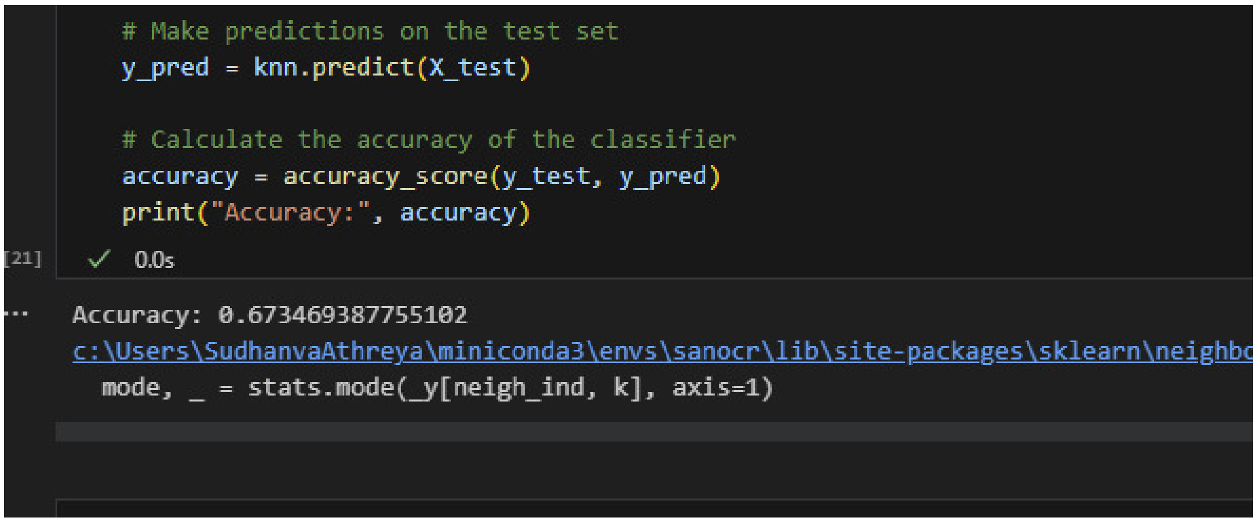
OCR Classifier Model

KNN and SVM Models (9.6.23)

The original dataset consisted of around 20 good labels with around 10 training images per label. The accuracy achieved from these models were around 70% at this time.

