Nairit Das 24MAI0097

Title: Driver Behavior Detection Using CNN & AlexNet

Description: This notebook demonstrates the use of Convolutional Neural Networks (CNNs) for detecting and classifying driver behavior to enhance road safety. It begins with an exploratory analysis of the dataset, followed by data preprocessing, augmentation, and CNN implementation. Models like AlexNet are employed to classify driver behaviors into various categories based on input images. The process includes feature extraction, model architecture design, and training.

Dataset Exploration: Insightful analysis of the dataset to understand the distribution of behaviors.
Preprocessing and Augmentation: Techniques such as resizing, normalization, and augmentation to enhance the dataset for better model performance.
CNN Implementation: Step-by-step creation of CNN architectures tailored for behavior classification.
AlexNet Integration: Leveraging AlexNet for feature extraction and classification due to its proven effectiveness in image-based tasks.
Performance Metrics: Evaluation of the model using accuracy, precision, recall, and confusion matrices to ensure robust results.
Applications: Practical use cases, such as real-time driver monitoring for road safety and accident prevention.

Code:

Nairit Das 24MAI0097

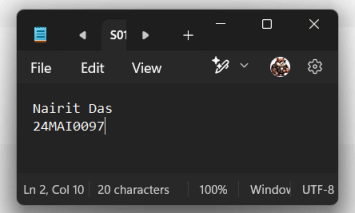
Driver Safety Prediction

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[3] !kaggle datasets download robinreni/revitsone-5class
```

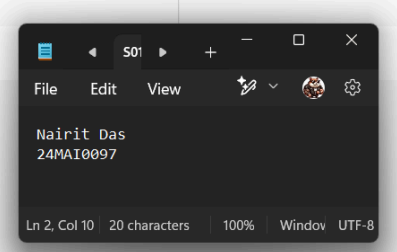
```
Dataset URL: https://www.kaggle.com/datasets/robinreni/revitsone-5class
License(s): DbCL-1.0
Downloading revitsone-5class.zip to /content
100% 981M/983M [00:34<00:00, 34.5MB/s]
100% 983M/983M [00:34<00:00, 29.9MB/s]
```




```
[4] import zipfile
    import os

    # Define the path to your downloaded zip file
    zip_file_path = '/content/revitsone-5class.zip'  # Change this to the actual path of your zip file

    # Define the destination folder for extraction
    extraction_folder = '/content/drive/MyDrive/DATA2/'  # Change this if you want to extract to a different folder

    # Check if the destination folder exists, if not, create it
    if not os.path.exists(extraction_folder):
```

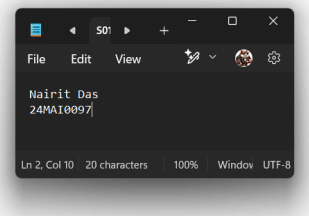
```
[4] extraction_folder = '/content/drive/MyDrive/DATA2/'  # Change this if you want to extract to a different folder

    # Check if the destination folder exists, if not, create it
    if not os.path.exists(extraction_folder):
        os.makedirs(extraction_folder)

    # Extract the downloaded zip file
    with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
        zip_ref.extractall(extraction_folder)

    print('Dataset extraction complete.')
```

```
Dataset extraction complete.
```



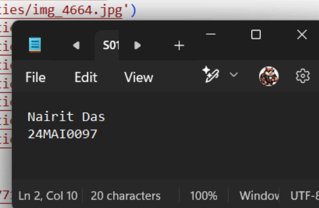

```
[5] import warnings
    warnings.filterwarnings("ignore")
    from numpy import asarray
    import numpy as np
    import pandas as pd
    from PIL import Image
    import cv2
    import glob
    import os
    import random
    import subprocess
    import matplotlib.pyplot as plt
    from skimage.io import imread
    from matplotlib.patches import Rectangle
    import tensorflow as tf
```

```
image_list_other = []
image_list_safe = []
image_list_talking = []
image_list_text = []
image_list_turn = []

# Update the base path to where the dataset was extracted
base_path = '/content/drive/MyDrive/DATA2/Revitsone-5classes/'

for other in os.listdir(os.path.join(base_path, "other_activities")):
    if other.endswith(".png") or other.endswith(".jpg"):
        image_list_other.append(os.path.join(base_path, "other_activities", other))
        print(os.path.join(base_path, "other_activities", other))

for safe in os.listdir(os.path.join(base_path, "safe_driving")):
    if safe.endswith(".png") or safe.endswith(".jpg"):
        image_list_safe.append(os.path.join(base_path, "safe_driving", safe))
        print(os.path.join(base_path, "safe_driving", safe))

for talking in os.listdir(os.path.join(base_path, "talking_phone")):
    if talking.endswith(".png") or talking.endswith(".jpg"):
        image_list_talking.append(os.path.join(base_path, "talking_phone", talking))
        print(os.path.join(base_path, "talking_phone", talking))

for text in os.listdir(os.path.join(base_path, "texting_phone")):
    if text.endswith(".png") or text.endswith(".jpg"):
        image_list_text.append(os.path.join(base_path, "texting_phone", text))
        print(os.path.join(base_path, "texting_phone", text))
```



