Assignment 01

E9 246 Advanced Image Processing

Name: **SRIYA R G**
SR No: 22414
Date of Sub: 04.02.2024

1. PCA-SIFT

- Compute the PCA-SIFT feature descriptors for the images given here (You can ignore the second step).
- Modify the images by (a) Scaling, (b) Rotation (c) Gaussian blur and obtain the keypoints for these images.
- Analyse the keypoints detected qualitatively and quantitatively (number of keypoints detected in each case).

0.1 Q1- Answer

- Details of implementation:

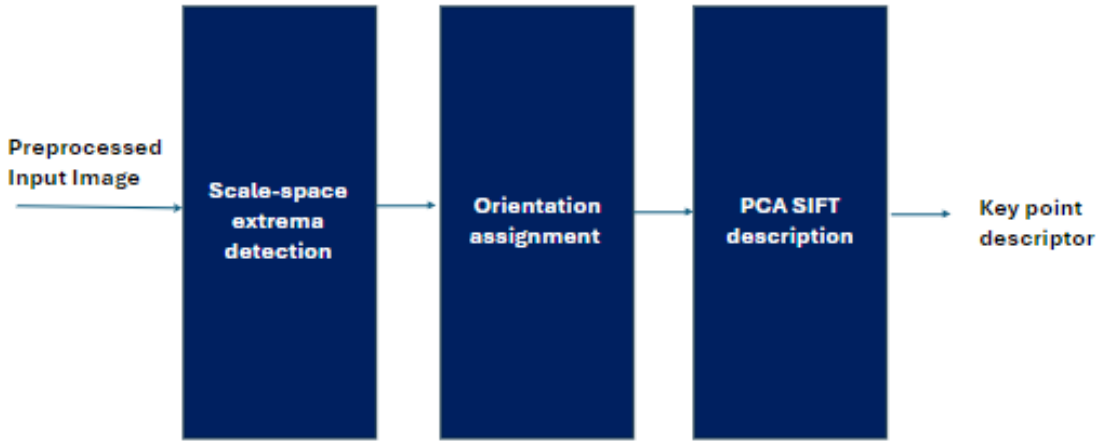

Figure 1: Caption

- Image preprocessing: Input image is read and converted to grayscale, downsampled if required and resized accordingly.
- Calculation of number of octaves: Number of octave images was calculated as $int(round(log(min(image.s$ 1)) . For the given image, 7 octaves were calculated.
- Number of scales per octave: As a rule of thumb, number is selected between 4 and 6. Both gave similar results.
- Generation of Gaussian Pyramid: k factor was calculated as $2^{(1/numberofscales)}$. For the input image, Gaussian blur (for each octave and scale, 28 in total) was applied with sigma, where sigma is k^{scale}
- Generation of difference of Gaussian images: Adjacent Gaussian images were subtracted to obtain Difference of Gaussian images. After each octave, Gaussian was downsampled by 2.

- Extrema detection: Each point was compared with it's 8 neighbors of same scale and 9 neighbors of adjacent scales each. If was detected as a key point if is is either maximum or minimum among these 26 neighbors.
- Orientation assignment was done as discussed in Lowe's paper. For each interest point, surrounding patch of radius 1.5 times the respective scale was chosen and gradient was computed and dominant orientation was assigned to the patch based on 10 histogram bins. Each keypoint was stored as a tuple with location, scale, octave and orientation information.
- PCA SIFT descriptor representation of keypoint: For each keypoint, 41x41 patch around it was selected and gradients in x and y direction were computed and stored as a long vector of size 3042x1. This dimension was reduced using PCA to 20x1 vector and normalized for each key point.

