REPORT ON

Nepali Currency Classifiers

PROJECT For

Machine Learning (CSE4020)

Slot: F2

Submitted By

Krishna Sah Teli (18BCE2475)

Kamal Subedi (18BCE2479)

Saurav Rauniyar (18BCE2482)

Sushan Gautam (18BCE2488)

Under the guidance of

Prof. Dr. Sharmila Banu K

Assistant Professor Sr. Grade 1

(SCHOOL OF COMPUTER SCIENCE AND ENGINEERING)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

May, 2021

1. INTRODUCTION

Paper currency recognition is one of the important sector of pattern or chain or rule based recognition. Paper recognition has various potenial applications which includes e-banking, currency monitoring system, money exchange machines and more. Every currency paper is different from one another, feature set are different, and so is specific regional currency mark and color. Its difficult for native people to find out whether the paper currency is fake or real so, one can imagine how difficult would it be for tourists, visually impaired individuals. The vitality of the currency in countries across the globe is known to the all of us. Entire economy of the country depends on currency. Thus, this project aims at automating the currency identifier tool such that it makes an impact in day to day life. Ther are reasearchers, developers trying to make things digitally easily accesible to the people and for the people. Currency has been detected so far using hardware embedded devices which cannot be owned by a mediocre people. The long term objective of this project is to deploy an android application, web based application and making things user friendly.But for the time being, this project aims at building a neuron network for Nepali currency.

The Nepalese currency does not have any special feature in them, except for the watermark and number on the bottom of the bill, so its difficult for a normal people to recognize one can imagine how difficult would it be for the visually impaired and tourist- someone who haven't seen the currency before can differentiate between the notes of different values. The visually impaired people, unlike us, face difficulties in day to day monetary transactions since they cannot recognize the bank notes as easily as we can in Nepal. As an initiative for the Nepalese visually impaired community, this project will aid them to recognize notes without any hassle. By uploading a bank note, this ML model will recognize it and display the value of the note. This project would be extended to the android app soon so that, by hovering a smartphone over a note, that app could recognize and play an audio enabling the user to hear and know the value of the note.

2. LITERATURE SURVEY FROM GITHUB

To know the basic concept behind the image processing and image input for training the learning algorithm, we have gone through many previously existing methodologies. In many research paper, we came to know about how fake currency has been recognized using deep learning and neural network. The papers [5], [6],[7] describes different methodologies which are used to detect as well recognize fake currencies.

[1] describes the neural based recognition and verification techniques which is used in banknote machine. This also describes this technology is growing and many countries are implementing for accepting paper currency. Authors have used verification and classification steps in different multiple layers perceptrons. They have trained learning algorithm and found out that the methodology is well suited for performing the bank note recognition and verification. Their experimental result shows the cost machine is quite low and neural netwok technology is likely to be very effective also when non-sophisticated sensors report.

[2] paper describes the different approaches for image recognition like CNN, SSD, MLP, etc. They have explained like system can also recognize and authenticate currency notes as human being do with the help of their visual capability. Here authors have used deep learning technique to perform experiment. They have successfully employed this technique to improve the accuracy of currency detection. They have also followed the common methodology like feature extraction, dividing dataset into test and training set, perform currency classification based on extracted features and finally identification.

[3]shows the paper currency verification system based on characteristic extraction using image processing. Here in this paper, authors have tried to overcome the counterfeiting issues in bank notes. Counterfeiting is most serious problems since anyone can print out the bank note easily with the help of laser printer. Authors have designed a system that can easily recognize the paper currency notes with fast speed and in less time. They have used different approaches in order to develop the system, image processing technique, edge detection, image segmentation, characteristic extraction, comparing images. They have done using MATLAB to detect the features of paper currency. In this way they have successfully overcome counterfeiting problems of bank note.

[4] paper describes a new model for paper currency recognition. This is mathematical model called Gaussian Mixture model (GMM) which is very much popular tool for density estimation. Basically the parameters of GMM are estimated based on maximum likelihood principle. This model helps to improve the effectiveness, accuracy, precision and F1 score. This paper has done image recognition using GMM based on structural risk minimization. As a result, this methodology is more flexible alternative and lead to improve currency recognition.

