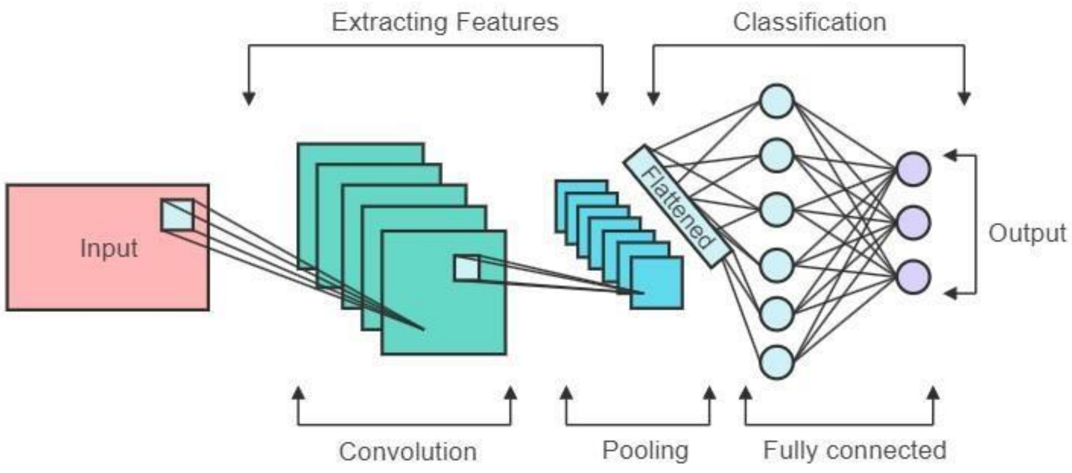
CNN:

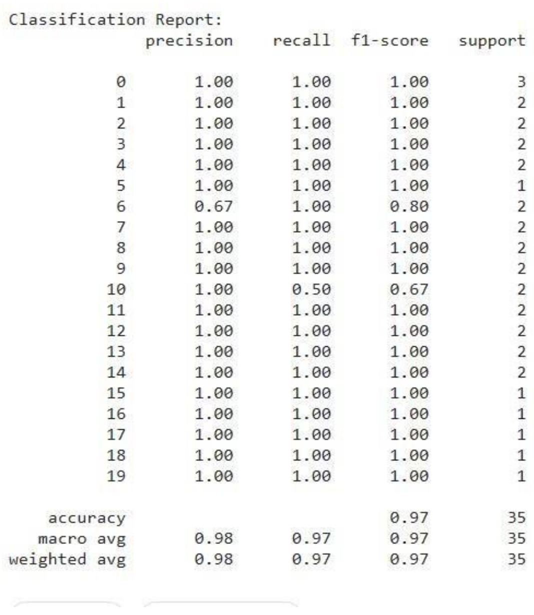


```
Model: "sequential"

_________________________________________________________________

Layer (type)              Output Shape            Param #

=================================================================

conv2d (Conv2D)           (None, 224, 224, 16)     448

_________________________________________________________________

__ max_pooling2d (MaxPooling2D) (None, 112, 112, 16)     0

_________________________________________________________________

__ conv2d_1 (Conv2D)          (None, 112, 112, 16)     2320

_________________________________________________________________

__ max_pooling2d_1 (MaxPooling2 (None, 56, 56, 16)       0

_________________________________________________________________

__ conv2d_2 (Conv2D)          (None, 54, 54, 32)       4640

_________________________________________________________________
```

```
__ max_pooling2d_2 (MaxPooling2 (None, 27, 27, 32)       0
___________________________________________________________________ flatten (Flatten)
(None, 23328)           0

________________________________________________________

____________ dense (Dense)            (None, 256)            5972224

________________________________________________________

____________ dense_1 (Dense)           (None, 128)            32896

________________________________________________________________

___ dense_2 (Dense)           (None, 20)             2580

=================================================================

Total params: 6,011,108

Trainable params: 6,011,108

Non-trainable params: 0

_________________________________________________________________
```

Explanation of the layers:

- Conv2D (1st layer):
  - Output Shape: (None, 224, 224, 16)
  - Number of Parameters: 448 (3x3 filter size * 16 filters + 16 biases) MaxPooling2D (1st layer):
- Output Shape: (None, 112, 112, 16)
  - No trainable parameters (Pooling layer) Conv2D (2nd
  - layer):
- Output Shape: (None, 112, 112, 16)
  - Number of Parameters: 2320 (3x3 filter size * 16 filters + 16 biases) MaxPooling2D (2nd layer):
  - Output Shape: (None, 56, 56, 16)
- No trainable parameters (Pooling layer) Conv2D (3rd layer):
  - Output Shape: (None, 54, 54, 32)
- Number of Parameters: 4640 (3x3 filter size * 32 filters + 32 biases) MaxPooling2D (3rd layer):
  - Output Shape: (None, 27, 27, 32)
  - No trainable parameters (Pooling layer) Flatten:
- Output Shape: (None, 23328)
  - No trainable parameters (Flatten layer) Dense (1st fully
  - connected layer):
- Output Shape: (None, 256)
  - Number of Parameters: 5,972,224 (23328 input neurons * 256 output neurons + 256 biases)
  - Dense (2nd fully connected layer):
- Output Shape: (None, 128)
  - Number of Parameters: 32,896 (256 input neurons * 128 output neurons + 128 biases) Dense (Output layer):
- Output Shape: (None, 20) (Assuming it's a classification task with 20 classes)
  - Number of Parameters: 2,580 (128 input neurons * 20 output neurons + 20 biases) Total
  - Trainable Parameters:
- 6,011,108
  - Here's the training summary:
-
- Initial Training Accuracy: 5.71%
- Final Training Accuracy: 91.43%
- Initial Validation Accuracy: 5.71%
- Final Validation Accuracy: 97.14%



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

           0       1.00      1.00      1.00         3
           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00         2
           3       1.00      1.00      1.00         2
           4       1.00      1.00      1.00         2
           5       1.00      1.00      1.00         1
           6       0.67      1.00      0.80         2
           7       1.00      1.00      1.00         2
           8       1.00      1.00      1.00         2
           9       1.00      1.00      1.00         2
          10       1.00      0.50      0.67         2
          11       1.00      1.00      1.00         2
          12       1.00      1.00      1.00         2
          13       1.00      1.00      1.00         2
          14       1.00      1.00      1.00         2
          15       1.00      1.00      1.00         1
          16       1.00      1.00      1.00         1
          17       1.00      1.00      1.00         1
          18       1.00      1.00      1.00         1
          19       1.00      1.00      1.00         1