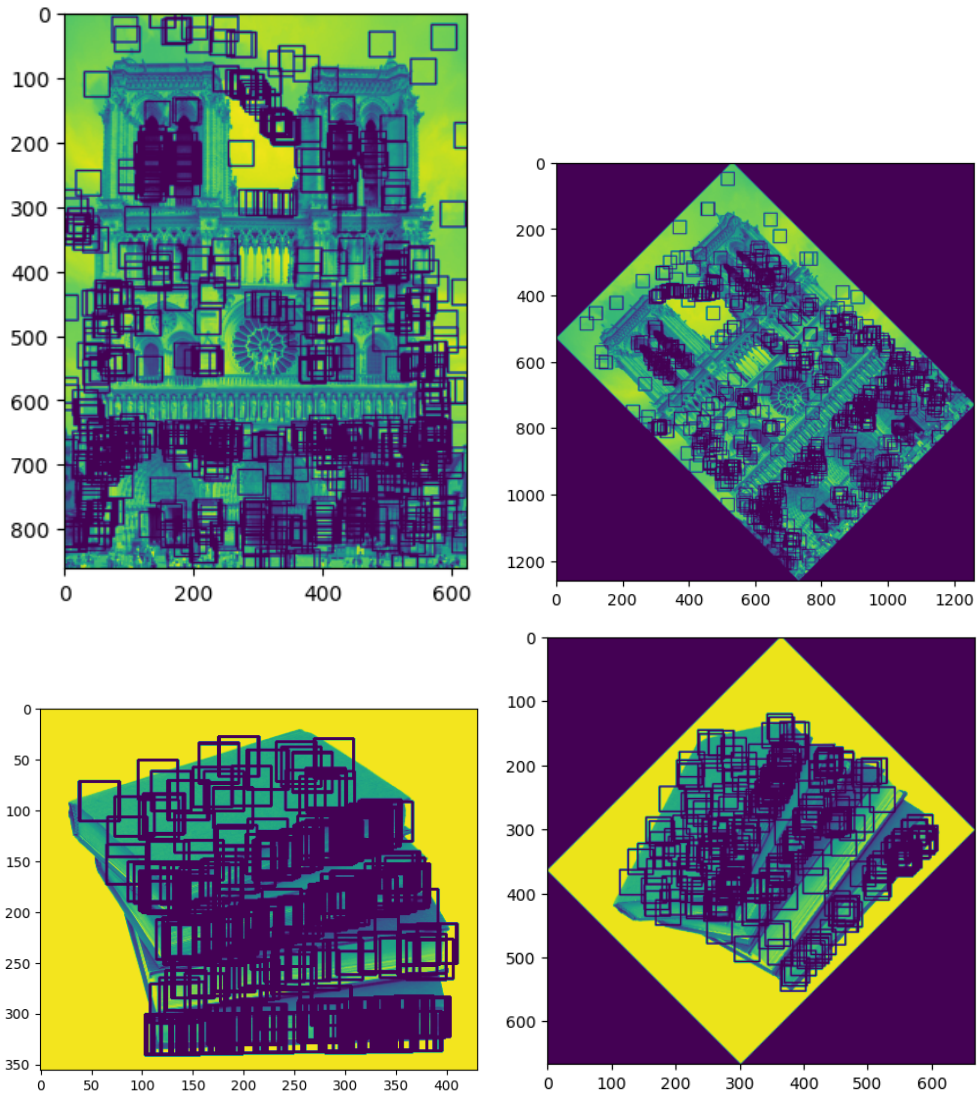

Figure 2: Example of 41x41 patch around keypoints

- Matching: Distance between descriptors (original and modified as per the question) were calculated and good matches were calculated based on Lowe's ratio test with ratio factor as 0.75.