There are more different image processing methodologies which used to detect fake currency. In paper [5], authors have tried to overcome the counterfeiting issues by detection fake image using image processing. They have done findings some methodologies like water marking, optical variable ink, security thread, latent image and techniques like counterfeit detection pen, which make ensure to detect fake currency. Similar process and methodology has been described in paper[6] to detect the fake currency. Here authors have done detection over variation in barcode among the real and fake one.

3. GAP IDENTIFIED

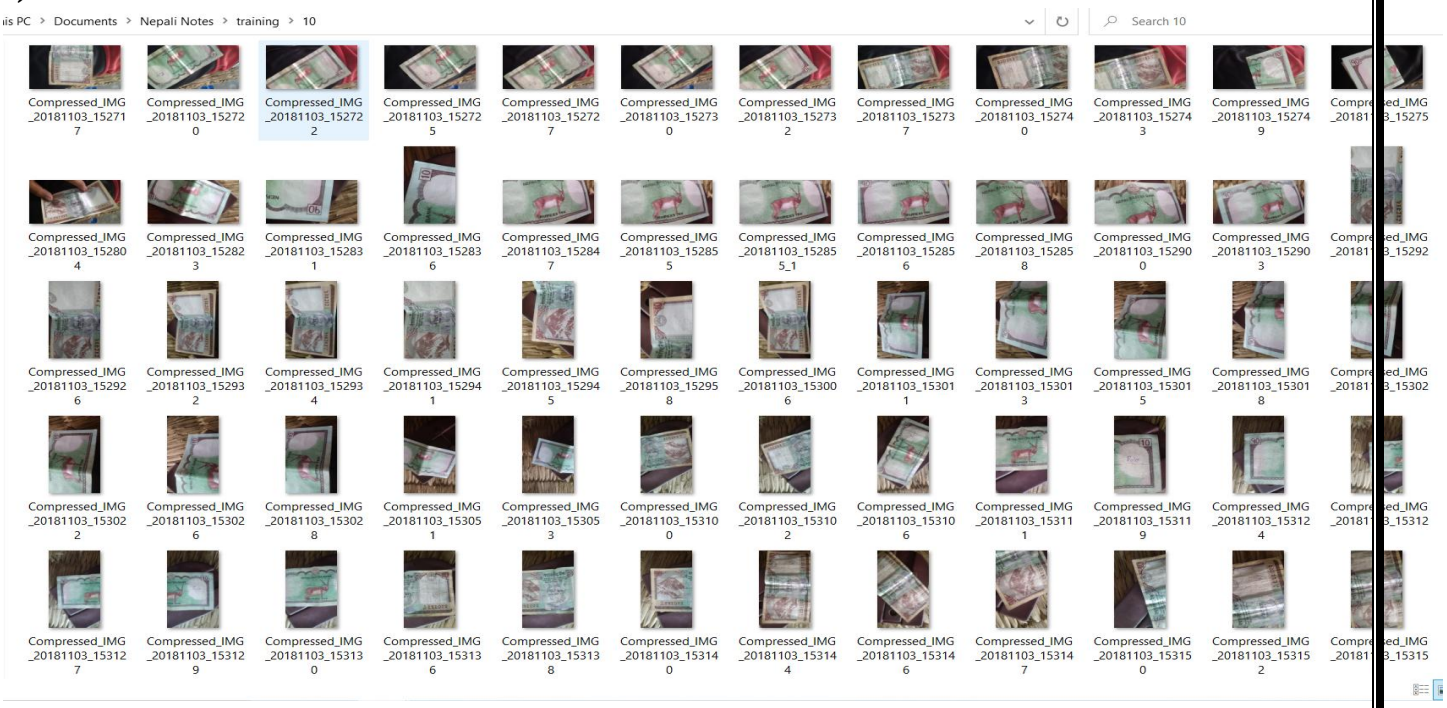
Since we have gone through various research papers and we came to know many new algorithms and methodologies for detecting as well recognizing paper currency. In many countries, their banks have been using ML and deep learning based developed tools to avoid counterfeit and other discrepancies by comparing with original one. Although many countries have implemented those technologies to avoid counterfeit by detecting as well recognizing currency, there has not been done to recognize or detect different values notes. Hence in this we are aiming to develop a model which can recognize Nepali rupees easily with high accuracy and precision. Further more our work can also be extended to develop android app which can provide service like recognizing as well acts as assistance for visually impaired people.