    accuracy                           0.97        35
   macro avg       0.98      0.97      0.97        35
weighted avg       0.98      0.97      0.97        35
```

Classification Deep Learning Model

Dataset Preparation:

The dataset was structured with 20 category folders, each containing "Train" and "Validation" subfolders.
Image data augmentation techniques were applied to enhance the diversity of the training
dataset. so we read the training dataset from folder:" augmented_train"

Data Loading

The training and validation images were loaded using a Python script, traversing through the category folders.
Images were loaded, preprocessed, and organized into arrays for both training and validation.

Model Architecture:

The EfficientNetB0 architecture was chosen as the base model for feature extraction.
A custom dense layer was added on top for classification with softmax activation.
The model was compiled using the Adam optimizer, categorical crossentropy loss, and accuracy as the evaluation metric.

Training:

The model was trained on the training dataset for a specified number of epochs.
Data augmentation during training aimed to improve the model's generalization.

Evaluation:

The trained model was evaluated on the validation dataset to assess its accuracy and generalization.
Key metrics, including training and validation accuracy, were monitored during training.

Prediction and Classification:

A function was developed to classify new images using the trained model.
Images were loaded, preprocessed, and fed into the model for prediction.
The predicted category was displayed based on the model's output.
Results:

The trained deep learning model achieved a test accuracy of 94% after 5 epochs.
The classification function successfully predicted categories for new images, providing insights into the model's practical utility.
The model's performance was further assessed using the validation dataset, revealing a test accuracy of 97%.

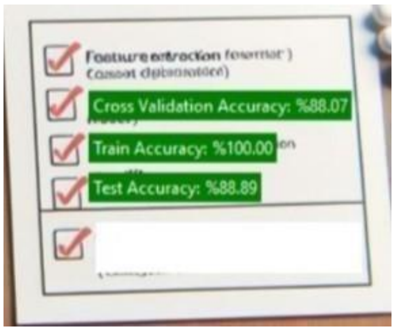
Classical Classification Task

Feature Extraction with HoG:

I began by implementing feature extraction using Histogram of Oriented Gradients (HoG). The process involved reading the image path, converting it to grayscale, and resizing it to (128*128) for uniformity. Subsequently, the HoG descriptor was employed to compute image features.

SVM Classifier:

Initial classification using Support Vector Machine (SVM) yielded promising results with a test accuracy of 83%. However, a concern arose due to a higher training accuracy of 100%, indicating potential overfitting. To mitigate this, I introduced data augmentation techniques, including cropping, brightness and contrast adjustments, saturation, rotation, blur, scale, and redundancy. This diversification increased the test accuracy to 88%, though the training accuracy remained elevated.

