University of Prince Mugrin

College of Computer and Cyber Sciences

Department of Artificial Intelligence

AI417- Introduction to Deep Learning

Course Project – Semester II (Spring 2023)

Students' Emotion Detection Model

Team Members:

Aisha Ahmad | 4010058

Salwa Shama | 4010405

Samah Shama | 4010403

Sana Shama | 4010404

Instructor:

Dr. Ahmed Elhayek

May 21, 2023

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Overview of The Project

To achieve UPM's objective of providing excellent academic programs and a supportive learning environment for students, a deep learning-based model for emotion detection using face recognition can be designed and developed. This model can help monitor student engagement and emotional responses to learning materials during lectures. Additionally, it can be used for statistical research to analyse student reactions to various activities and events organized by different entities at UPM (e.g., clubs) to rank them based on their positive influence.

Overall, this project will help the instructors gain insights into the effectiveness of their teaching methods and adjust them as needed. In addition, it offers objective measures of emotional responses, allowing for better-informed decision-making regarding future actions. In the end, this will lead to in a more productive and interesting learning environment for UPM students.

CHAPTER 2: DATASET

2.1 Data Description

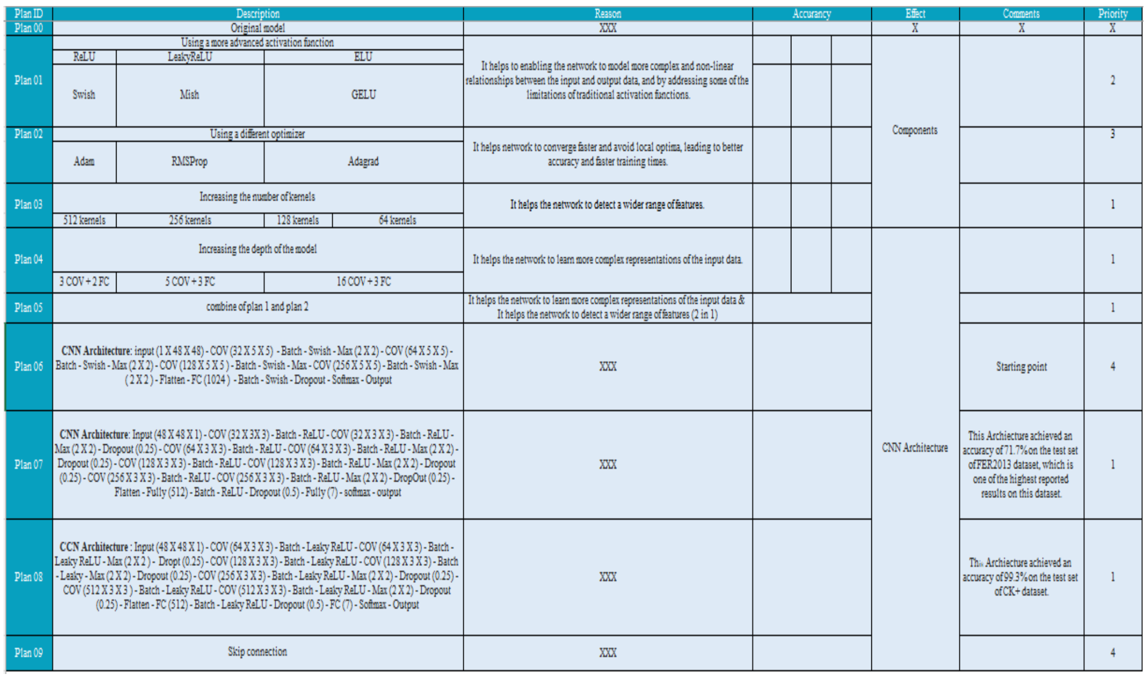
There are many popular data sets on the Web that can be used for facial expression recognition; in our case, we used two different kinds of data sets. The first one is FER2013, which contains 28709 examples in the training set and 3589 in the validation set. This dataset divides the emotion into seven feeling expressions: 0 = anger, 1 = disgust, 2 = fear, 3 = happy, 4 = sad, 5 = surprise, and 6 = neutral [1]. The second one is CKPLUS, which contains 981 grayscale training face images and has the same 7 categories of feeling expressions as the FER2013 datasets [2].

