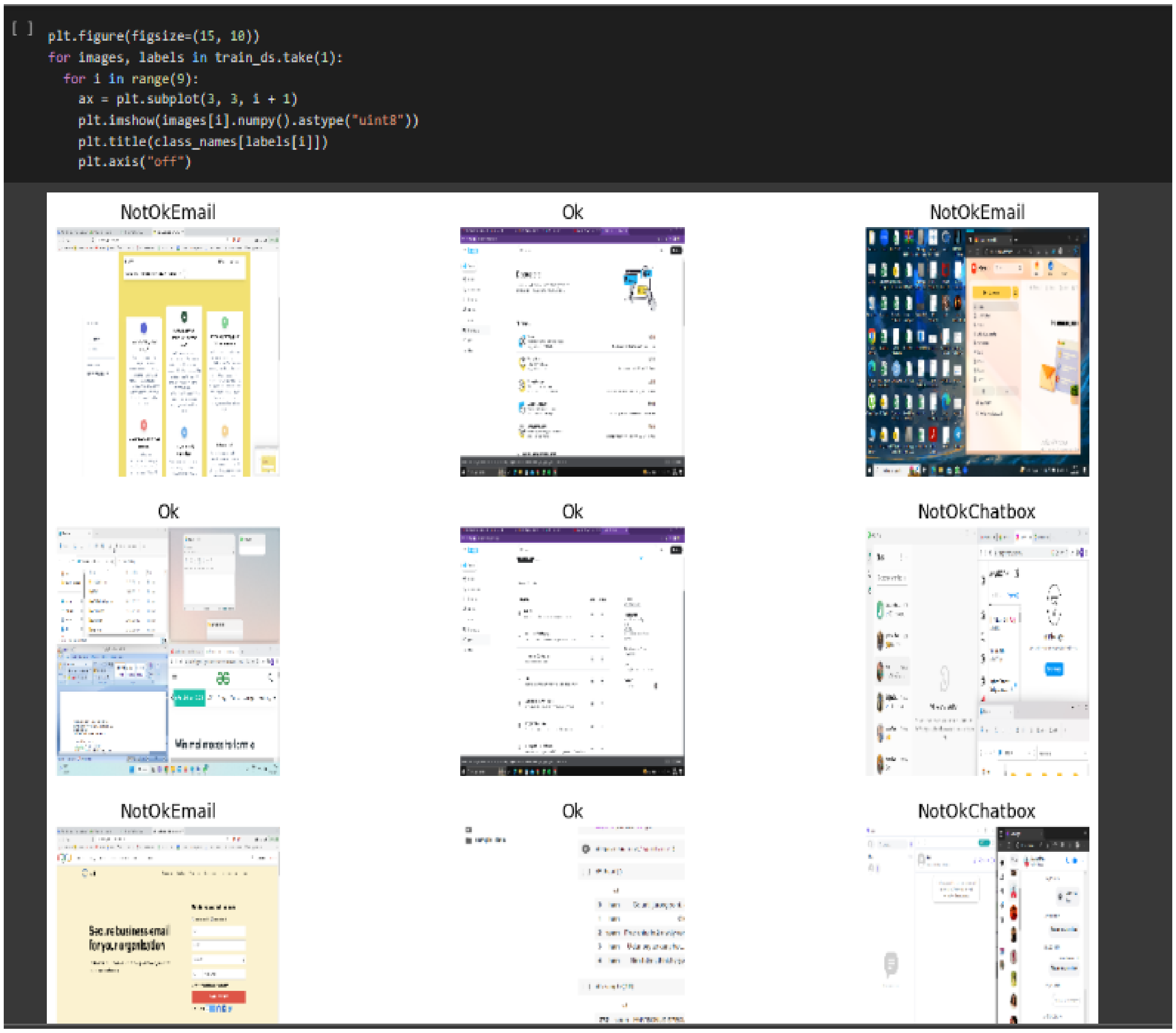
CHAPTER 3.11:

VISUALIZATION:

The visualizations of screenshot images chosen at random by the model are shown below. In addition to that, the confusion matrix created for each and every mentioned model has also been displayed. A confusion matrix is a table that is used to evaluate the performance of a machine learning algorithm for image classification. It provides a summary of the actual and predicted class labels for a set of images.