- Results

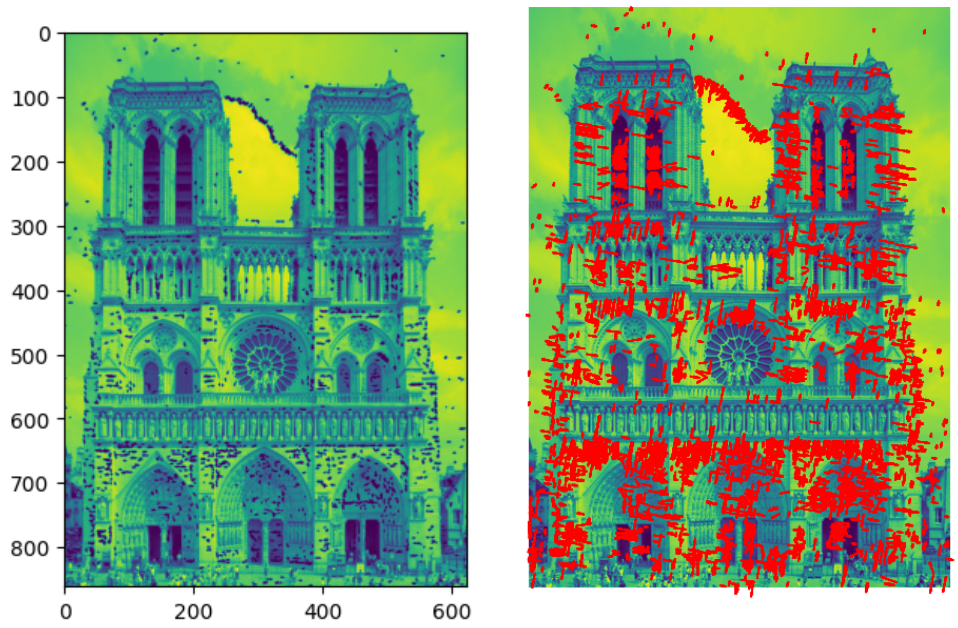

Figure 3: Detected Keypoints for image 1

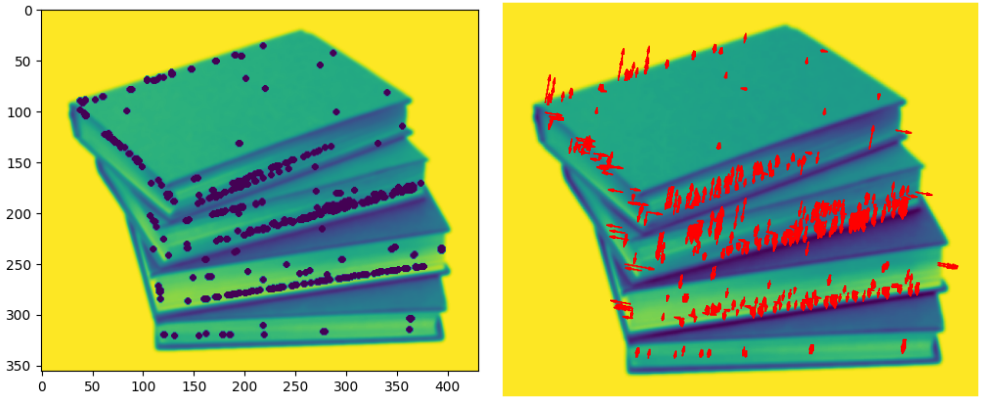

Figure 4: Detected Keypoints for image 2

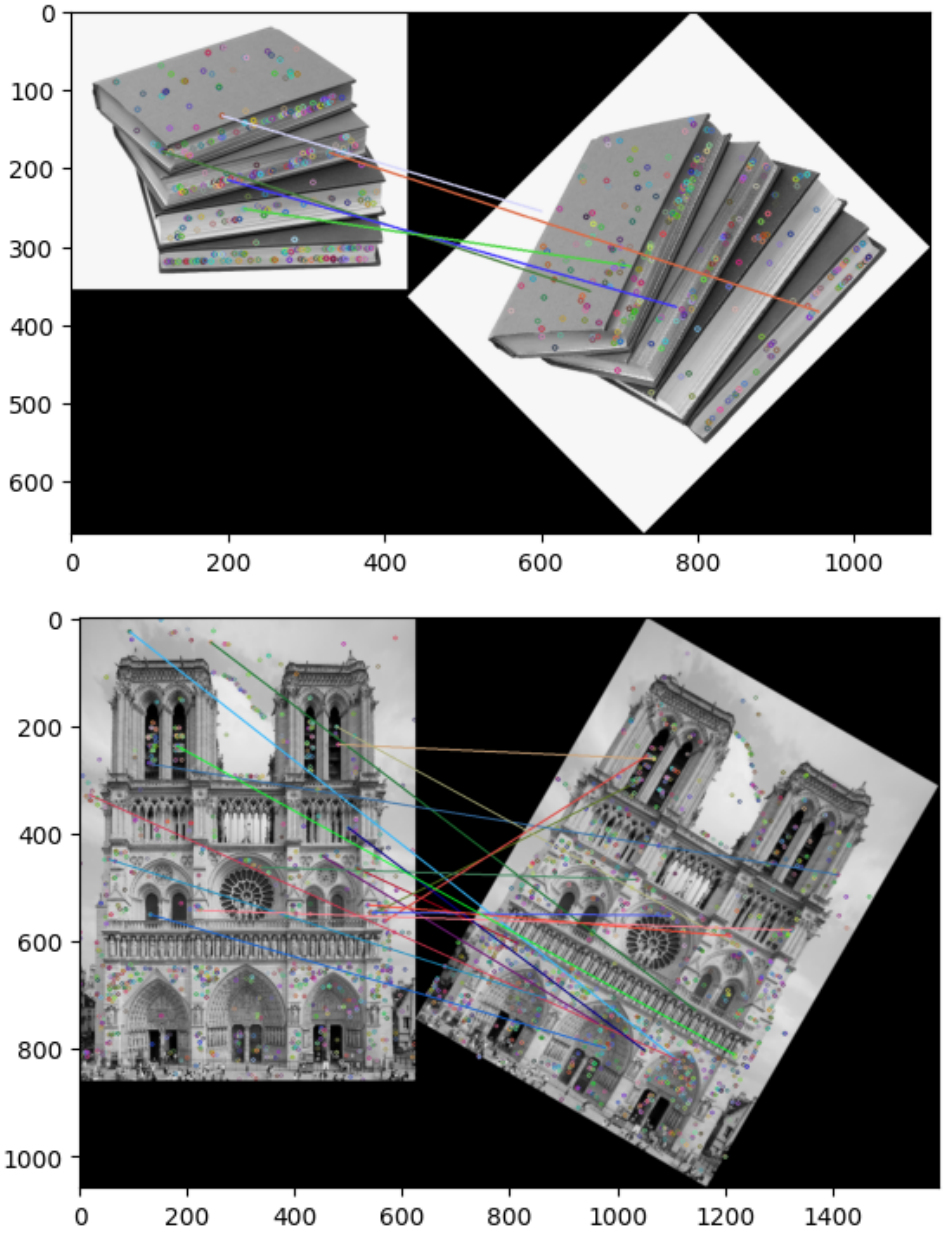

Figure 5: Matching results with rotation and scale

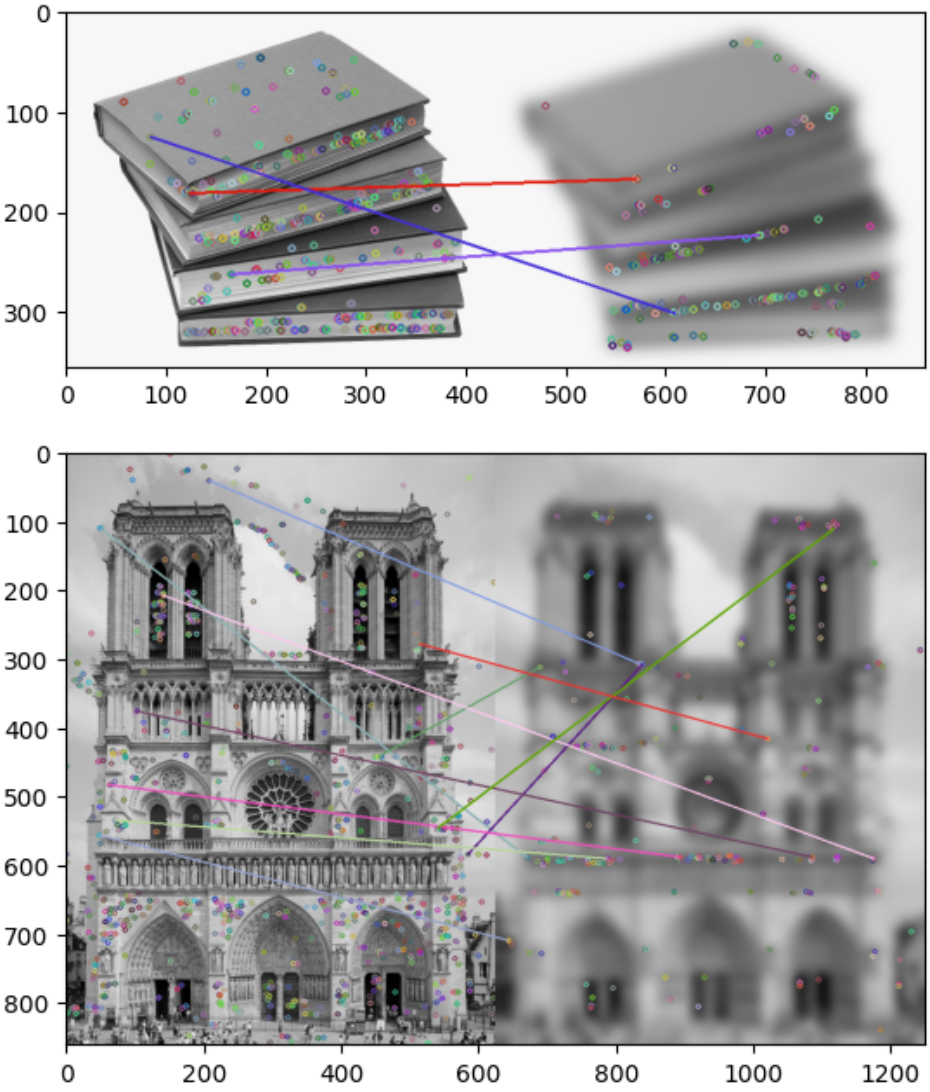

Figure 6: Matching results with blur

- Observations and learnings:

  - Quantitatively, around 32000 keypoints were detected for img1 (original) and 3200 keypoints were detected for img2. When performed