CHAPTER 3: DEVELOPMENT MODEL PLAN

3.1 Developing Plan

AI model building planning is key to ensuring that the developing effort, the spent time, and the consumed resources are well organized. It is also a super important piece of the puzzle for delivering high-accuracy models.

By the way, there is no silver bullet for building models because it tightly depends on the project itself and the dataset. In our project "Students' Emotion Detection ", we will be following this plan: As we can see, this plan consists of two parts, one related to model components and another related to model architecture. In the first part, we collected the most styles and components that have achieved high accuracy whatever for emotion detection or similar applications. The second part is about CNN architectures that have achieved high accuracy whatever by using the same dataset or a different one.



Plan ID	Description				Reason	Accurancy			Effect	Comments	Priority
Plan 00	Original model				XXX				X	X	X
Plan 01	Using a more advanced activation function				It helps to enabling the network to model more complex and non-linear relationships between the input and output data, and by addressing some of the limitations of traditional activation functions.				Components		2
	ReLU	LeakyReLU	ELU								
	Swish	Mish	GELU								
Plan 02	Using a different optimizer				It helps network to converge faster and avoid local optima, leading to better accuracy and faster training times.						3
	Adam	RMSProp	Adagrad								
Plan 03	Increasing the number of kernels				It helps the network to detect a wider range of features.						1
	512 kernels	256 kernels	128 kernels	64 kernels							
Plan 04	Increasing the depth of the model				It helps the network to learn more complex representations of the input data.				CNN Architecture		1
	3 COV + 2 FC	5 COV + 3 FC	16 COV + 3 FC								
Plan 05	combine of plan 1 and plan 2				It helps the network to learn more complex representations of the input data & It helps the network to detect a wider range of features (2 in 1)						1
Plan 06	**CNN Architecture**: input (1 X 48 X 48) - COV (32 X 5 X 5) - Batch - Swish - Max (2 X 2) - COV (64 X 5 X 5) - Batch - Swish - Max (2 X 2) - COV (128 X 5 X 5) - Batch - Swish - Max - COV (256 X 5 X 5) - Batch - Swish - Max (2 X 2) - Flatten - FC (1024) - Batch - Swish - Dropout - Softmax - Output				XXX					Starting point	4
Plan 07	**CNN Architecture**: Input (48 X 48 X 1) - COV (32 X 3X 3) - Batch - ReLU - COV (32 X 3 X 3) - Batch - ReLU - Max (2 X 2) - Dropout (0.25) - COV (64 X 3 X 3) - Batch - ReLU - COV (64 X 3 X 3) - Batch - ReLU - Max (2 X 2) - Dropout (0.25) - COV (128 X 3 X 3) - Batch - ReLU - COV (128 X 3 X 3) - Batch - ReLU - Max (2 X 2) - Dropout (0.25) - COV (256 X 3 X 3) - Batch - ReLU - COV (256 X 3 X 3) - Batch - ReLU - Max (2 X 2) - DropOut (0.25) - Flatten - Fully (512) - Batch - ReLU - Dropout (0.5) - Fully (7) - softmax - output				XXX					This Archiecture achieved an accuracy of 71.7%on the test set of FER2013 dataset, which is one of the highest reported results on this dataset.	1
Plan 08	**CCN Architecture** : Input (48 X 48 X 1) - COV (64 X 3 X 3) - Batch - Leaky ReLU - COV (64 X 3 X 3) - Batch - Leaky ReLU - Max (2 X 2) - Dropt (0.25) - COV (128 X 3 X 3) - Batch - Leaky ReLU - COV (128 X 3 X 3) - Batch - Leaky - Max (2 X 2) - Dropout (0.25) - COV (256 X 3 X 3) - Batch - Leaky ReLU - Max (2 X 2) - Dropout (0.25) - COV (512 X 3 X 3) - Batch - Leaky ReLU - COV (512 X 3 X 3) - Batch - Leaky ReLU - Max (2 X 2) - Dropout (0.25) - Flatten - FC (512) - Batch - Leaky ReLU - Dropout (0.5) - FC (7) - Softmax - Output				XXX					This Archiecture achieved an accuracy of 99.3%on the test set of CK+ dataset.	1
Plan 09	Skip connection				XXX						4