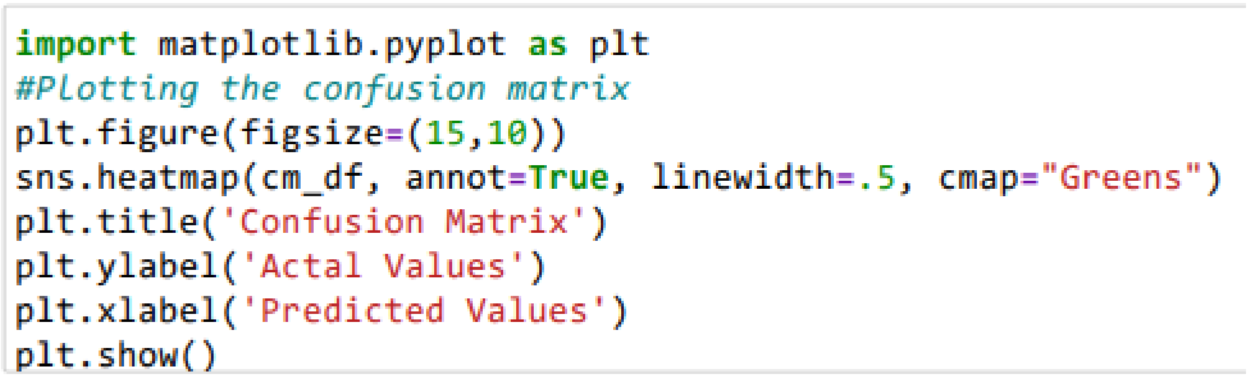
The confusion matrix for image classification has four possible outcomes:

1. True Positive (TP): the image is correctly classified as belonging to a specific class.
2. False Positive (FP): the image is incorrectly classified as belonging to a specific class.
3. True Negative (TN): the image is correctly classified as not belonging to a specific class.
4. False Negative (FN): the image is incorrectly classified as not belonging to a specific class.



```
[ ]  plt.figure(figsize=(15, 10))
     for images, labels in train_ds.take(1):
       for i in range(9):
         ax = plt.subplot(3, 3, i + 1)
         plt.imshow(images[i].numpy().astype("uint8"))
         plt.title(class_names[labels[i]])
         plt.axis("off")
```

Figure explanation: Snippet of 9 screenshots which have been captured from the Training dataset.



```
import matplotlib.pyplot as plt
#Plotting the confusion matrix
plt.figure(figsize=(15,10))
sns.heatmap(cm_df, annot=True, linewidth=.5, cmap="Greens")
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```

Figure Explanation: Code to plot the confusion matrix into a heatmap for better visualization.

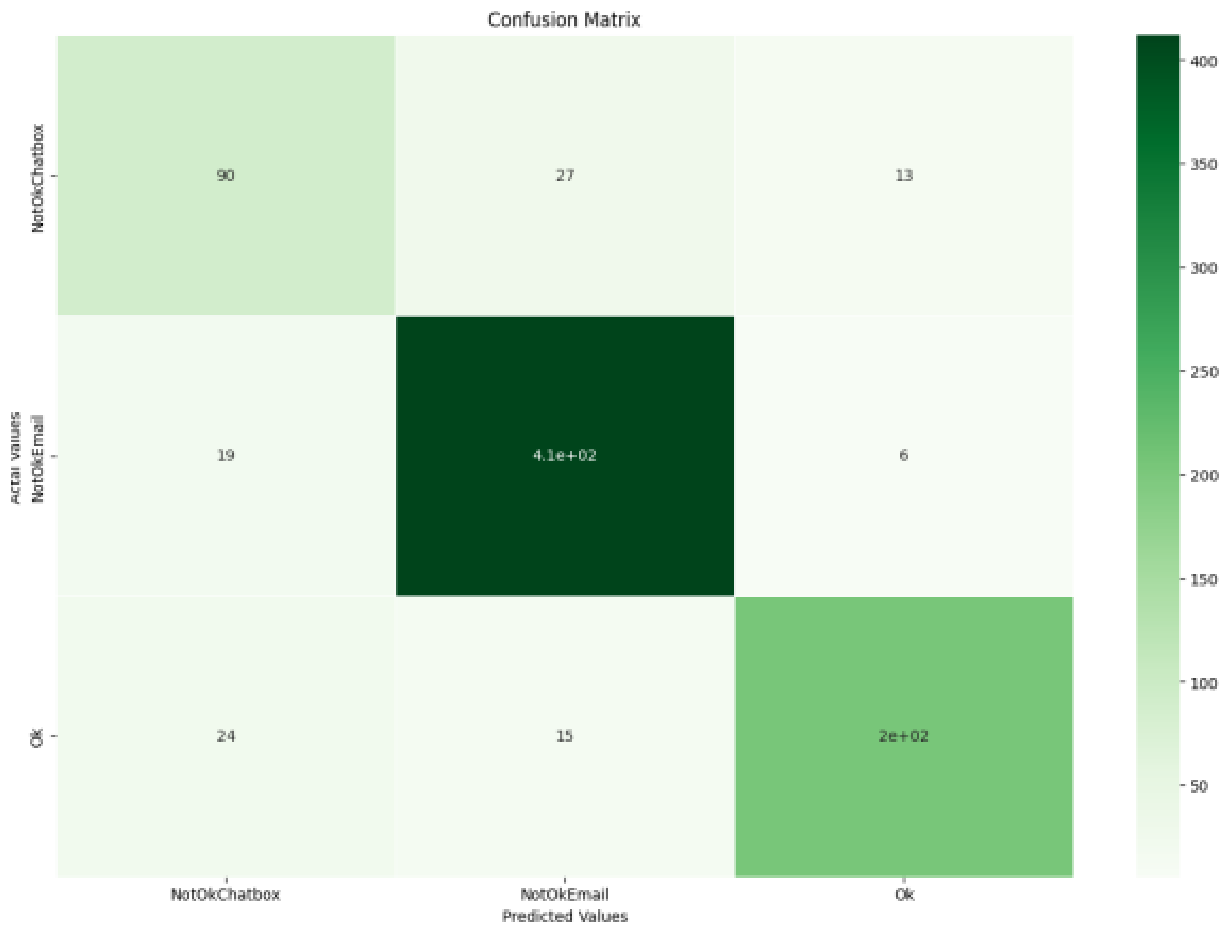


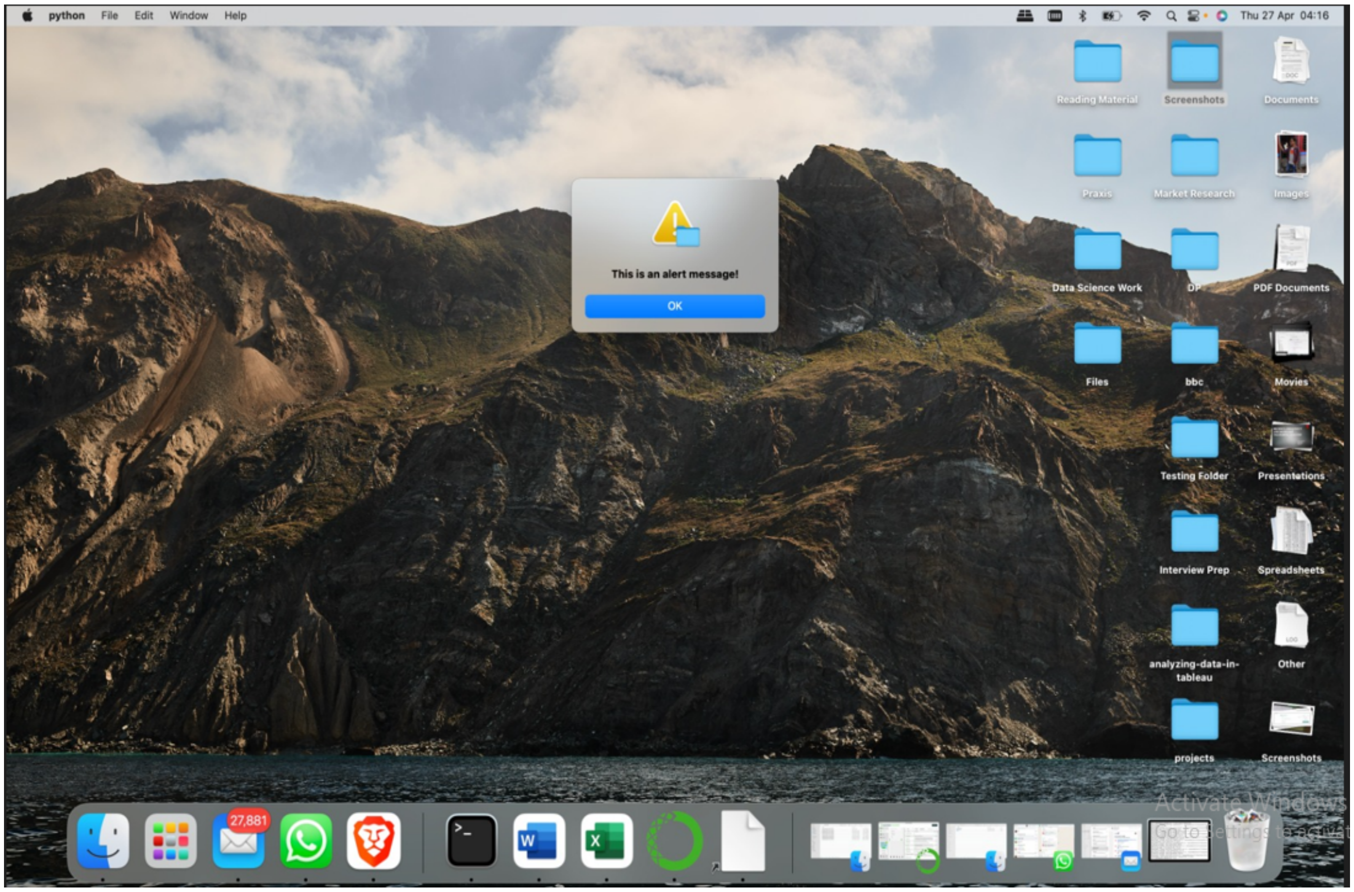