modifications(blur, rotation, downscalaing), number of detected keypoints reduced to around 25000 and 2000 respectively.
  - Hardcoding number of octaves gave different results. Lower the number, lesser the key points detected. Downscaling gave lesser number of keypoints, Blurring factor between 1 and 5 gave more keypoints. This might be attributed to the fact that keypoint localization is not performed

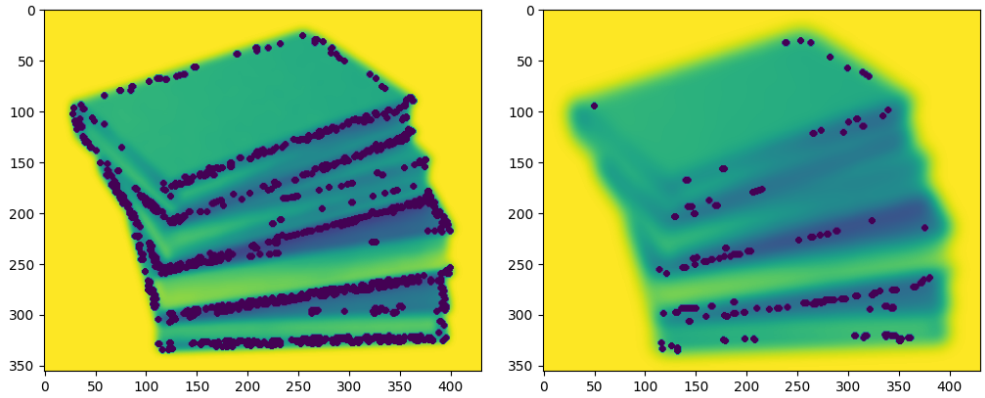

Figure 7: Blur factor of 4 (left) and blur factor of 8(Right)