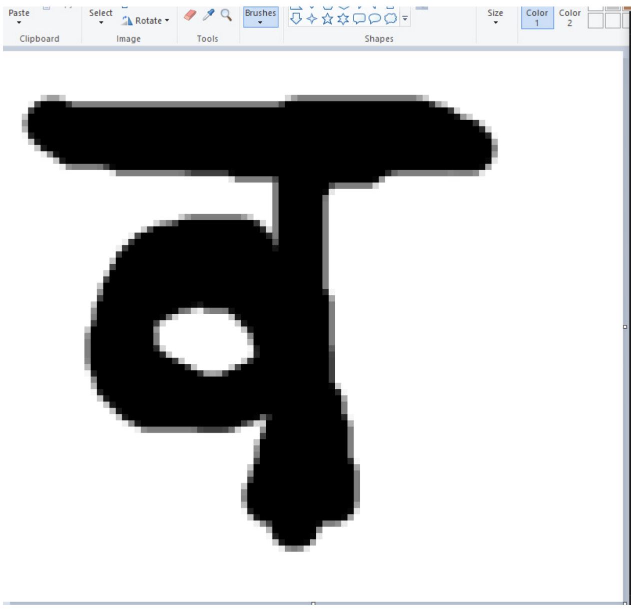


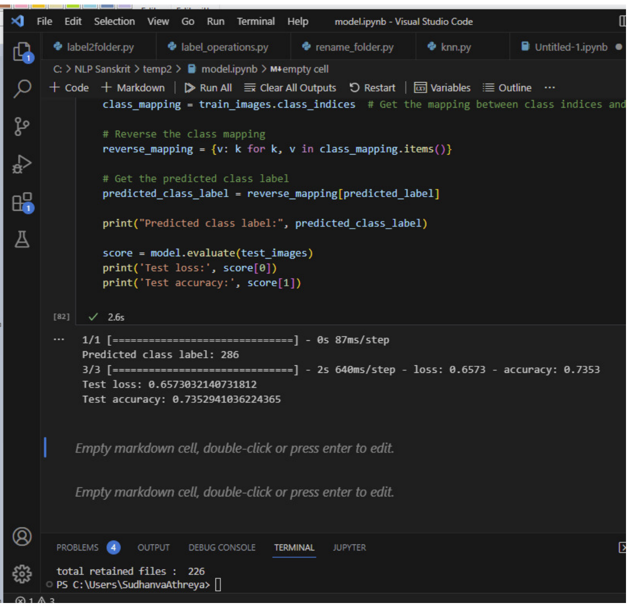
```
# Make predictions on the test set
y_pred = knn.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
[21]  ✓ 0.0s
```

```
Accuracy: 0.673469387755102
c:\Users\SudhanvaAthreya\miniconda3\envs\sanocr\lib\site-packages\sklearn\neighbo
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```





```
class_mapping = train_images.class_indices  # Get the mapping between class indices and

# Reverse the class mapping
reverse_mapping = {v: k for k, v in class_mapping.items()}

# Get the predicted class label
predicted_class_label = reverse_mapping[predicted_label]

print("Predicted class label:", predicted_class_label)

score = model.evaluate(test_images)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

[82] ✓ 2.6s

```
1/1 [==============================] - 0s 87ms/step
Predicted class label: 286
3/3 [==============================] - 2s 640ms/step - loss: 0.6573 - accuracy: 0.7353
Test loss: 0.6573032140731812
Test accuracy: 0.7352941036224365
```

Empty markdown cell, double-click or press enter to edit.

Empty markdown cell, double-click or press enter to edit.

```
total retained files :  226
PS C:\Users\SudhanvaAthreya>
```

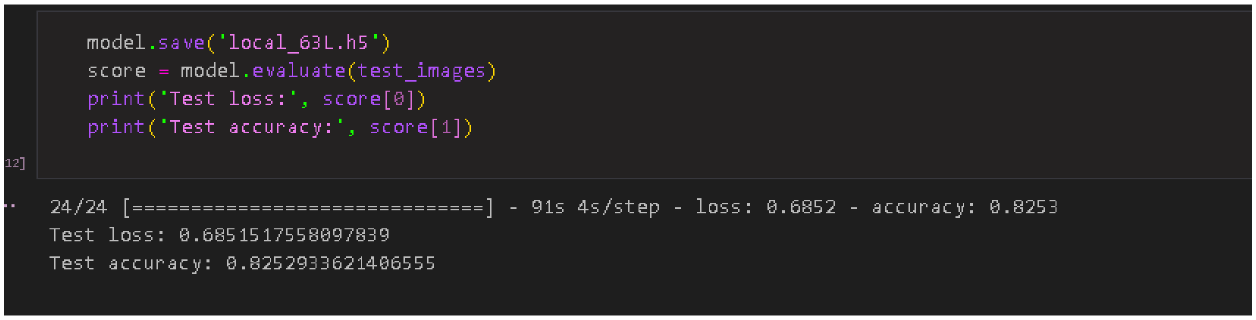
Code can be found here - > https://github.com/sud0x00/SharadaScripture_Classifier-Model

The accuracy however decreased once we had a better dataset , around 50 labels and 10-15 images per label. KNN and SVM couldn't handle multi-label classification well and the dropped accuracy was around 45%.

Then we started to use a **Resnet50** model.