Figure 3.1 – Developing Plan

CHAPTER 4: BULID THE MODEL

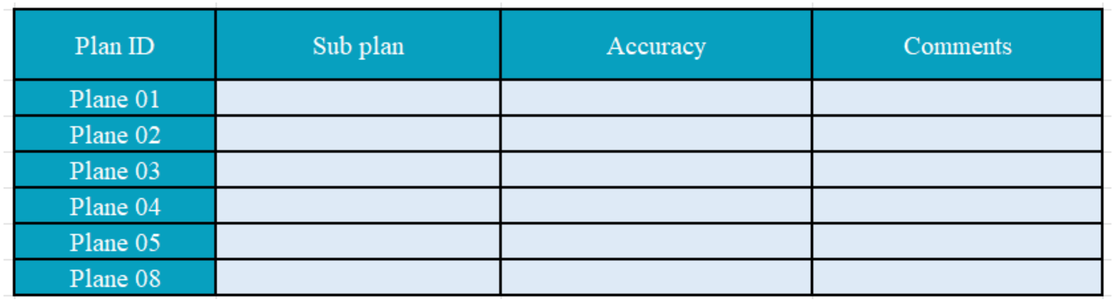
Applying our plan to FER 2013, we got the following results: As you can see, unfortunately, the accuracy is not very high. After searching, we found that the highest accuracy that is achieved by this dataset by using CNN is 73.28%, and there is a new version of this dataset called FER2013+ that addresses some issues with FER2013. Even though we had some problems training this data, we played with the CNN architecture, and the highest result we got was XXX. For more detail on the training result, visit Chapter 5: Analysis and Discussion [3].

Plan ID	Sub plan	Accuracy	Comments
Plane 01			
Plane 02			
Plane 03			
Plane 04			
Plane 05			
Plane 08			

Figure 4.1 – FER2013 Training Results

As we can see in Figure 4.1, yes, not all plans are applied because, at the beginning, we felt there was something wrong and the model accuracy didn't improve a lot, and we figured out the reason that we mentioned earlier. As they say "You don't need to drink the entire ocean to confirm that it is salty". These few plans were enough for us to discover we had a problem.

The same plan was applied to the CKPLUS dataset, and as we can see, there are better results in the training, with the highest accuracy being XXX, but honestly, in the testing data, the results are not very good, and that was expected for the small amount of data we trained the model on. For more details on training results, visit Chapter 5: Analysis and Discussion.



Plan ID	Sub plan	Accuracy	Comments
Plane 01			
Plane 02			
Plane 03			
Plane 04			
Plane 05			
Plane 08			

Figure 4.1 – CKPLUS Training Results

Before wrapping up this chapter, it must be mentioned that we tried many architectural model styles with different combinations, and those were just samples.

CHAPTER 5: ANALYSIS AND DISCUSSION

This chapter is the thunder of this report. We will discuss what we have learned from this project by going through the following topics: Successful Design Features and Reasons, Unsuccessful Design Features and Reasons, Additional Design Features and Improvements, Reasons for Applying Some Changes and Change's Reason that Resolved the Issue.

5.1 Successful Design Features and Reasons

XXX..

5.2 Unsuccessful Design Features and Reasons

XXX..

5.3 Additional Design Features and Improvements

XXX..

5.4 Reasons for Applying Some Changes

XXX..

5.5 Change's Reason that Resolved the Issue

XXX..

CHAPTER 6: CONCLUSION

6.1 Conclusion

In conclusion, we applied our knowledge of deep learning to our project. We began by writing a comprehensive training plan to re-architect the model of CNN by going through fully connected layers, depth neural networks, and convolutional neural network concepts. Throughout the project, we utilized the techniques and concepts we learned in our lectures and improved our understanding through hands-on application. Also, we learned something very important: even if you have a good architecture model, that doesn't mean you will get high accuracy because both models and the cleans of data are highly coupled together. As Abraham Lincom said, "Give me six hours to chop down a tree, and I will spend the first four sharpening the axe", The same story can be used in preparing the data and then training the model. Overall, this project allowed us to gain practical experience and reinforce our understanding of the course material.

6.2 References

[1] M. Sambare, “Fer-2013,” Kaggle, https://www.kaggle.com/datasets/msambare/fer2013 (accessed May 21, 2023).