```
for text in os.listdir(os.path.join(base_path, "texting_phone")):
    if text.endswith(".png") or text.endswith(".jpg"):
        image_list_text.append(os.path.join(base_path, "texting_phone", text))
        print(os.path.join(base_path, "texting_phone", text))

for turn in os.listdir(os.path.join(base_path, "turning")):
    if turn.endswith(".png") or turn.endswith(".jpg"):
        image_list_turn.append(os.path.join(base_path, "turning", turn))
        print(os.path.join(base_path, "turning", turn))
```

```
[7] # Update the paths to match the actual paths in the lists.
    # The following example assumes that the base path is correct
    # but the other parts of the path are incorrect

    # Corrected paths for image_list_other
    image_list_other.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/other_activities/img_79.jpg')
    image_list_other.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/other_activities/img_4664.jpg')
    image_list_other.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/other_activiti
    image_list_other.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/other_activiti
    image_list_other.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/other_activiti
    image_list_other.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/other_activiti
    image_list_other.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/other_activiti
    image_list_other.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/other_activiti
    image_list_other.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/other_activiti

    # Corrected paths for image_list_turn
    image_list_turn.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/turning/img_877
    image_list_turn.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/turning/img_62337.jpg')
    image_list_turn.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/turning/img_67523.jpg')
    image_list_turn.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/turning/img_70552.jpg')
    image_list_turn.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/turning/img_84605.jpg')
    image_list_turn.remove('/content/drive/MyDrive/DATA2/Revitsone-5classes/turning/img_101434.jpg')
```

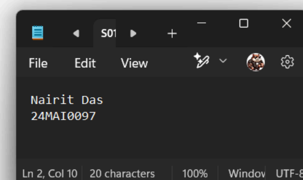



```
[8] font = {'family':'Times New Roman','color':'#1f211f'}
    background_color = '#fab72f'
```

```
[9] plt.figure(1, figsize=(15, 9))
```


```
[8] font = {'family':'Times New Roman','color':'#1f211f'}
    background_color = '#fab72f'
```

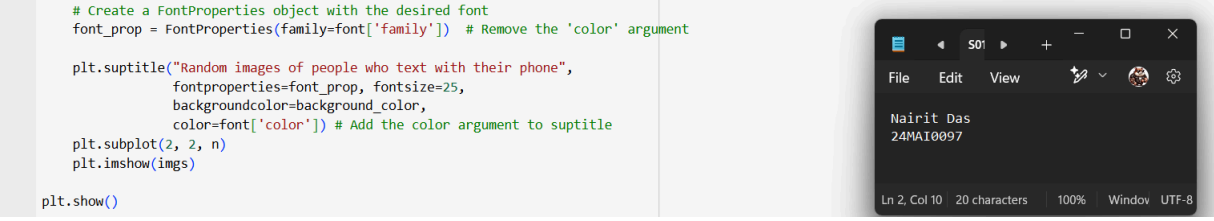

```
[9] plt.figure(1, figsize=(15, 9))

    # Adjust the position of the title above the subplots
    plt.subplots_adjust(top=0.85)  # Adjust the top margin for title spacing
    plt.axis('off')

    n = 0

    # Add the title and position it above the subplots
    plt.text(
        0.5, 0.93, "Random images of people who talk with their phone",
        ha='center', va='center', transform=plt.gcf().transFigure,
        fontsize=25,
        bbox=dict(facecolor='lightgray', alpha=0.8)
    )

    # Loop to add images in the subplots
    for i in range(4):
        n += 1
        random_img = random.choice(image_list_talking)
        imgs = imread(random_img)
        plt.subplot(2, 2, n)
        plt.imshow(imgs)
        plt.axis('off')  # Remove axis labels and ticks from subplots
```

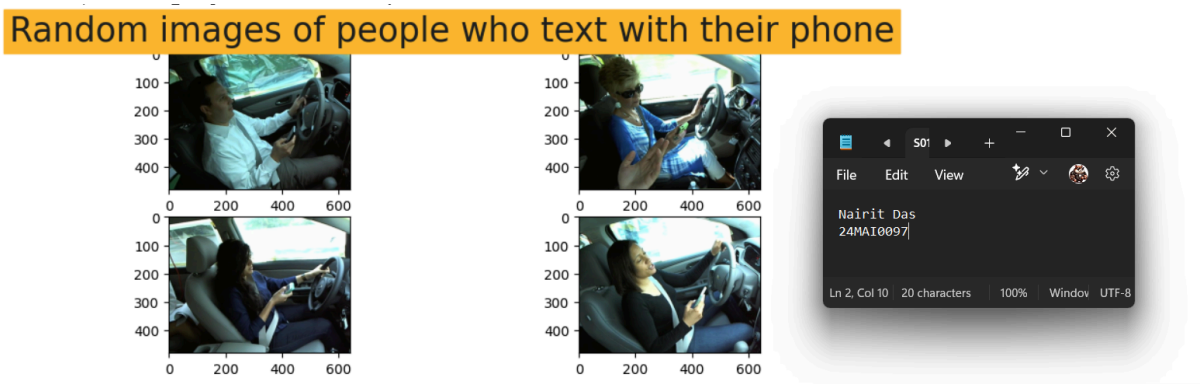


Nairit Das 24MAI0097

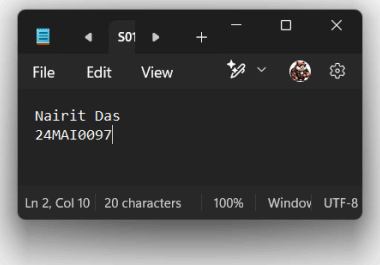
```
plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()
```