Resnet50 (14.6.23)

By this time we had around 200 classes with at least 10 images per label. The accuracy achieved was around 80%



```
model.save('local_63L.h5')
score = model.evaluate(test_images)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

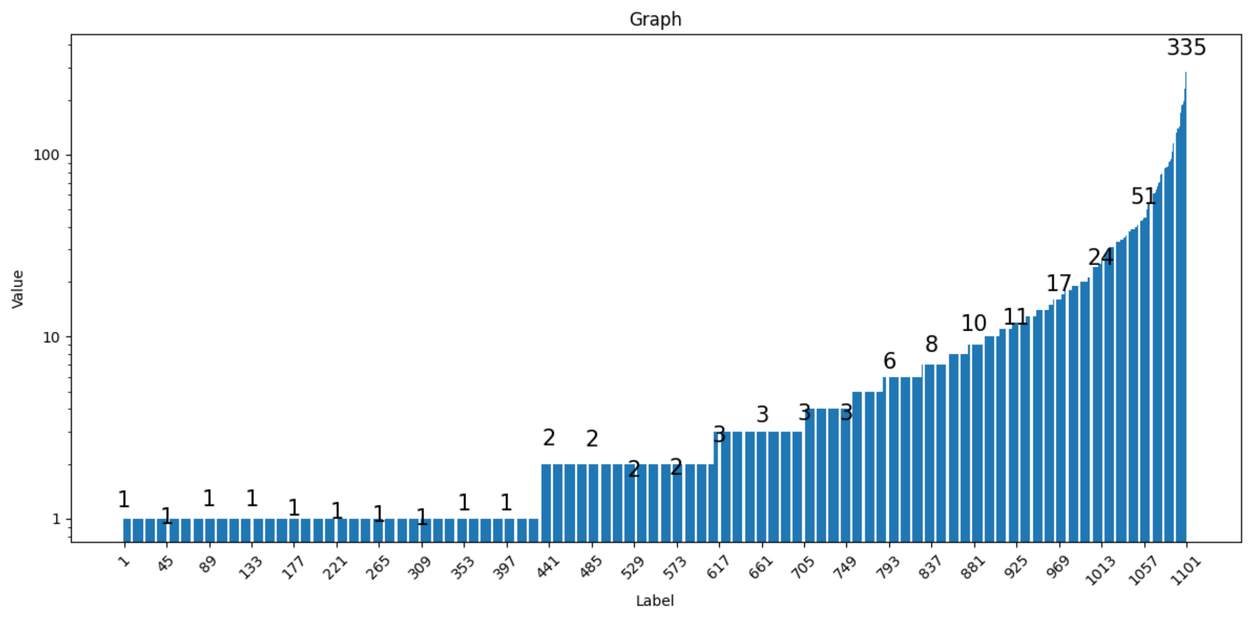
12]