- Preprocessing image with Gaussian blur or interpolating affected number of keypoints detected. Upsampling gave more keypoints.

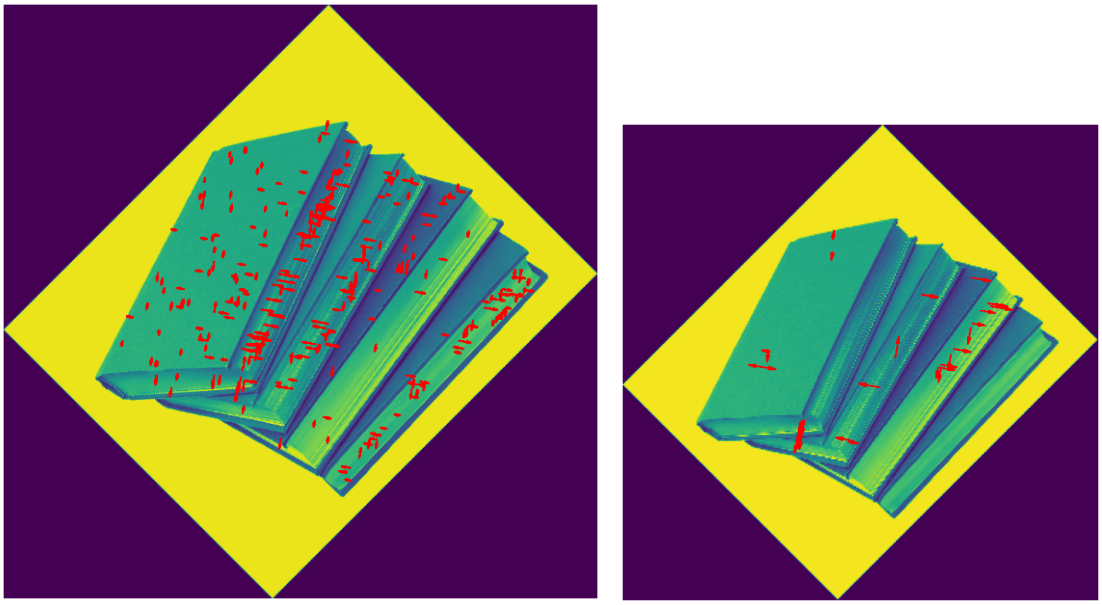

Figure 8: Upscaling (left) and Downscaling (Right)

2. Image Classification

- Build a custom CNN with conv, sigmoid, pooling and fc layers. Train the network for the CIFAR-10 dataset given [here] using the training data and report the performance on test data. Change the non linearity to ReLU, train and test the model. Compare the results obtained using these two models.
- Evaluate the accuracy on the additional test set given. here.

Q2- Answer

- Details of implementation:

- Data preprocessing: The given data (Training and testing) was loaded and preprocessed. Preprocessing included: a) normalizing image pixel values, i.e, dividing the intensities by 255. b) Image resizing, c) Converting categorical labels into one-hot representation.
- Architecture: As given, a custom CNN was defined using sequential model of Python keras, which included convolutional layers, activation function, pooling, flattening(To convert the output into 1D array) and fully connected layers. In the last fc layer, softmax activation was used to convert raw scores (logits) into probabilities. In previous convolutional layers, Sigmoid vs ReLu activation functions were used and the performance was compared. Number of filters, size and layers was decided empirically.