```
from matplotlib.font_manager import FontProperties

plt.figure(1, figsize=(15, 9))
plt.axis('off')
n = 0
for i in range(4):
    n += 1
    random_img = random.choice(image_list_text)
    imgs = imread(random_img)

    # Create a FontProperties object with the desired font
    font_prop = FontProperties(family=font['family'])  # Remove the 'color' argument

    plt.suptitle("Random images of people who text with their phone",
                 fontproperties=font_prop, fontsize=25,
                 backgroundcolor=background_color,
                 color=font['color']) # Add the color argument to suptitle
    plt.subplot(2, 2, n)
    plt.imshow(imgs)

plt.show()
```

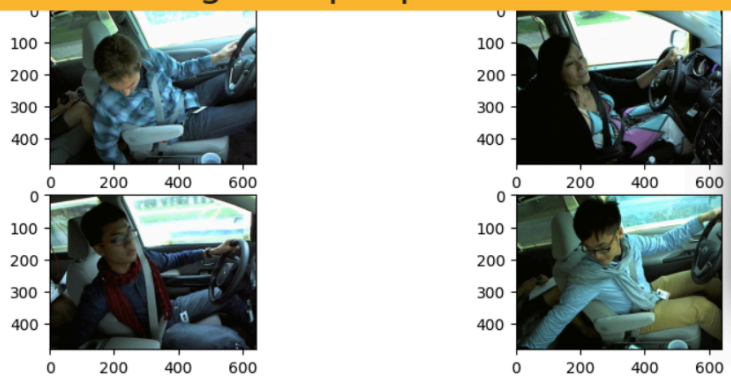


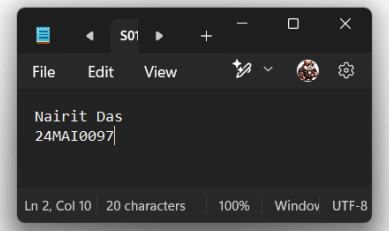



```
plt.figure(1, figsize=(15, 9))
plt.axis('off')
n = 0
for i in range(4):

    n += 1
    random_img = random.choice(image_list_turn)
    imgs = imread(random_img)
    # Use font properties directly instead of fontdict
    plt.suptitle("Random images of people who turn around",
                 fontsize=25,
                 fontfamily=font['family'], # Use font['family'] for font family
                 color=font['color'], # Use font['color'] for font color
                 backgroundcolor=background_color)
    plt.subplot(2,2,n)
    plt.imshow(imgs)

plt.show()
```

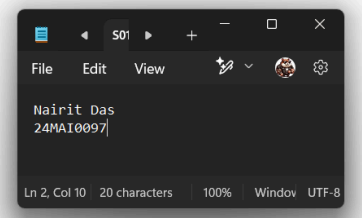


Random images of people who turn around

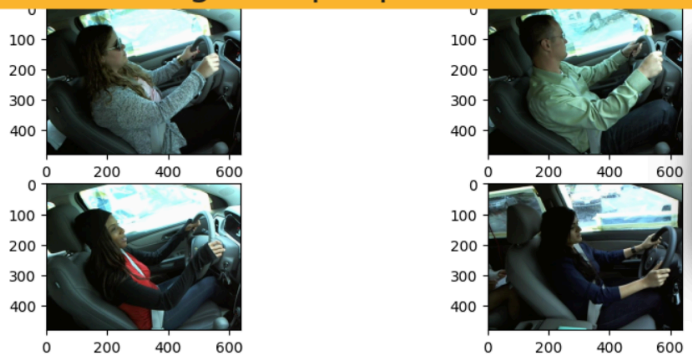


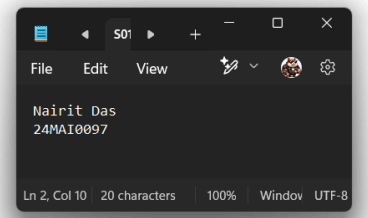