4. DATASET PREVIEW

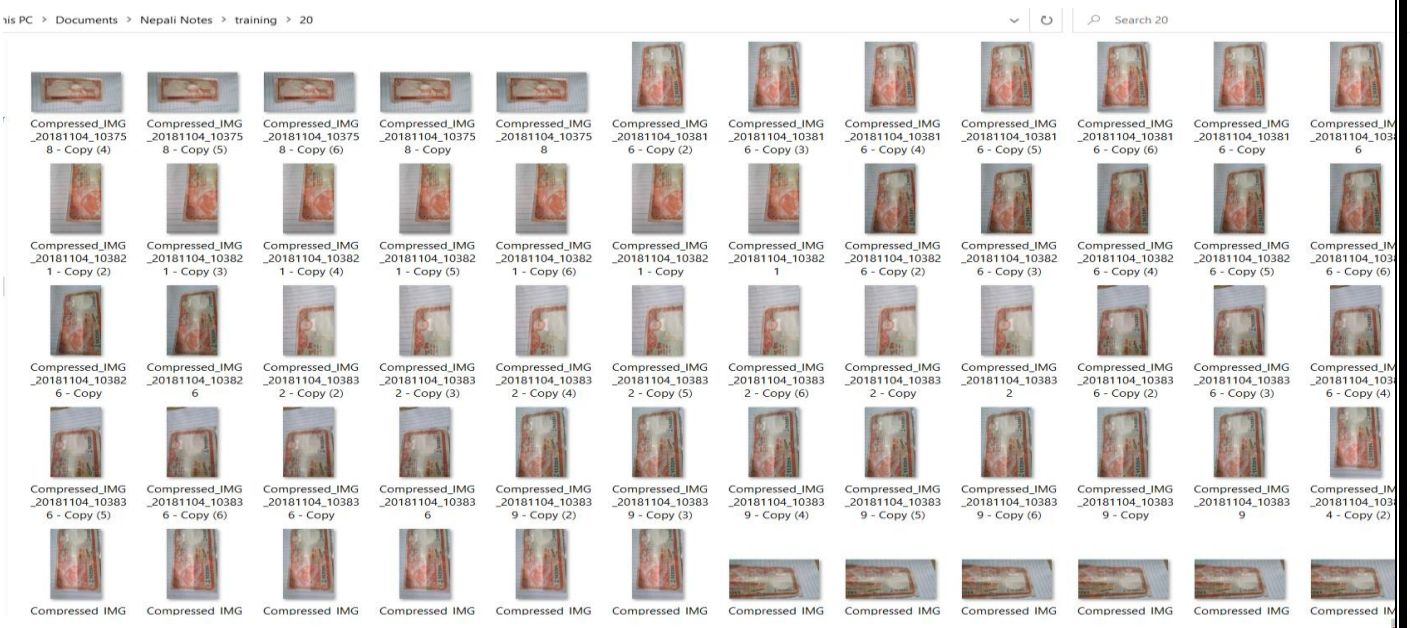
In order to make a solution that will work on realistic images taken from smart phone devices, we would require the deep neural networks to be trained on such images. So, to start I took pictures of the notes of two currency categories only: Rs.10 notes and Rs.20 notes. That way I would quickly know if the approach I am taking is right or not. I gathered about 200 pictures from each category.

Data Set Link: https://drive.google.com/drive/folders/1izqS8l-_TVIEbKF7Wrt9ywIkRVl3BsLD