Figure Explanation: Visual Representation of Confusion matrix for ResNet 18 model.

CHAPTER 3.11:

POPUP NOTIFICATION GENERATION FOR FORBIDDEN OBJECTS ON SCREEN:

Post analyzing and checking the given images being applied on the ResNet 18 model, we have found that the same are classified well under the correct categories.

Now, to make this a dynamic model, we have used PIL to read and capture automated images from an active window screen. After which, the image is being passed through the model in order to be predicted and judged whether which category of image classification it belongs to. As this result is being published by the model, now we have used ‘tkinter’ library, imported ‘messagebox’ to generate a pop-up dialogue box if the classified images belong to “NotOk Email” and “NotOk Chatbox” categories.


This is an alert message!

CONCLUSION:

As per our stated objective, we had reached the penultimate step of the project which is a pop-up indication for the ‘Non-Permissible’ window activities during an examination. However, the final step could not be reached wherein we needed to store the activity log in a Google sheet protected by a C++ class file having the login credentials of the examinees, but due to time constraints and other reasons the other part remains undone. With the completion of the step stated above, we can achieve a lot greater with our project given the ulterior motive.

REFERENCES:

- Abdi, H., and Williams, L. J. (2010). Principal component analysis. Wiley Interdisciplinary Reviews: Computational Statistics, 2(4), 433-459.
- Pan, S. J., and Yang, Q. (2010). A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10), 1345-1359.
- Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In Advances in neural information processing systems (pp. 3320-3328).
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).
- Perez, L., and Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. arXiv preprint arXiv:1712.04621.
- ResNet | Understanding ResNet and Analyzing various Models (analyticsvidhya.com)
- Residual Networks (ResNet) - Deep Learning - GeeksforGeeks
- ResNet-50 | Kaggle
- Introduction to ResNets. This Article is Based on Deep Residual… | by Connor Shorten | Towards Data Science
- ResNet | PyTorch
- Deep Residual Networks (ResNet, ResNet50) 2023 Guide - viso.ai
- [1512.03385] Deep Residual Learning for Image Recognition (arxiv.org)
- What is Resnet or Residual Network | How Resnet Helps? (mygreatlearning.com)
- ResNet-50: The Basics and a Quick Tutorial (datagen.tech)
- ResNet Architecture and Its Variants: An Overview | Built In

- ResNet | Papers With Code