```
plt.figure(1, figsize=(10, 4))
plt.axis('off')
n = 0
for i in range(4):

    n += 1
    random_img = random.choice(image_list_safe)
    imgs = imread(random_img)
    plt.suptitle("Random images of people who drive safely",
                 fontsize=25,  # Removed 'fontdict' and used individual font properties
                 fontfamily=font['family'],  # Set font family
                 color=font['color'],  # Set font color
                 backgroundcolor=background_color)
    plt.subplot(2,2,n)
    plt.imshow(imgs)

plt.show()
```



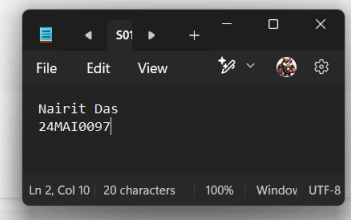
Random images of people who drive safely






```
print("Number of samples in (Class = Other) = " ,len(image_list_other))
print("Number of samples in (Class = Safe Driving) = " ,len(image_list_safe))
print("Number of samples in (Class = Talking Phone) = " ,len(image_list_talking))
print("Number of samples in (Class = Texting Phone) = " ,len(image_list_text))
print("Number of samples in (Class = Turning) = " ,len(image_list_turn))
```

```
Number of samples in (Class = Other) =  2119
Number of samples in (Class = Safe Driving) =  2203
Number of samples in (Class = Talking Phone) =  2169
Number of samples in (Class = Texting Phone) =  2203
Number of samples in (Class = Turning) =  2057
```

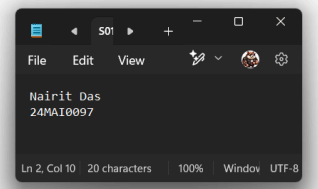



```
print(.75*len(image_list_other) , .2*len(image_list_other) ,.05*len(image_list_other))
print(.75*len(image_list_safe) , .2*len(image_list_safe) ,.05*len(image_list_safe))
print(.75*len(image_list_talking) , .2*len(image_list_talking) ,.05*len(image_list_talking))
print(.75*len(image_list_text) , .2*len(image_list_text) ,.05*len(image_list_text))
print(.75*len(image_list_turn) , .2*len(image_list_turn) ,.05*len(image_list_turn))
```

```
1589.25 423.8 105.95
1652.25 440.6 110.15
1626.75 433.8 108.45
1652.25 440.6 110.15
1542.75 411.40000000000003 102.85000000000001
```

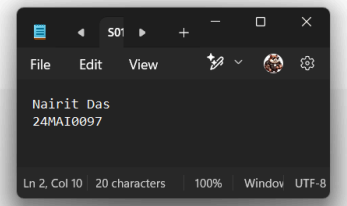
Nairit Das 24MAI0097

```
print("Train","Test", "Valid")

train_other = image_list_other[:1589]
test_other = image_list_other[1589:2012]
valid_other = image_list_other[2012:]

print (len(train_other), len(test_other), len(valid_other))

train_safe = image_list_safe[:1652]
test_safe = image_list_safe[1652:2092]
valid_safe = image_list_safe[2092:]

print (len(train_safe), len(test_safe), len(valid_safe))

train_talking = image_list_talking[:1626]
test_talking = image_list_talking[1626:2059]
valid_talking = image_list_talking[2059:]

print (len(train_talking), len(test_talking), len(valid_talking))

train_text = image_list_text[:1652]
test_text = image_list_text[1652:2092]
valid_text = image_list_text[2092:]

print (len(train_text), len(test_text), len(valid_text))

train_turn = image_list_turn[:1547]
test_turn = image_list_turn[1547:1959]
valid_turn = image_list_turn[1959:]
```




```
print (len(train_turn), len(test_turn), len(valid_turn))
```

```
Train Test Valid
1589 423 107
1652 440 111
1626 433 110
1652 440 111
1547 412 98
```



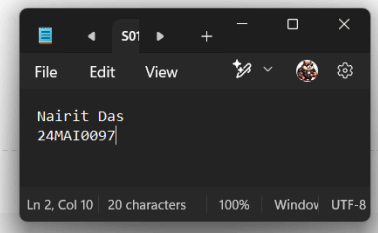

```
train_other_df = pd.DataFrame({'image':train_other, 'label':'Other'})
train_safe_df = pd.DataFrame({'image':train_safe, 'label':'Safe'})
train_talking_df = pd.DataFrame({'image':train_talking, 'label':'Talk'})
train_text_df = pd.DataFrame({'image':train_text, 'label':'Text'})
train_turn_df = pd.DataFrame({'image':train_turn, 'label':'Turn'})
test_other_df = pd.DataFrame({'image':test_other, 'label':'Other'})
test_safe_df = pd.DataFrame({'image':test_safe, 'label':'Safe'})
test_talking_df = pd.DataFrame({'image':test_talking, 'label':'Talk'})
test_text_df = pd.DataFrame({'image':test_text, 'label':'Text'})
test_turn_df = pd.DataFrame({'image':test_turn, 'label':'Turn'})
```