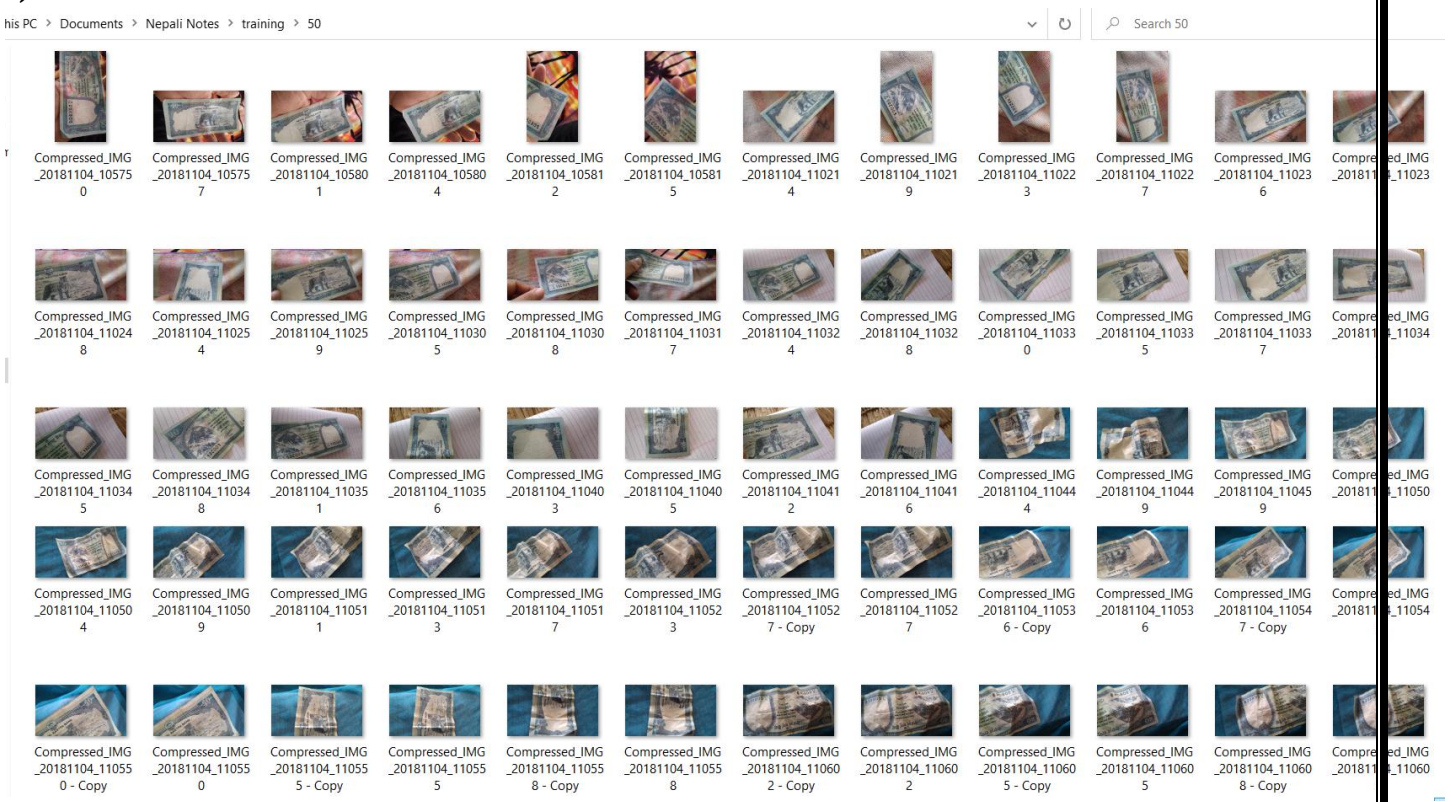
a) Rs 10 Note:



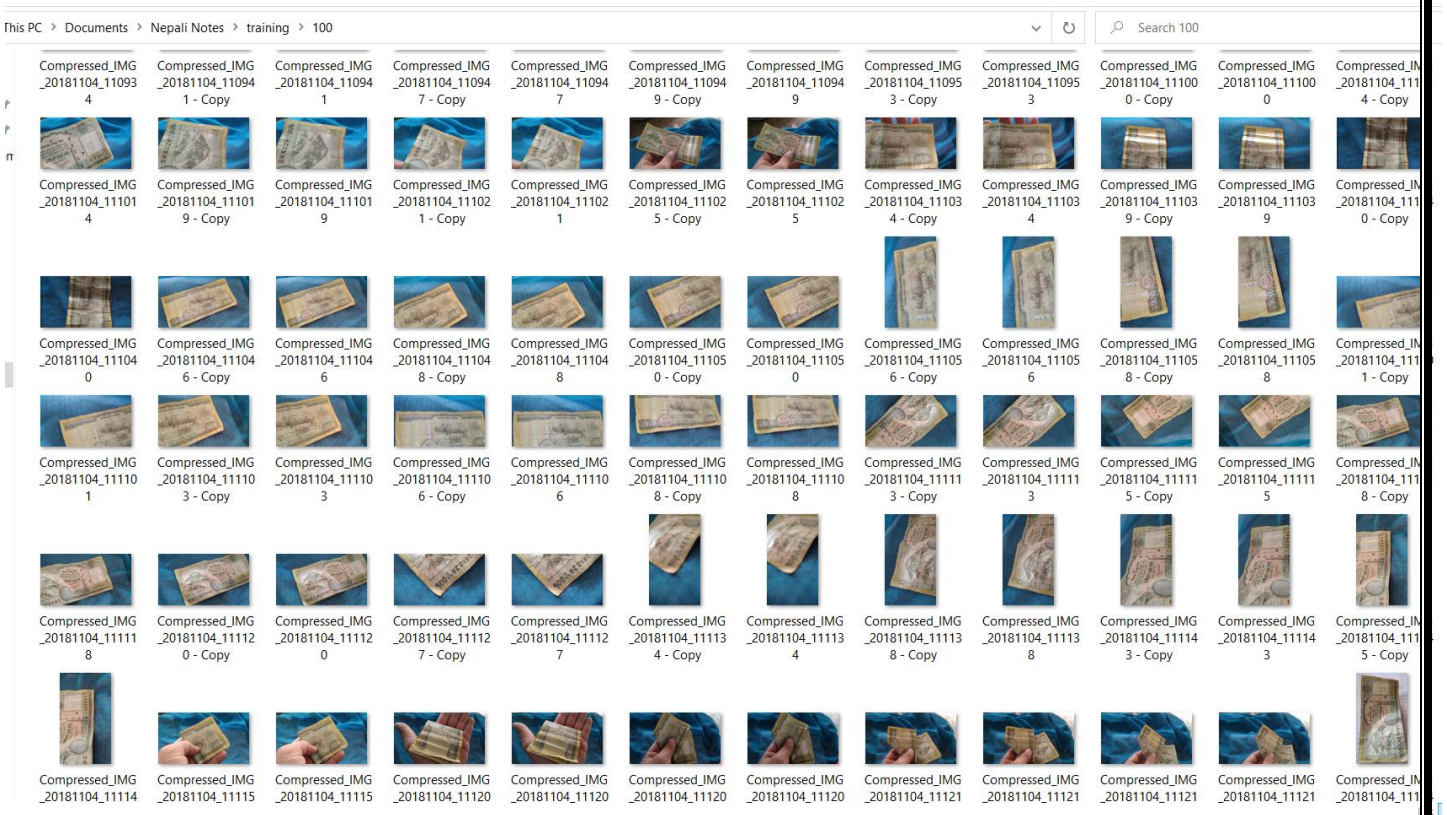
b) Rs 20 Note:



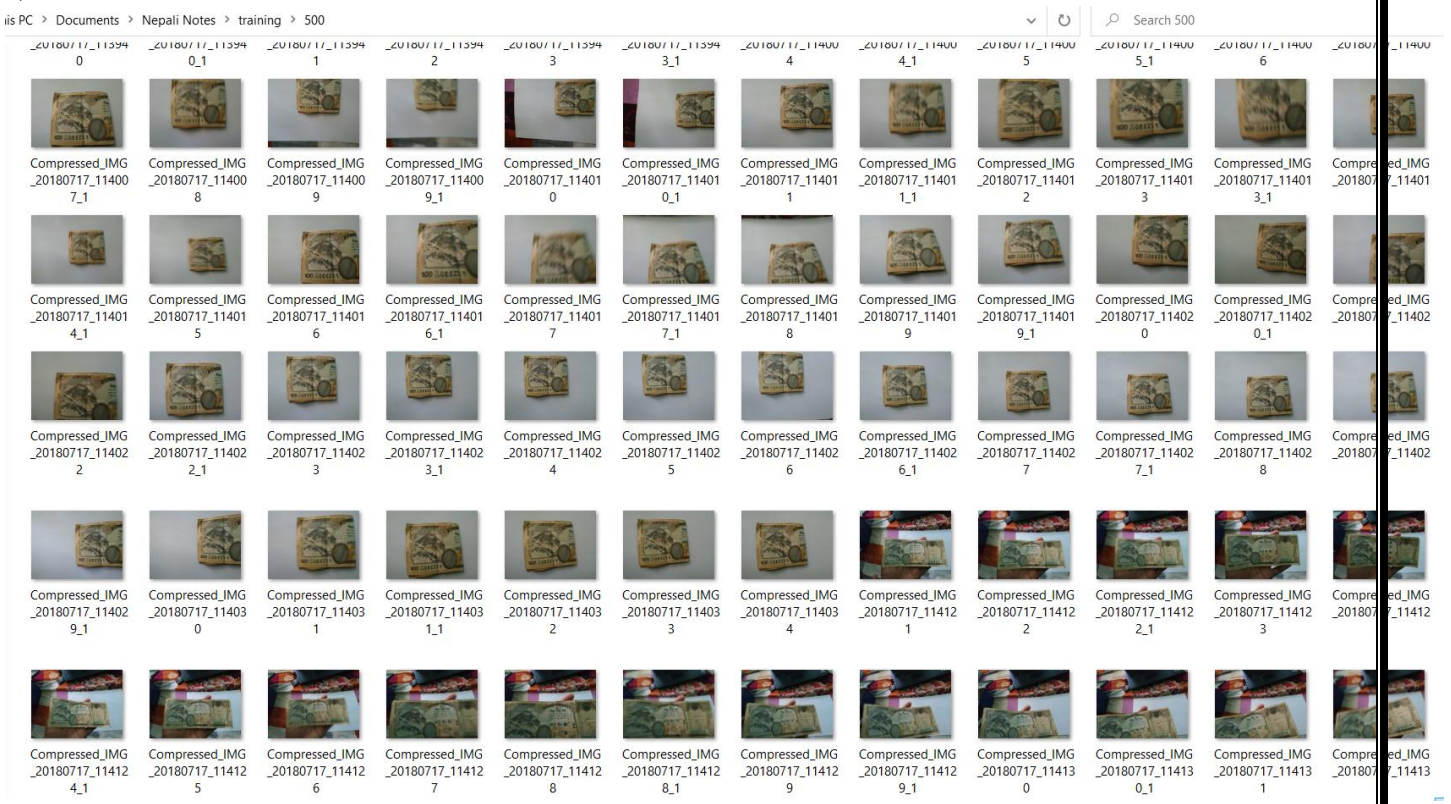
c) Rs 50 Note:



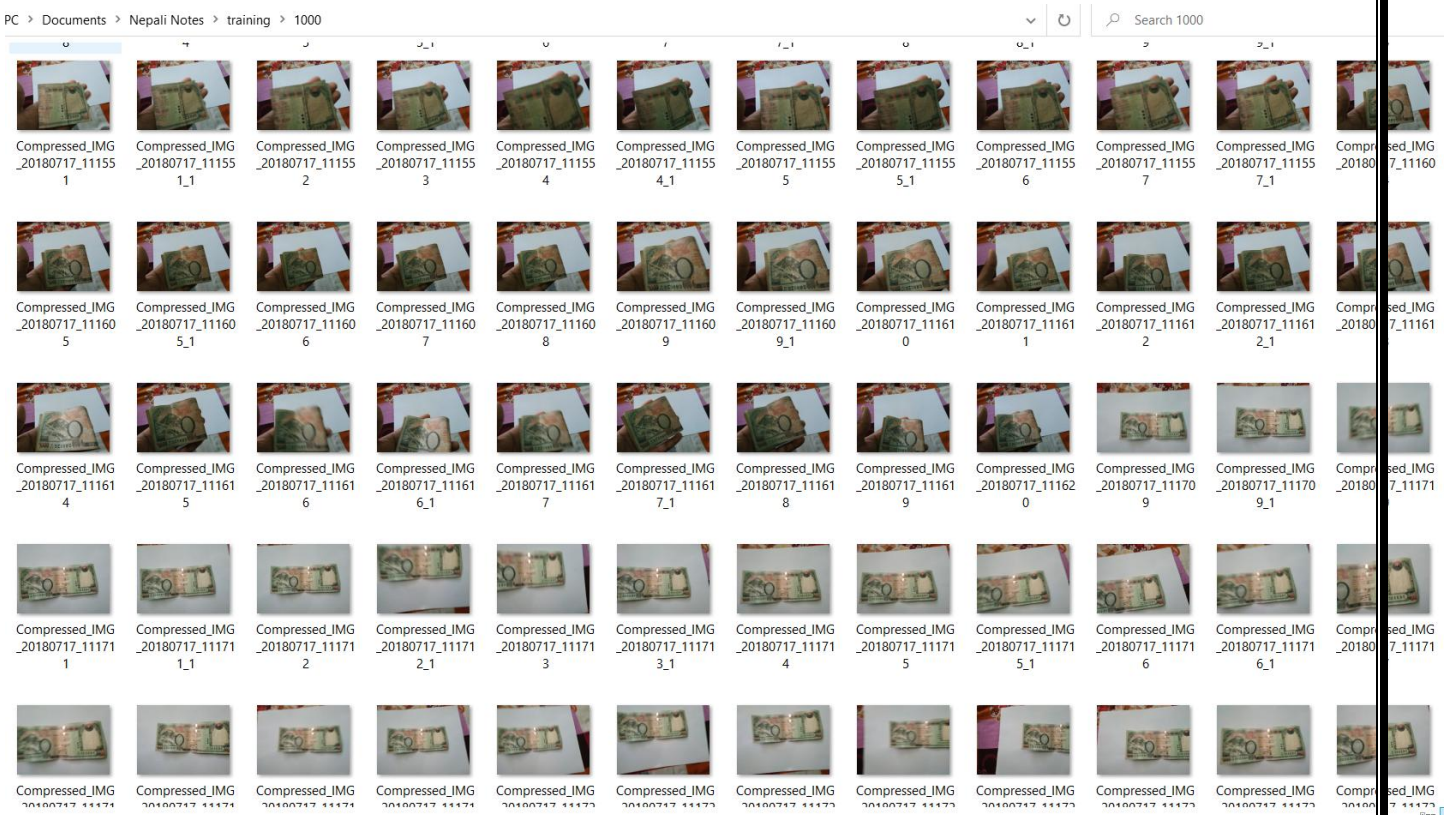
d) Rs 100 Note:



e) Rs 500 Note:



f) Rs 1000 Note:



5. PROCESS

5.1 DATA COLLECTION AND PREPROCESSING

Some previously collected data turned out to be noisy and shaky, negatively affecting the loss value in the training process. To eliminate the impact and optimize the training process, I needed to revisit the data collected. The affected data was recurated for the final dataset, which included all six categories of Nepalese cash notes. In each category, there were approximately 6,480 datasets available, of which approximately 1,500 were used for training and approximately 480 were used for validation. In total, around 6,480 datasets of all seven categories of Nepalese cash notes were manually collected.

For the final model, TF.Keras's **ImageDataGenerator** class was used for an easy dataset generator pipeline. In the training dataset, augmentation was introduced, whereas the validation set was not augmented but only rescaled. Data augmentation introduced in training sets were rescaled and resized to 150x150.

```
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
```

5.2 MODEL CONSTRUCTION

After preprocessing the dataset and creating the data generator for the training process, final model architecture was constructed.

5.2.1 Keras Sequential Class was used to construct model

The Keras **sequential** class helps to form a cluster of a layer that is linearly stacked into tf.keras.Model. The features of training and inference are provided by sequential to this model.

If we have a simple model where each layer is **sequentially connected** from the input layer until the output layer, then we can use the sequential model. Basically in this model, there is a **single chain of layers** from the input to the output and there are **no layers that have multiple inputs**. This model also needs to have a **single input** and a **single output.**

This is how we create a model using the Sequential model in tf.keras. It is a very simple and straightforward API and can be used to quickly build models that have a linear structure.In the case of Sequential models, the process is simple:

- Initialize the sequential model.
- Add layers in the correct order by model.add()

```
model = tf.keras.models.Sequential([

  tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
```

```
tf.keras.layers.MaxPooling2D(2, 2),

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Conv2D(128, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

#tf.keras.layers.Conv2D(128, (3,3), activation='relu'),

#tf.keras.layers.MaxPooling2D(2,2),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(250, activation='relu'),

tf.keras.layers.Dense(6, activation='softmax') ])
```

A code snippnet of the sequential model creation has been pasted above. Then we feed our 150 X 150 pixels size image into the Convolution layer of 2D image where we convert 2D image into spatial image. 32 filters are used having a size of 3 X 3. ReLU activation function is used. And also the image is 3 which mean RGB image is being fed into the twin layer of the sequential model. MaxPooling takes the top layer or the main features extracted from convolution layers of size (2,2).