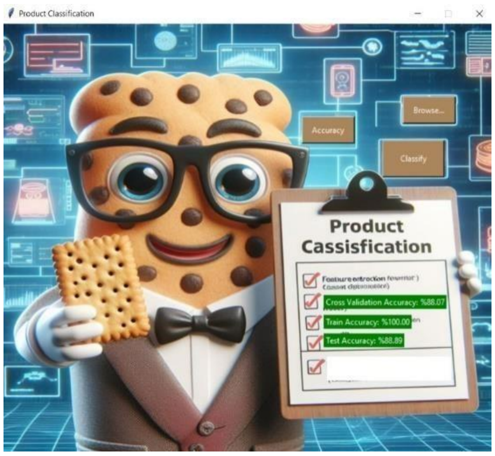


Cross Validation Accuracy: %88.07
Train Accuracy: %100.00
Test Accuracy: %88.89

To address overfitting further, I changed the sizing of the image at features extraction function and set it to 224*224 but It didn't worth then I experimented with SVM hyperparameters. Adjusting the C parameter and exploring different kernel functions, I settled on a poly kernel with degree=1 and a linear kernel with C=0.7. This optimization resulted in a test accuracy of 86.11%. Despite this improvement, the training accuracy persisted as a concern.

By searching for solutions, I found that cross validation accuracy will help me to evaluate the performance of my classifier, so I calculated and displayed it, and it varies at each change happen to hyperparammeters or the data augmentation

I explored alternative classifiers, including Random Forest and Logistic Regression. However, these alternatives yielded lower accuracies and resource-intensive training, respectively. Ultimately, SVM proved to be the most effective, with a final accuracy plateauing at 86%.



Cross Validation Accuracy: %88.07
Train Accuracy: %100.00
Test Accuracy: %88.89

GUI Implementation:

To enhance user interaction, I integrated a simple GUI using the tkinter library. Three buttons—Accuracy, Browse, and Classify—were added for distinct functionalities. The "Accuracy" button displays the model accuracies, "Browse" facilitates the uploading of validation images, and "Classify" performs image classification, displaying the predicted category. The GUI components were meticulously formatted to ensure a visually appealing and user-friendly interface, aligning with the background theme.

Part 2

2- siames model

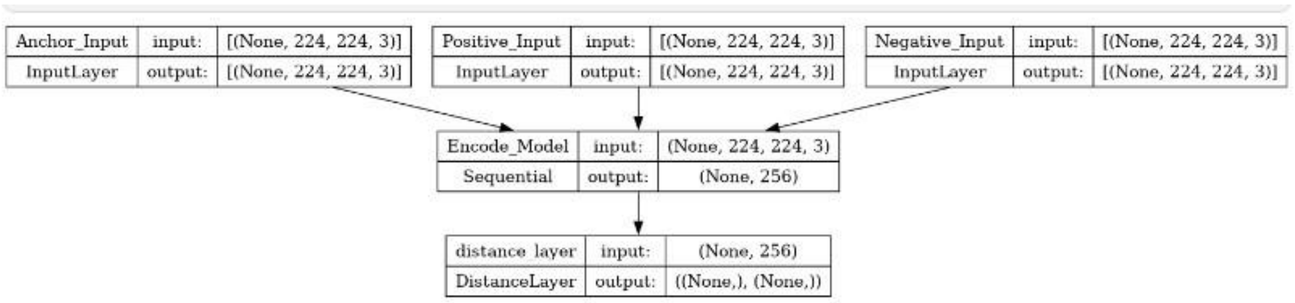
1-data prep >> resize (224,224,3) , read the imgs RGB , create train and test dict. Contain folder name and number of imgs in each class , create triples by get the anchor , positive and negative images and store it ,

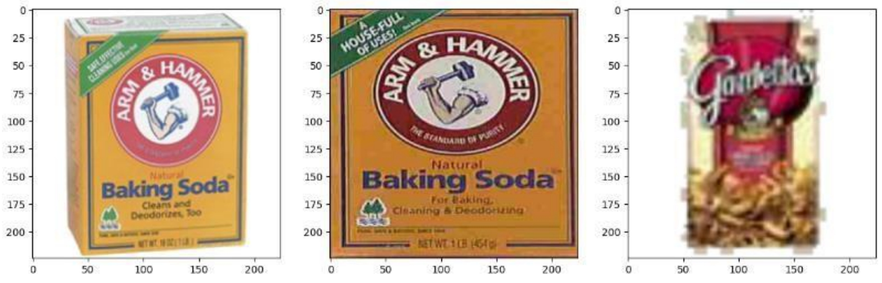
2-model tech >> create triples , apply encoding on images using xception model and freeze some layers in it ,calculate the embedding and the distance ,

3- train time >> 710 sec   test time >> 6.7 sec

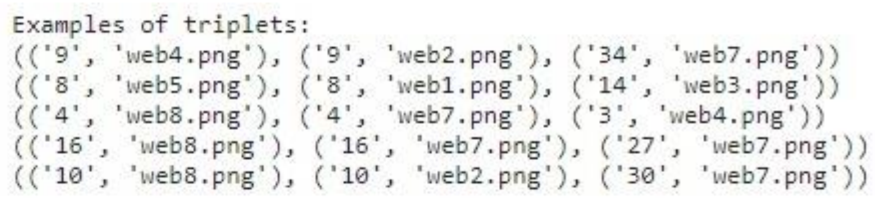
4- train accuracy >> 96%
test accuracy >>93%

the model archi

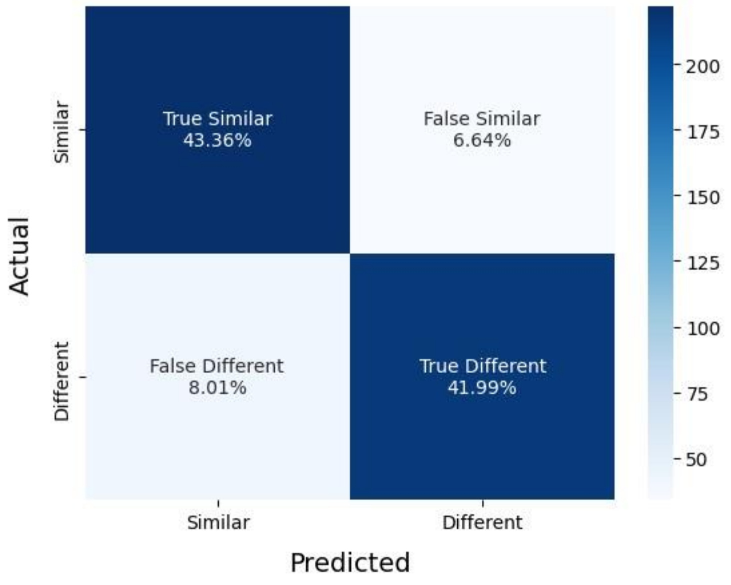




Example of the tripels



```
Examples of triplets:
(('9', 'web4.png'), ('9', 'web2.png'), ('34', 'web7.png'))
(('8', 'web5.png'), ('8', 'web1.png'), ('14', 'web3.png'))
(('4', 'web8.png'), ('4', 'web7.png'), ('3', 'web4.png'))
(('16', 'web8.png'), ('16', 'web7.png'), ('27', 'web7.png'))
(('10', 'web8.png'), ('10', 'web2.png'), ('30', 'web7.png'))
```


True Similar
False Similar
False Different
True Different
Actual
Similar
Different
Predicted
Similar
Different