```
valid_other_df = pd.DataFrame({'image':valid_other, 'label':'Other'})
valid_safe_df = pd.DataFrame({'image':valid_safe, 'label':'Safe'})
valid_talking_df = pd.DataFrame({'image':valid_talking, 'label':'Talk'})
valid_text_df = pd.DataFrame({'image':valid_text, 'label':'Text'})
valid_turn_df = pd.DataFrame({'image':valid_turn, 'label':'Turn'})
train_df = pd.concat([train_other_df, train_safe_df, train_talking_df, train_text_df, train_turn_df])
test_df = pd.concat([test_other_df, test_safe_df, test_talking_df, test_text_df, test_turn_df])
val_df = pd.concat([valid_other_df, valid_safe_df, valid_talking_df, valid_text_df, valid_turn_df])
```

```
[8] train_df.head()
```

	image	label
0	/content/drive/MyDrive/DATA2/Revitsone-5classe...	Other
1	/content/drive/MyDrive/DATA2/Revitsone-5classe...	Other
2	/content/drive/MyDrive/DATA2/Revitsone-5classe...	Other
3	/content/drive/MyDrive/DATA2/Revitsone-5classe...	Other
4	/content/drive/MyDrive/DATA2/Revitsone-5classe...	Other

Next steps: Generate code with train_df | View recommended plots | New interactive sheet




```
[9] print("Number of rows in train dataframe is: ", len(train_df))
    print("Number of rows in test dataframe is: ", len(test_df))
    print("Number of rows in val dataframe is: ", len(val_df))
```

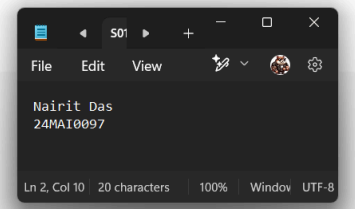
```
Number of rows in train dataframe is:  8066
Number of rows in test dataframe is:  2148
Number of rows in val dataframe is:  537
```

```
random_img_height = random.choice(train_other)
image= cv2.imread(random_img_height)

height, width= image.shape[:2]

print("The height is ", height)
```

Nairit Das 24MAI0097

```
[19] print("Number of rows in train dataframe is: ", len(train_df))
     print("Number of rows in test dataframe is: ", len(test_df))
     print("Number of rows in val dataframe is: ", len(val_df))
```

```
Number of rows in train dataframe is:  8066
Number of rows in test dataframe is:  2148
Number of rows in val dataframe is:  537
```



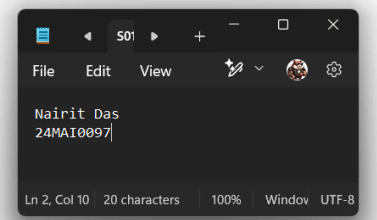

```
[20] random_img_height = random.choice(train_other)
     image= cv2.imread(random_img_height)

     height, width= image.shape[:2]

     print("The height is ", height)

     print("The width is ", width)
```

```
The height is  480
The width is  640
```

```
[21] Batch_size = 64
     Img_height = 240
     Img_width = 240
```

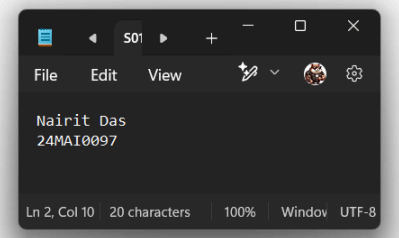
```
[22] trainGenerator = ImageDataGenerator(rescale=1./255.)
     valGenerator = ImageDataGenerator(rescale=1./255.)
     testGenerator = ImageDataGenerator(rescale=1./255.)
```

```
trainDataset = trainGenerator.flow_from_dataframe(
  dataframe=train_df,
  class_mode="categorical",
  x_col="image",
  y_col="label",
  batch_size=Batch_size,
  seed=42,
  shuffle=True,
  target_size=(Img_height,Img_width) #set the height and width of the images
)

testDataset = testGenerator.flow_from_dataframe(
  dataframe=test_df,
  class_mode='categorical',
  x_col="image",
  y_col="label",
  batch_size=Batch_size,
  seed=42,
  shuffle=True,
  target_size=(Img_height,Img_width)
)

valDataset = valGenerator.flow_from_dataframe(
  dataframe=val_df,
  class_mode='categorical',
  x_col="image",
  y_col="label",
  batch_size=Batch_size,
  seed=42,
  shuffle=True,
```



Nairit Das 24MAI0097