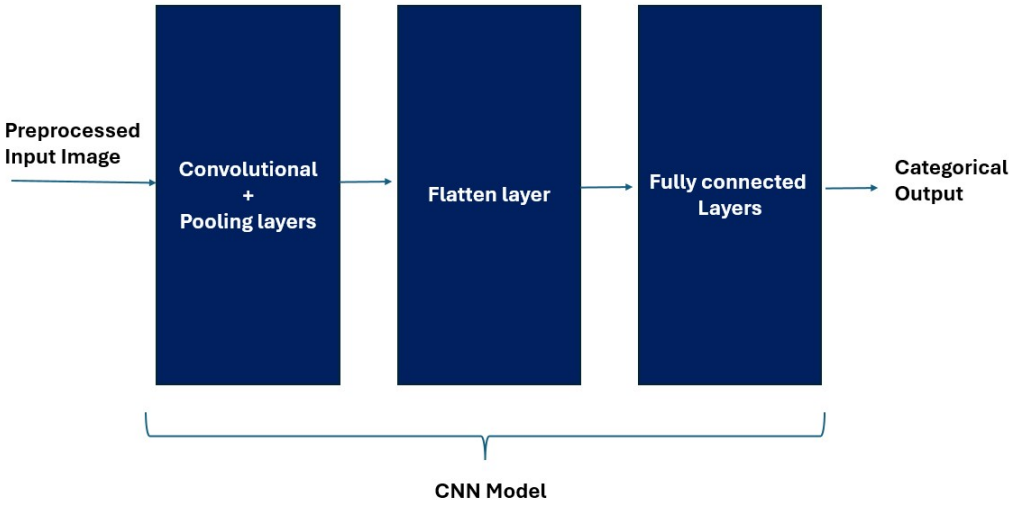

Figure 9: Pictorial representation of CNN model implementation

- Choice of optimizer: Out of all optimizers, ADAM gave the best results
- Choice of Pooling: Out of Maximum Pooling and Average pooling, Average pooling gave slightly better results.
- Batch size: Different batch sizes of 16,32,64 were tried and batch size of 32 gave the best accuracy
- Stopping criteria: Validation loss (Cross categorical entropy) for each epoch was monitored. If the difference did not change by 0.00001 for 5 epochs, training was stopped. This too was chosen empirically.
- Evaluation metric: Accuracy on training and test data
- Final model

```
# Convolutional layer with 256 filters, each of size 3x3
model.add(layers.Conv2D(256, (3, 3), activation=activation, input_shape=(32, 32, 3)))
# AveragePooling layer with pool size 2x2
model.add(layers.AveragePooling2D((2, 2)))
# Convolutional layer with 64 filters, each of size 3x3,
model.add(layers.Conv2D(128, (3, 3), activation=activation))
# AveragePooling layer with pool size 2x2
model.add(layers.AveragePooling2D((2, 2)))
# Convolutional layer with 128 filters, each of size 3x3
model.add(layers.Conv2D(128, (3, 3), activation=activation))
# AveragePooling layer with pool size 2x2
model.add(layers.AveragePooling2D((2, 2)))
# Flatten layer to flatten the output for the fully connected layer
model.add(layers.Flatten())
# FC layers
model.add(layers.Dense(128, activation=activation))
model.add(layers.Dense(64, activation=activation))
# Output layer with 10 neurons (for 10 classes) and 'softmax' activation function
model.add(layers.Dense(10, activation='softmax'))
```