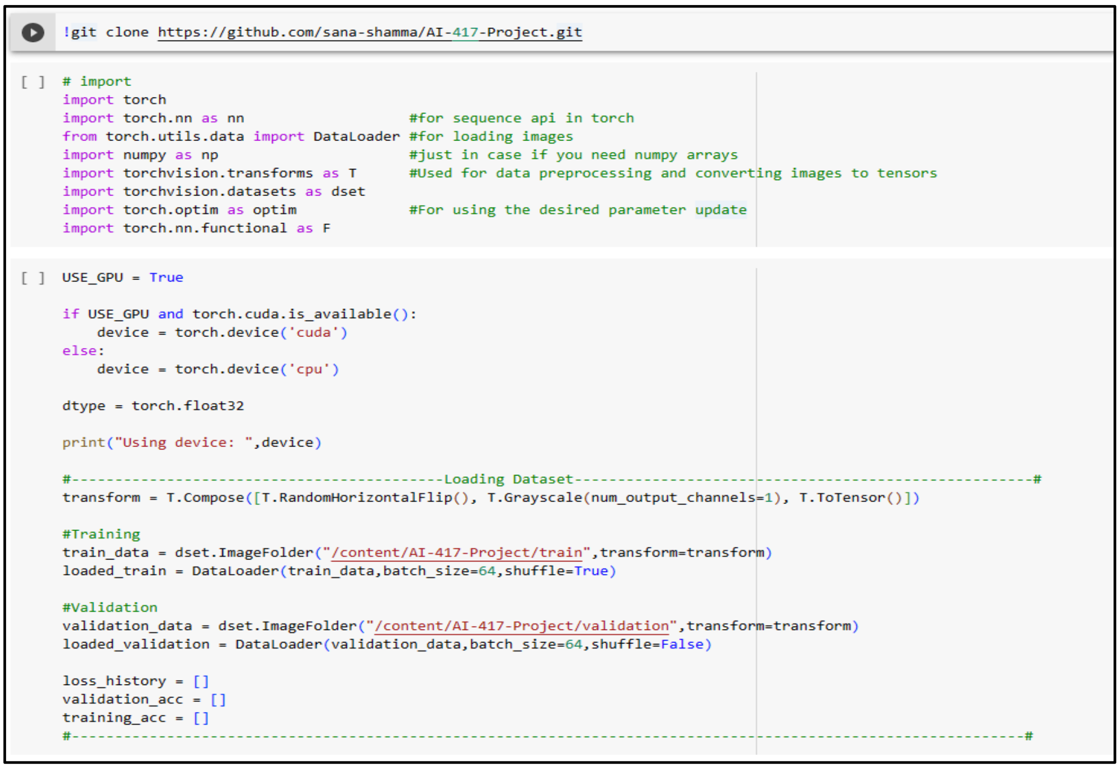
[2] A. Shawon, “CKPLUS,” Kaggle, https://www.kaggle.com/datasets/shawon10/ckplus?datasetId=65125&amp;sortBy=voteCount (accessed May 21, 2023).

[3] Y. Khaireddin and Z. Chen, “Facial emotion recognition: State of the art performance on FER2013,” arXiv.org, https://arxiv.org/abs/2105.03588 (accessed May 21, 2023).

APPENDICES

1. Students’ Emotion Detection Code

Below is the source code for our students’ emotion detection program, which is implemented in Python.