```
24/24 [==============================] - 91s 4s/step - loss: 0.6852 - accuracy: 0.8253
Test loss: 0.6851517558097839
Test accuracy: 0.8252933621406555
```

** In most of the test cases , external images drawn in MS-Paint were used to test how the model held up.**

We received the newly annotated dataset with a total of 16000 images belonging to 1100 labels on 18.6.23

Graph below represents the data and the data imbalance is clearly visible. The reduced accuracy and increased overfitting is due to this.



Reasons to why multiple data augmentation technique will badly affect the model :

Assuming that Keras ImageDataGenerator is used for the augmentations

1. **Random Rotation -** There's a step of **skew correction** which corrects the page skew which makes this kind of augmentation obsolete.
2. **Random Flips** - Flipping the character vertically or horizontally will make the character unrecognizable and sometimes even change the meaning of characters.
3. **Random Shifts -** The annotated **characters are precisely cropped** and performing random shifts on the dataset will result in deformed characters
4. **Random Zoom** - Same as random shifting , performing random zooms will remove the required features which the deep learning model needs.
5. **Random Brightness** - Changing the brightness of the train data won't help at all since the characters are binarized (thresholded) before they are sent for training.
6. **Random Channel Shift** - Same as random brightness , changing the color channel won't help.
7. **Random Erasing -** Same as the above methods , the character might change completely.

Other Image Augmentation Frameworks(such as Augmentor) were also explored.

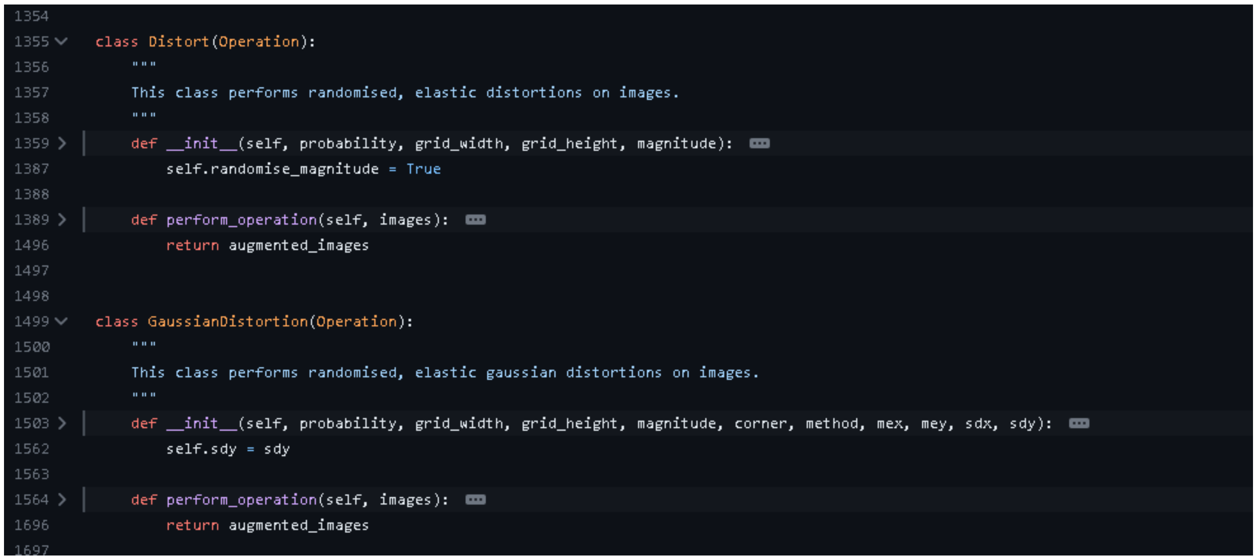
Summary :
1. The deep learning model will start to memorize rather than generalize the image features when augmentations like brightness and channel shifting is performed.
2. Augmentations like rotation , flips , shifts , zoom will have negative effects on indic languages due to the presence of diacritics.

Certain sets of Augmentations were tested on a smaller data set of images. Two tests were performed.
1. The first test involved comparison of an augmented dataset with an imbalanced dataset.
2. The second test was performed using a balanced set of images and then replacing certain images with augmented images.

Augmentations methodologies which can be considered :
1. Shearing: Shearing is a linear transformation that tilts the image along a specific axis. It preserves parallel lines but distorts the shape of objects.
    a. Result : Negative , the accuracy was reduced
    b. The model performed badly on an unseen dataset (Images generated using MS-Paint)

2. Perspective transforms: Perspective transforms involve altering the perspective or viewpoint of an image.
    a. Result : Negative , the accuracy was reduced
    b. The model performed badly on an unseen dataset (Images generated using MS-Paint)
3. Distortion : Elastic and Gaussian Elastic Distortions
    a. Result : Slight Variation in Accuracy -> It varied across labels.
    b. The model performance didn't change much.

The results noted were from the balanced dataset. The accuracy increased after the augmentations in the imbalanced dataset but that doesn't imply that they were the reason for it. The overfitting was avoided and the model could generalize more text. As it can be seen from the balanced dataset that having a good balanced dataset will result in a good accuracy.



```python

class Distort(Operation):
    """
    This class performs randomised, elastic distortions on images.
    """
    def __init__(self, probability, grid_width, grid_height, magnitude): ...
        self.randomise_magnitude = True

    def perform_operation(self, images): ...
        return augmented_images


class GaussianDistortion(Operation):
    """
    This class performs randomised, elastic gaussian distortions on images.
    """
    def __init__(self, probability, grid_width, grid_height, magnitude, corner, method, mex, mey, sdx, sdy): ...
        self.sdy = sdy

    def perform_operation(self, images): ...
        return augmented_images
```

https://github.com/mdbloice/Augmentor/blob/master/Augmentor/Operations.py