Figure 10: Final Model

- Results
  - Sigmoid activation function

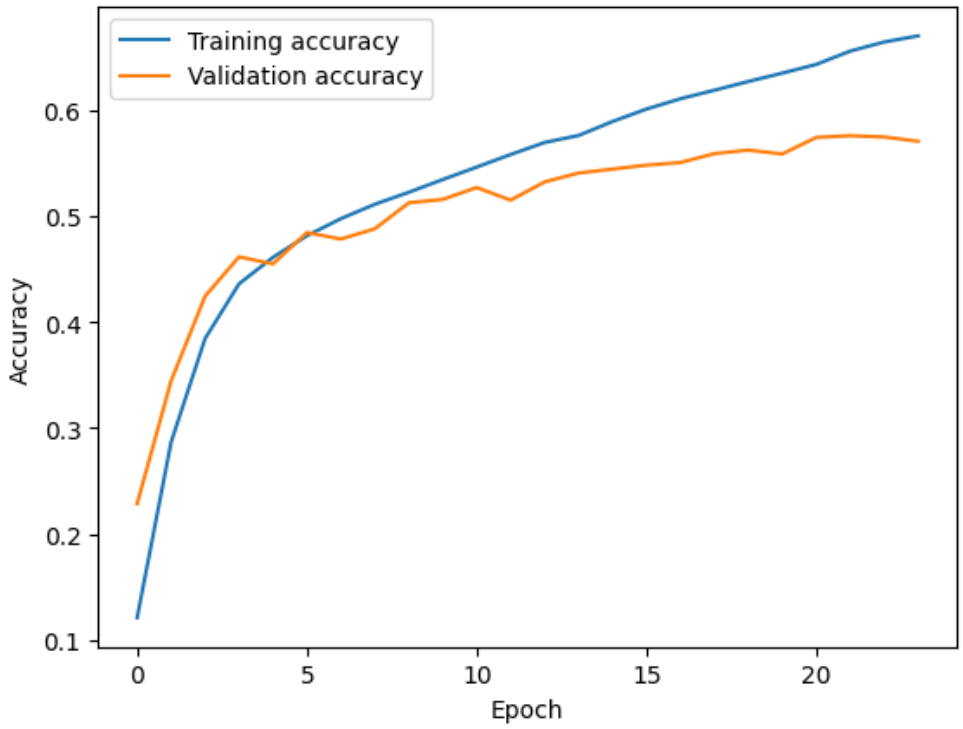

Figure 11: Accuracy plot for training and test data- Sigmoid activation function

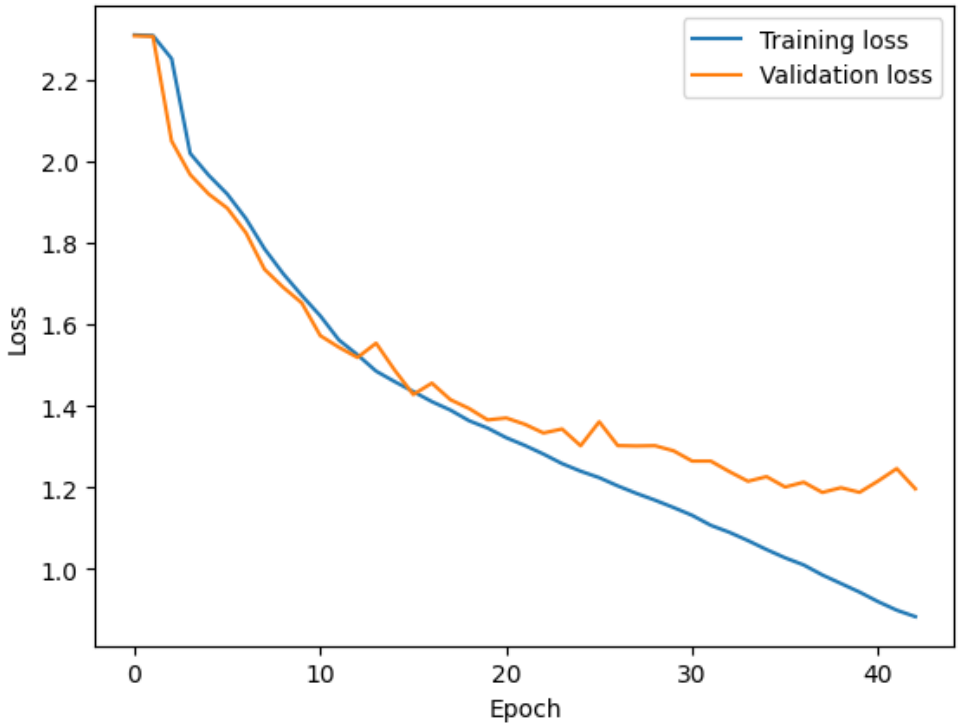

Figure 12: Loss plot for Sigmoid activation function

```
313/313 [==============================] - 34s 109ms/step - loss: 1.1874 - accuracy: 0.5879
Test accuracy: 58.79%
313/313 [==============================] - 35s 111ms/step
313/313 [==============================] - 35s 111ms/step - loss: 1.2025 - accuracy: 0.5822
Accuracy on additional test data: [1.2024807929992676, 0.5821999907493591]
```