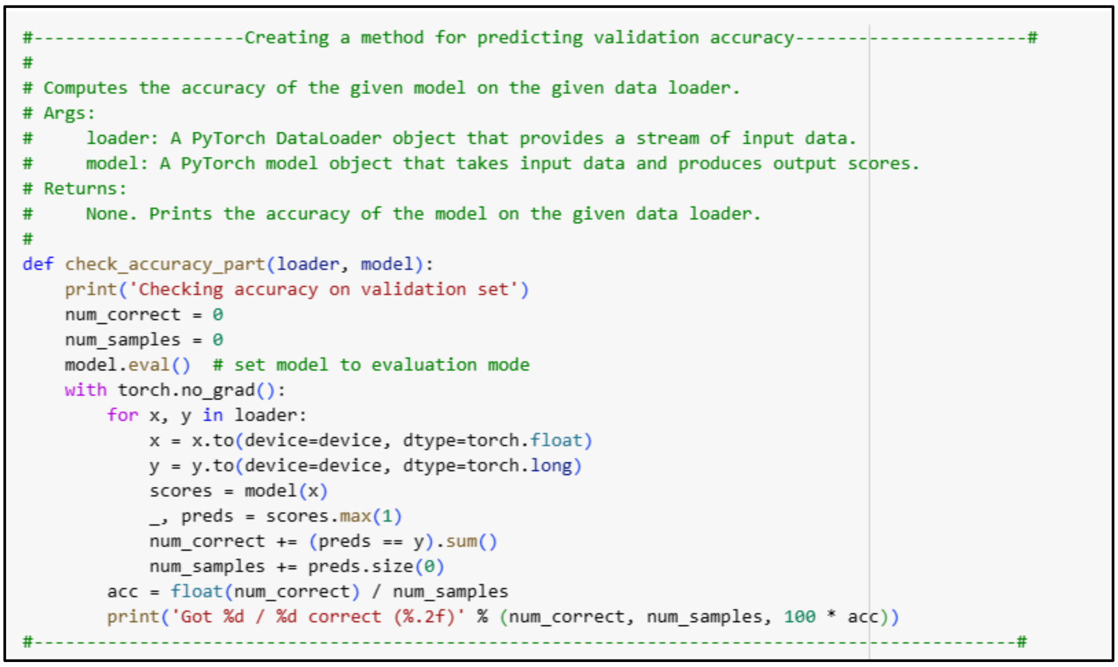


```
!git clone https://github.com/sana-shamma/AI-417-Project.git
```

```
[ ] # import
    import torch
    import torch.nn as nn                      #for sequence api in torch
    from torch.utils.data import DataLoader #for loading images
    import numpy as np                         #just in case if you need numpy arrays
    import torchvision.transforms as T         #Used for data preprocessing and converting images to tensors
    import torchvision.datasets as dset
    import torch.optim as optim                #For using the desired parameter update
    import torch.nn.functional as F
```

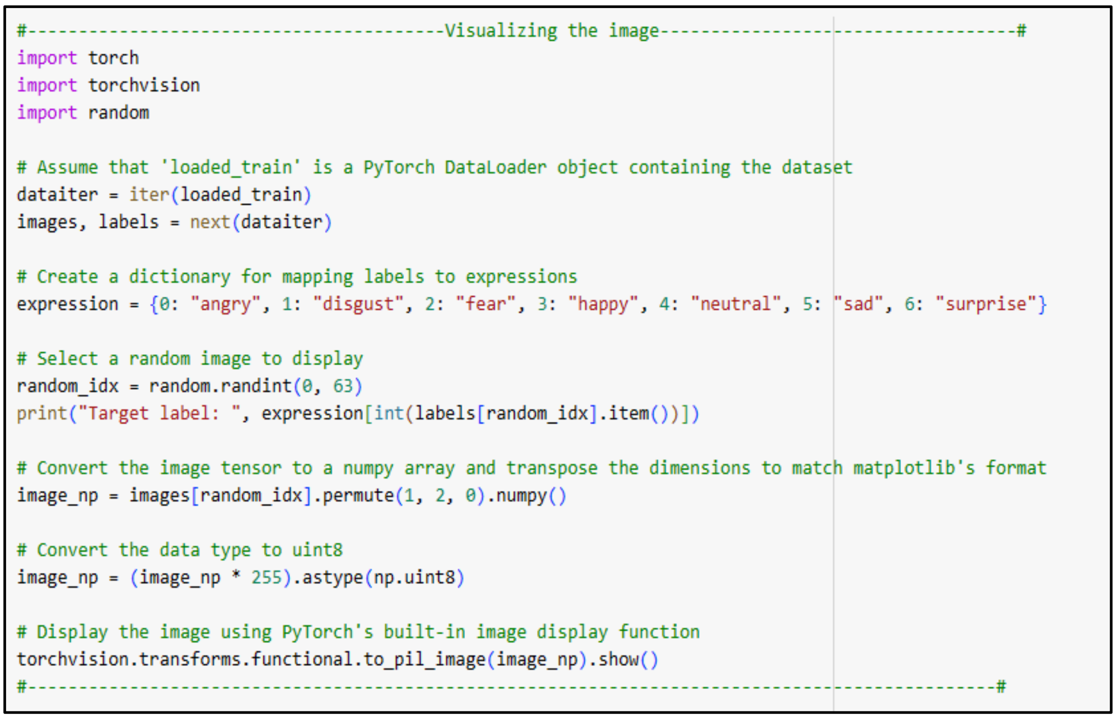
```
[ ] USE_GPU = True

    if USE_GPU and torch.cuda.is_available():
        device = torch.device('cuda')
    else:
        device = torch.device('cpu')

    dtype = torch.float32

    print("Using device: ",device)

    #--------------------------------------------Loading Dataset--------------------------------------------------------#
    transform = T.Compose([T.RandomHorizontalFlip(), T.Grayscale(num_output_channels=1), T.ToTensor()])

    #Training
    train_data = dset.ImageFolder("/content/AI-417-Project/train",transform=transform)
    loaded_train = DataLoader(train_data,batch_size=64,shuffle=True)

    #Validation
    validation_data = dset.ImageFolder("/content/AI-417-Project/validation",transform=transform)
    loaded_validation = DataLoader(validation_data,batch_size=64,shuffle=False)

    loss_history = []
    validation_acc = []
    training_acc = []
    #-----------------------------------------------------------------------------------------------------------------#
```