```python
def AlexNet():
    inp = layers.Input((240, 240, 3))
    x = layers.Conv2D(96, 11, 4, activation='relu')(inp)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D(3, 2)(x)
    x = layers.Conv2D(256, 5, 1, activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D(3, 2)(x)
    x = layers.Conv2D(384, 3, 1, activation='relu')(x)
    x = layers.Conv2D(384, 3, 1, activation='relu')(x)
    x = layers.Conv2D(256, 3, 1, activation='relu')(x)
    x = layers.MaxPooling2D(3, 2)(x)
    x = layers.Flatten()(x)
    x = layers.Dense(4096, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(4096, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(5, activation='softmax')(x)

    model_Alex = models.Model(inputs=inp, outputs=x)

    return model_Alex

model_Alex = AlexNet()
model_Alex.summary()
```



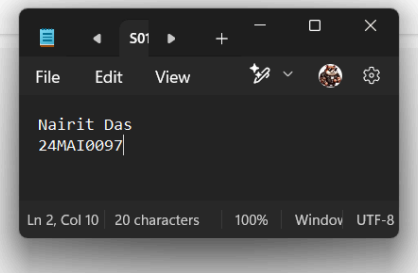
Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 240, 240, 3)	0
conv2d (Conv2D)	(None, 58, 58, 96)	34,944
batch_normalization (BatchNormalization)	(None, 58, 58, 96)	384
max_pooling2d (MaxPooling2D)	(None, 28, 28, 96)	0
conv2d_1 (Conv2D)	(None, 24, 24, 256)	614,656
batch_normalization_1 (BatchNormalization)	(None, 24, 24, 256)	1,024
max_pooling2d_1 (MaxPooling2D)	(None, 11, 11, 256)	0
conv2d_2 (Conv2D)	(None, 9, 9, 384)	885,120
conv2d_3 (Conv2D)	(None, 7, 7, 384)	1,327,488
conv2d_4 (Conv2D)	(None, 5, 5, 256)	884,992
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 256)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 4096)	4,198,400
dropout (Dropout)	(None, 4096)	0

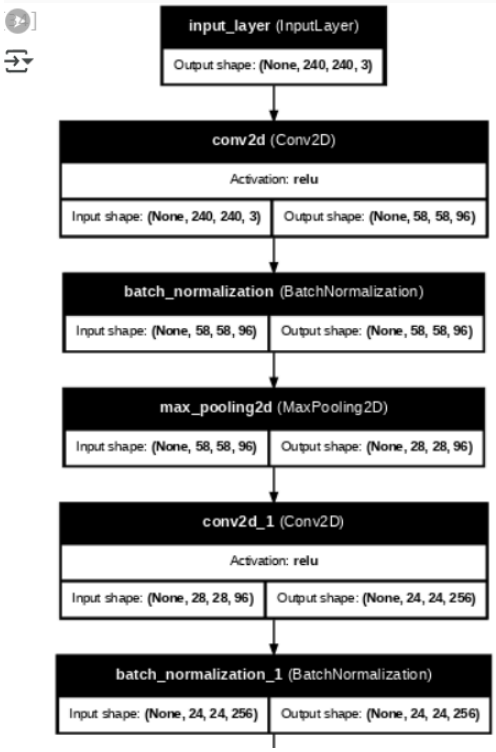
Nairit Das 24MAI0097

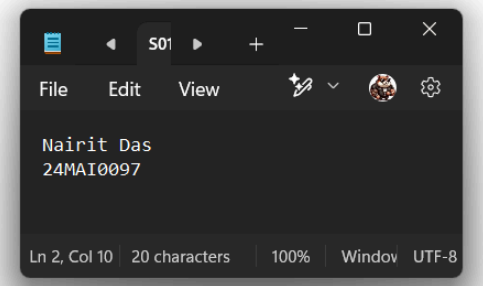
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16,781,312
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 5)	20,485

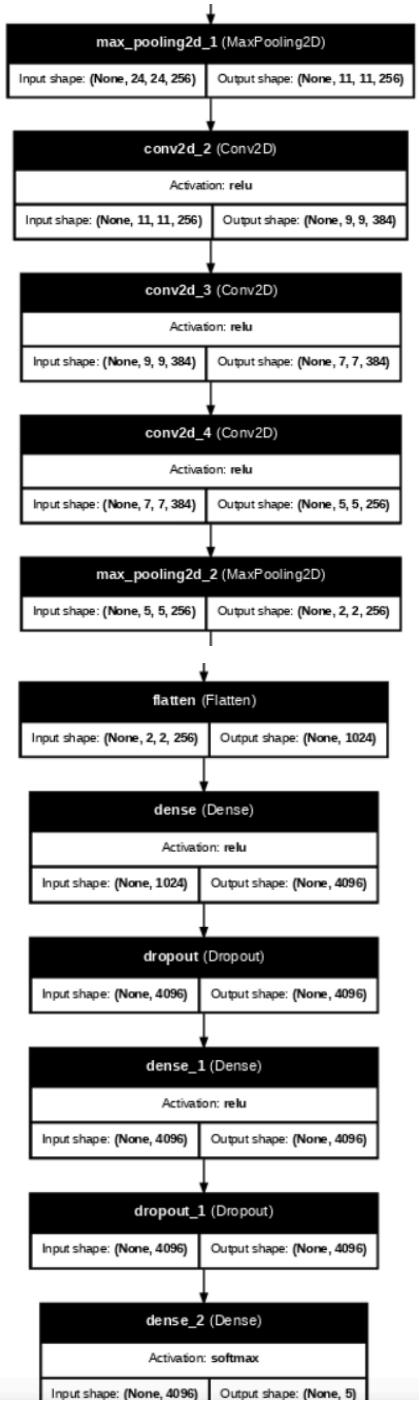
Total params: 24,748,805 (94.41 MB)
Trainable params: 24,748,101 (94.41 MB)
Non-trainable params: 704 (2.75 KB)

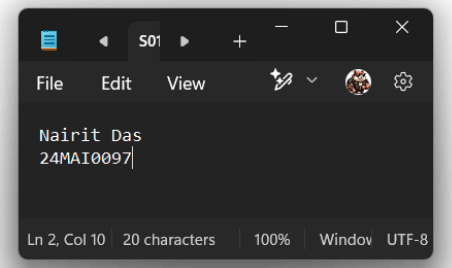


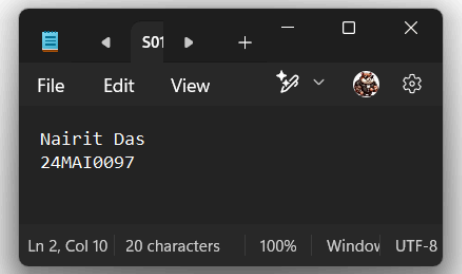