Figure 13: Screenshot of results for Sigmoid activation function

- Relu activation function

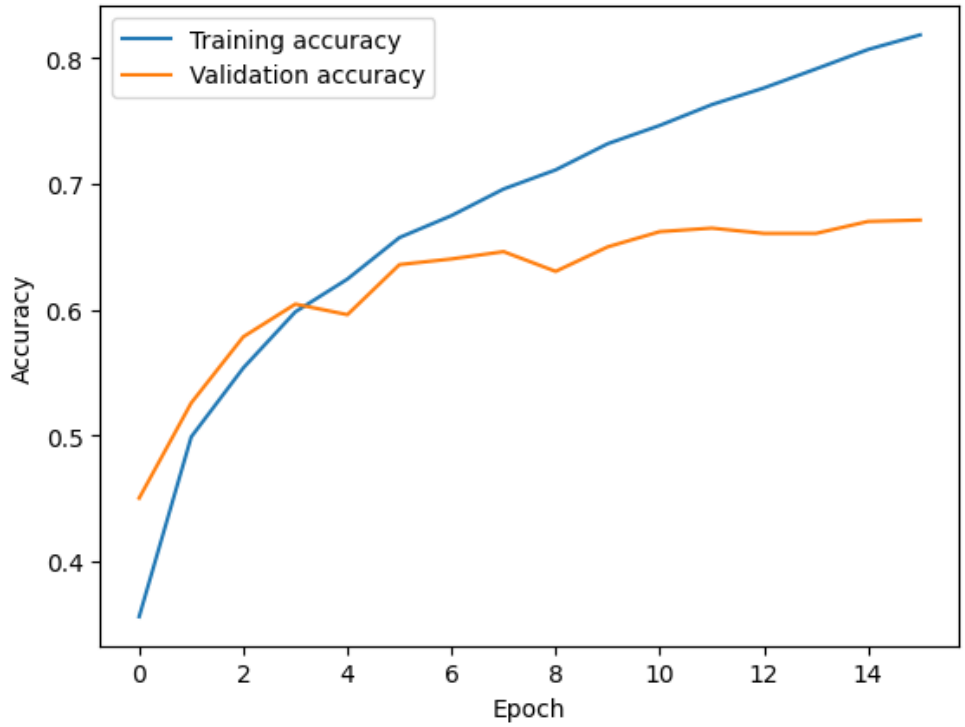

Figure 14: Accuracy plot for training and test data- ReLu activation function

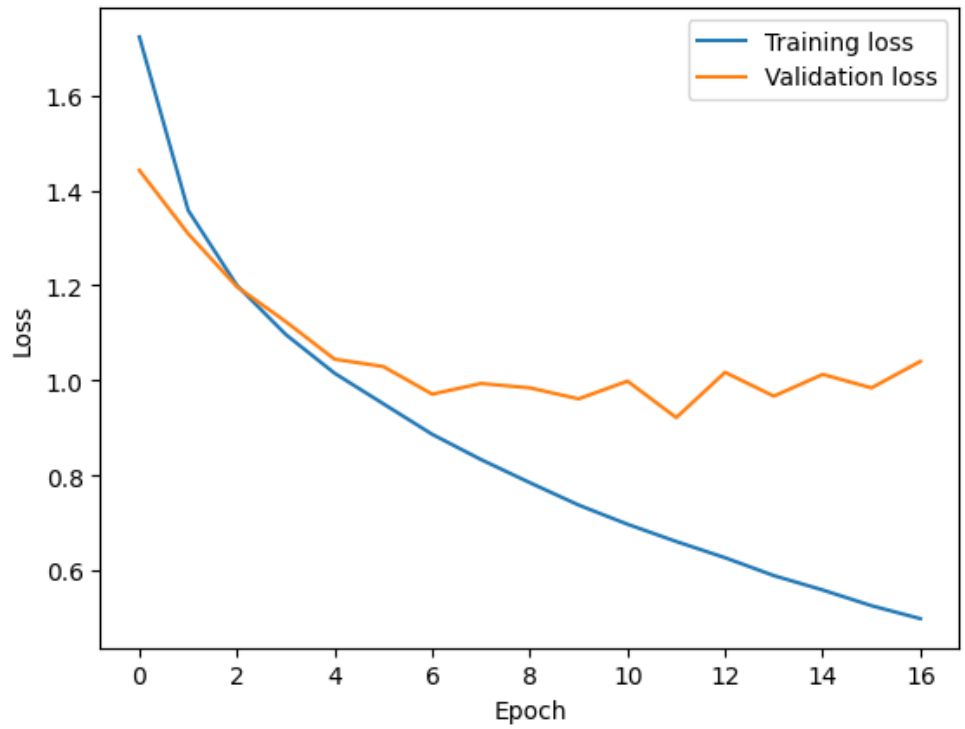

Figure 15: Loss plot for ReLu activation function

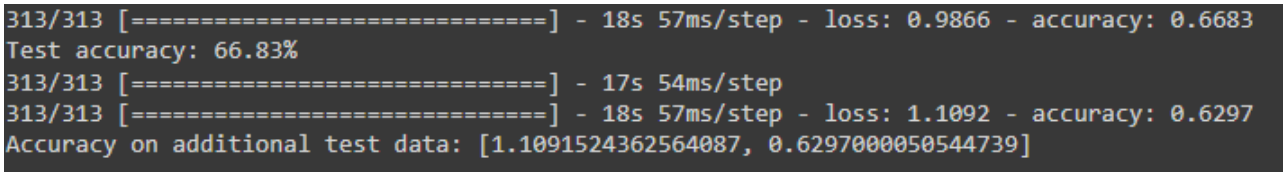

Figure 16: Screenshot of results for ReLu activation function

- Observations and learnings

	ReLu	Sigmoid
Epochs	17	43
Average time taken per epoch	200s	350s
Model Loss	0.9866	1.1874
Model Accuracy %	66.83	58.69
Accuracy % on additional test data	62.9	58.21

Table 1: Comparision between Sigmoid and ReLu models

  - Model with ReLu activation function gave better accuracy than with Sigmoid activation function. It also gave lesser loss and time to execute.

1 Appendix- Link for codes

- Link for Q1 https://colab.research.google.com/drive/1eZ1rs3Ps6zn75chjk7u-maTfPqYXg8Cy#scrollTo=cVqQ1mLfzODW&uniqifier=2
- Link for Q2 https://colab.research.google.com/drive/1NFwDrI50dyEIuNqKgeIYzZ-C5hzXtSTY#scrollTo=lpvDaAVAtbxo