Figure 1 – Loading Dataset



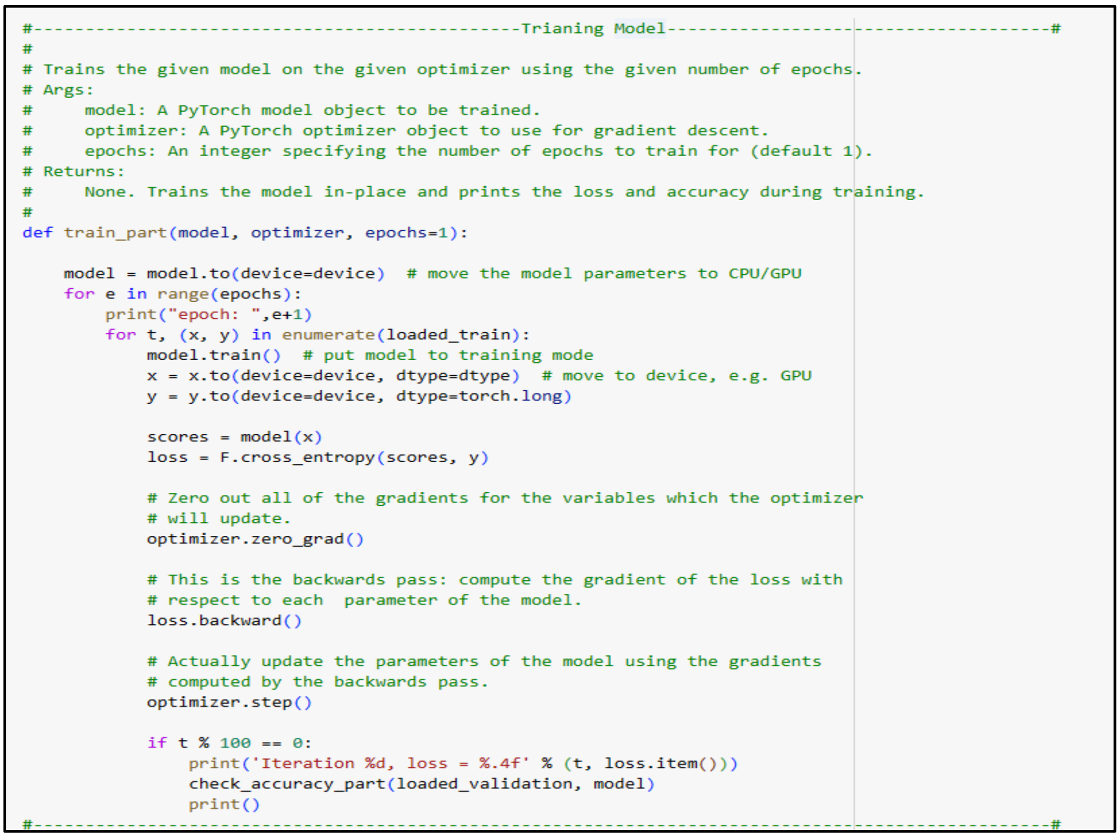
```
#--------------------Creating a method for predicting validation accuracy---------------------#
#
# Computes the accuracy of the given model on the given data loader.
# Args:
#     loader: A PyTorch DataLoader object that provides a stream of input data.
#     model: A PyTorch model object that takes input data and produces output scores.
# Returns:
#     None. Prints the accuracy of the model on the given data loader.
#
def check_accuracy_part(loader, model):
    print('Checking accuracy on validation set')
    num_correct = 0
    num_samples = 0
    model.eval()  # set model to evaluation mode
    with torch.no_grad():
        for x, y in loader:
            x = x.to(device=device, dtype=torch.float)
            y = y.to(device=device, dtype=torch.long)
            scores = model(x)
            _, preds = scores.max(1)
            num_correct += (preds == y).sum()
            num_samples += preds.size(0)
        acc = float(num_correct) / num_samples
        print('Got %d / %d correct (%.2f)' % (num_correct, num_samples, 100 * acc))
#--------------------------------------------------------------------------------------------#
```