```
tf.keras.utils.plot_model(
    model_Alex,
    to_file='alex_model.png',
    show_shapes=True,
    show_dtype=False,
    show_layer_names=True,
    show_layer_activations=True,
    dpi=100
)
```






max_pooling2d_1 (MaxPooling2D)
Input shape: (None, 24, 24, 256)
Output shape: (None, 11, 11, 256)
conv2d_2 (Conv2D)
Activation: relu
Input shape: (None, 11, 11, 256)
Output shape: (None, 9, 9, 384)
conv2d_3 (Conv2D)
Activation: relu
Input shape: (None, 9, 9, 384)
Output shape: (None, 7, 7, 384)
conv2d_4 (Conv2D)
Activation: relu
Input shape: (None, 7, 7, 384)
Output shape: (None, 5, 5, 256)
max_pooling2d_2 (MaxPooling2D)
Input shape: (None, 5, 5, 256)
Output shape: (None, 2, 2, 256)
flatten (Flatten)
Input shape: (None, 2, 2, 256)
Output shape: (None, 1024)
dense (Dense)
Activation: relu
Input shape: (None, 1024)
Output shape: (None, 4096)
dropout (Dropout)
Input shape: (None, 4096)
Output shape: (None, 4096)
dense_1 (Dense)
Activation: relu
Input shape: (None, 4096)
Output shape: (None, 4096)
dropout_1 (Dropout)
Input shape: (None, 4096)
Output shape: (None, 4096)
dense_2 (Dense)
Activation: softmax
Input shape: (None, 4096)
Output shape: (None, 5)


File
Edit
View
Nairit Das
24MAI0097
Ln 2, Col 10
20 characters
100%
UTF-8


File
Edit
View
Nairit Das
24MAI0097
Ln 2, Col 10
20 characters
100%
UTF-8

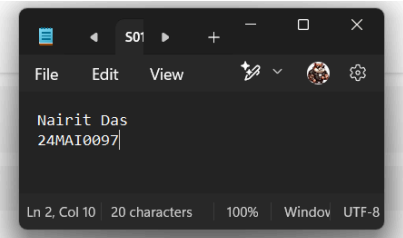
epochs

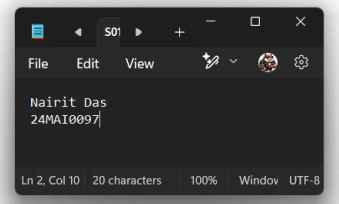
```
[26] model_Alex.compile(loss=BinaryCrossentropy(),
                        optimizer=Adam(learning_rate=0.001), metrics=['accuracy'])
```

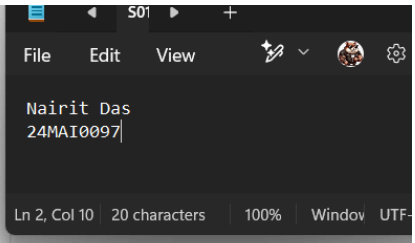
```
[27] Alex_model = model_Alex.fit(trainDataset, epochs=20, validation_data=valDataset)
```

```
Epoch 1/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 87s 565ms/step - accuracy: 0.3144 - loss: 0.8055 - val_accuracy: 0.3259 - val_loss: 0.5370
Epoch 2/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 66s 521ms/step - accuracy: 0.7163 - loss: 0.2564 - val_accuracy: 0.7393 - val_loss: 0.2657
Epoch 3/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 72s 570ms/step - accuracy: 0.8512 - loss: 0.1488 - val_accuracy: 0.8641 - val_loss: 0.1498
Epoch 4/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 73s 576ms/step - accuracy: 0.8960 - loss: 0.1090 - val_accuracy: 0.9330 - val_loss: 0.0747
Epoch 5/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 81s 568ms/step - accuracy: 0.9480 - loss: 0.0599 - val_accuracy: 0.9348 - val_loss: 0.0912
Epoch 6/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 73s 574ms/step - accuracy: 0.9457 - loss: 0.0560 - val_accuracy: 0.7486 - val_loss: 0.3396
Epoch 7/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 69s 540ms/step - accuracy: 0.9411 - loss: 0.0689 - val_accuracy: 0.9274 - val_loss: 0.0852
Epoch 8/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 72s 570ms/step - accuracy: 0.9595 - loss: 0.0461 - val_accuracy: 0.9534 - val_loss: 0.0617
Epoch 9/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 64s 500ms/step - accuracy: 0.9303 - loss: 0.0765 - val_accuracy: 0.9497 - val_loss: 0.0627
Epoch 10/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 79s 480ms/step - accuracy: 0.9584 - loss: 0.0453 - val_accuracy: 0.8231 - val_loss: 0.1803
Epoch 11/20
Epoch 12/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 64s 504ms/step - accuracy: 0.9731 - loss: 0.0317 - val_accuracy: 0.9609 - val_loss: 0.0513
Epoch 13/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 62s 491ms/step - accuracy: 0.9641 - loss: 0.0461 - val_accuracy: 0.9553 - val_loss: 0.0519
Epoch 14/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 63s 497ms/step - accuracy: 0.9788 - loss: 0.0278 - val_accuracy: 0.9665 - val_loss: 0.0530
Epoch 15/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 62s 483ms/step - accuracy: 0.9771 - loss: 0.0332 - val_accuracy: 0.9628 - val_loss: 0.0637
Epoch 16/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 59s 467ms/step - accuracy: 0.9773 - loss: 0.0310 - val_accuracy: 0.9758 - val_loss: 0.0578
Epoch 17/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 60s 475ms/step - accuracy: 0.9836 - loss: 0.0179 - val_accuracy: 0.9441 - val_loss: 0.0647
Epoch 18/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 59s 460ms/step - accuracy: 0.9737 - loss: 0.0346 - val_accuracy: 0.9292 - val_loss: 0.1029
Epoch 19/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 58s 458ms/step - accuracy: 0.9770 - loss: 0.0293 - val_accuracy: 0.9795 - val_loss: 0.0533
Epoch 20/20
127/127 ━━━━━━━━━━━━━━━━━━━━ 58s 460ms/step - accuracy: 0.9804 - loss: 0.0261 - val_accuracy: 0.9423 - val_loss: 0.0899
```

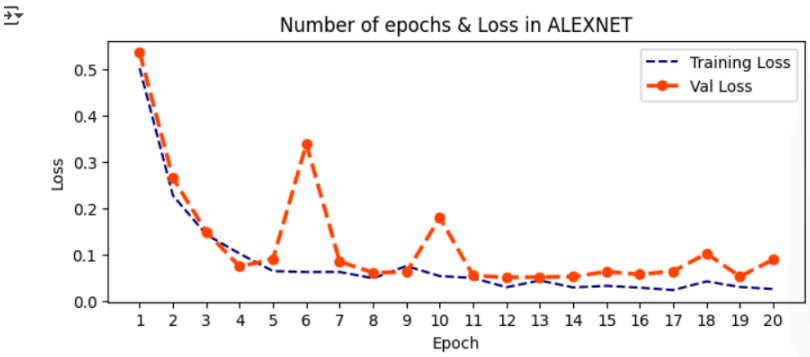
```
28] training_loss_alex = Alex_model.history['loss']
    val_loss_alex = Alex_model.history['val_loss']
    training_acc_alex = Alex_model.history['accuracy']
    val_acc_alex = Alex_model.history['val_accuracy']
    epoch_count = range(1, len(training_loss_alex) + 1)

    # Visualize loss history
    plt.figure(figsize=(8,3), dpi=200)
    plt.plot(epoch_count, training_loss_alex, 'r--', color= 'navy')
    plt.plot(epoch_count, val_loss_alex, '--bo',color= 'orangered', linewidth = '2.5', label='line with marker')
    plt.legend(['Training Loss', 'Val Loss'])
    plt.title('Number of epochs & Loss in ALEXNET')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.xticks(np.arange(1,21,1))
    plt.show();
```

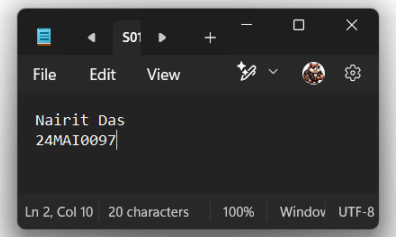






Nairit Das 24MAI0097






```
plt.figure(figsize=(10,5), dpi=100)
plt.plot(epoch_count, training_acc_alex, 'r--', color= 'navy')
plt.plot(epoch_count, val_acc_alex, '--bo',color= 'orangered', linewidth = '2.5', label='line with marker')
plt.legend(['Training Acc', 'Val Acc'])
plt.title('Number of epochs & Accuracy in ALEXNET')
plt.xlabel('Epoch')
plt.ylabel('Acc')
plt.xticks(np.arange(1,21,1))
plt.plot();
plt.show();
```