Figure 2 – Validation Function



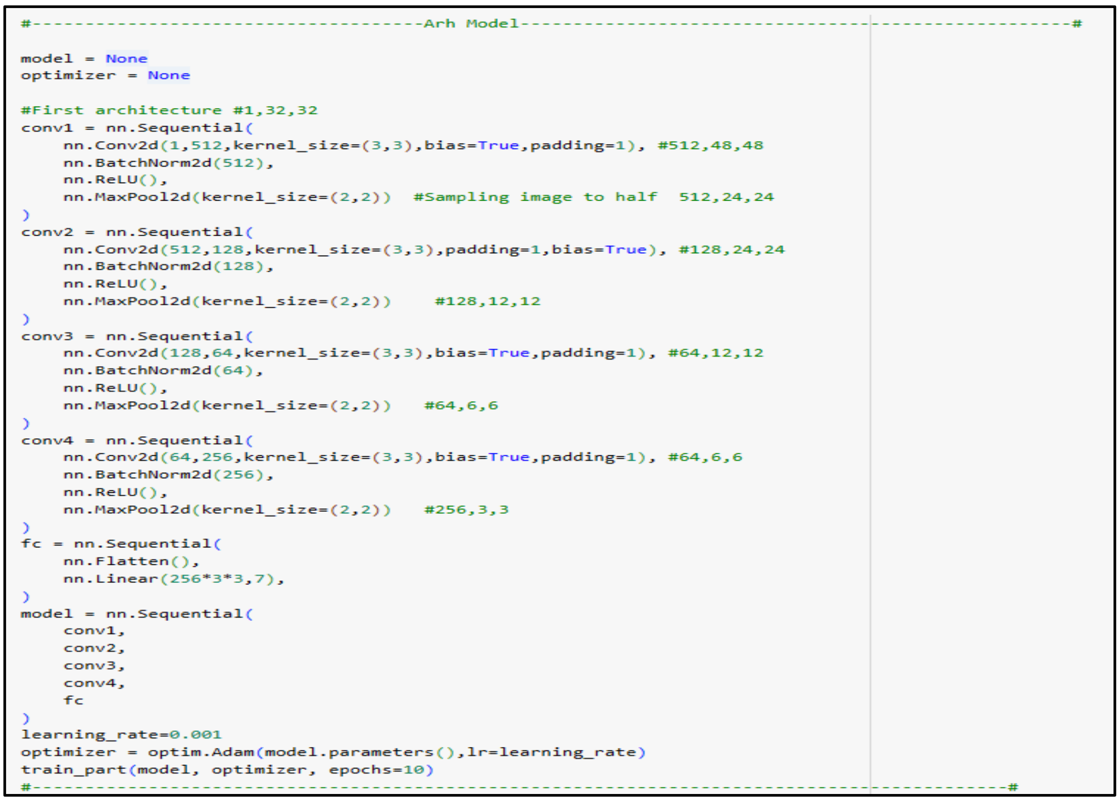
```
#----------------------------------------Visualizing the image------------------------------------#
import torch
import torchvision
import random

# Assume that 'loaded_train' is a PyTorch DataLoader object containing the dataset
dataiter = iter(loaded_train)
images, labels = next(dataiter)

# Create a dictionary for mapping labels to expressions
expression = {0: "angry", 1: "disgust", 2: "fear", 3: "happy", 4: "neutral", 5: "sad", 6: "surprise"}

# Select a random image to display
random_idx = random.randint(0, 63)
print("Target label: ", expression[int(labels[random_idx].item())])

# Convert the image tensor to a numpy array and transpose the dimensions to match matplotlib's format
image_np = images[random_idx].permute(1, 2, 0).numpy()

# Convert the data type to uint8
image_np = (image_np * 255).astype(np.uint8)

# Display the image using PyTorch's built-in image display function
torchvision.transforms.functional.to_pil_image(image_np).show()
#-------------------------------------------------------------------------------------------------#
```

Figure 3 – Visualizing Images



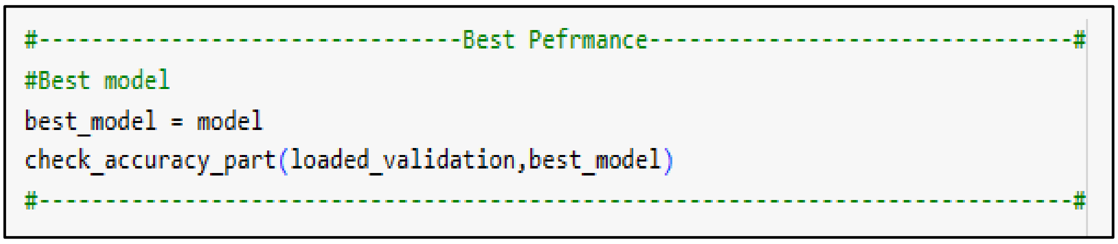
```
#-----------------------------------------------Trianing Model--------------------------------------#
#
# Trains the given model on the given optimizer using the given number of epochs.
# Args:
#     model: A PyTorch model object to be trained.
#     optimizer: A PyTorch optimizer object to use for gradient descent.
#     epochs: An integer specifying the number of epochs to train for (default 1).
# Returns:
#     None. Trains the model in-place and prints the loss and accuracy during training.
#
def train_part(model, optimizer, epochs=1):

    model = model.to(device=device)  # move the model parameters to CPU/GPU
    for e in range(epochs):
        print("epoch: ",e+1)
        for t, (x, y) in enumerate(loaded_train):
            model.train()  # put model to training mode
            x = x.to(device=device, dtype=dtype)  # move to device, e.g. GPU
            y = y.to(device=device, dtype=torch.long)

            scores = model(x)
            loss = F.cross_entropy(scores, y)

            # Zero out all of the gradients for the variables which the optimizer
            # will update.
            optimizer.zero_grad()

            # This is the backwards pass: compute the gradient of the loss with
            # respect to each  parameter of the model.
            loss.backward()

            # Actually update the parameters of the model using the gradients
            # computed by the backwards pass.
            optimizer.step()

            if t % 100 == 0:
                print('Iteration %d, loss = %.4f' % (t, loss.item()))
                check_accuracy_part(loaded_validation, model)
                print()
#---------------------------------------------------------------------------------------------------#
```

Figure 4 – Training Model



```
#------------------------------------Arh Model-----------------------------------------------------#

model = None
optimizer = None

#First architecture #1,32,32
conv1 = nn.Sequential(
    nn.Conv2d(1,512,kernel_size=(3,3),bias=True,padding=1), #512,48,48
    nn.BatchNorm2d(512),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=(2,2))  #Sampling image to half  512,24,24
)
conv2 = nn.Sequential(
    nn.Conv2d(512,128,kernel_size=(3,3),padding=1,bias=True), #128,24,24
    nn.BatchNorm2d(128),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=(2,2))    #128,12,12
)
conv3 = nn.Sequential(
    nn.Conv2d(128,64,kernel_size=(3,3),bias=True,padding=1), #64,12,12
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=(2,2))   #64,6,6
)
conv4 = nn.Sequential(
    nn.Conv2d(64,256,kernel_size=(3,3),bias=True,padding=1), #64,6,6
    nn.BatchNorm2d(256),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=(2,2))   #256,3,3
)
fc = nn.Sequential(
    nn.Flatten(),
    nn.Linear(256*3*3,7),
)
model = nn.Sequential(
    conv1,
    conv2,
    conv3,
    conv4,
    fc
)
learning_rate=0.001
optimizer = optim.Adam(model.parameters(),lr=learning_rate)
train_part(model, optimizer, epochs=10)
#---------------------------------------------------------------------------------------------#
```

Figure 5 – Model Arch



```
#-------------------------------Best Pefrmance-------------------------------#
#Best model
best_model = model
check_accuracy_part(loaded_validation,best_model)
#---------------------------------------------------------------------------#
```

Figure 6 – Model Accuracy