Then, the ouput of twin layer is fed as input to the first hidden layer then same Maxpooling layer filters the layer sequentially. In first hidden layer the convolution layer filter size is 64, second layer and third layer has 128 filters.

Now, after we have extracted all the top layer features of the hidden layer, we feed it into the Flatten layer() where, the output of the Maxpooling layer has many top layer features each having multiple column. This flatten layer, converts or flatten these multilple column into a single column.

Finally, the fully connected Dense layer comes into picture. Initially, the Dense layer has 250 units of neural after filtering the output of relu activated function with 250 neurons, the Dense layer is fed to 6 neuron (our target output size) into softmax funtion. Softmax function converts the last layer classification network value into a probability distribution ranging from 0 to 1 and summing the values to 1.

```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

input_layer = Input((2,))

x = Dense(4, activation = 'relu') (input_layer)

x = Dense(4, activation = 'relu') (x)

output_layer = Dense(1, activation = 'sigmoid) (x)

model = Model(input_layer, output_layer)
```

Functional API

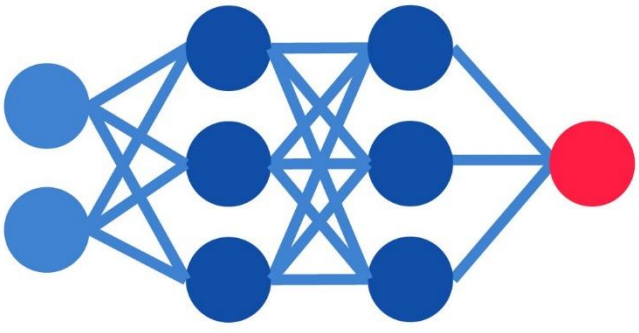
Layer Object

Previous layer which this layer is connected to

Input layer (list of layers if multiple inputs)

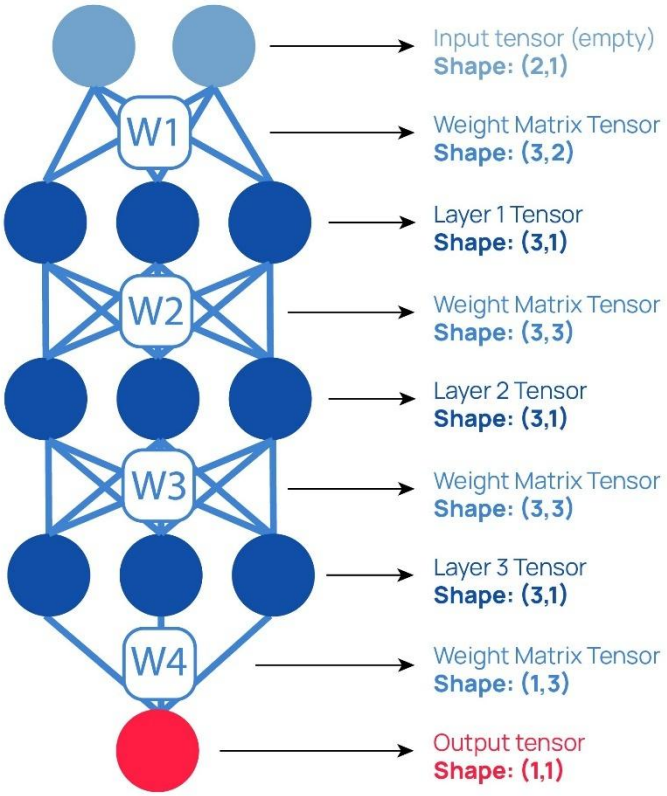
Output layer (list of layers if multiple outputs)

Resultant Model:



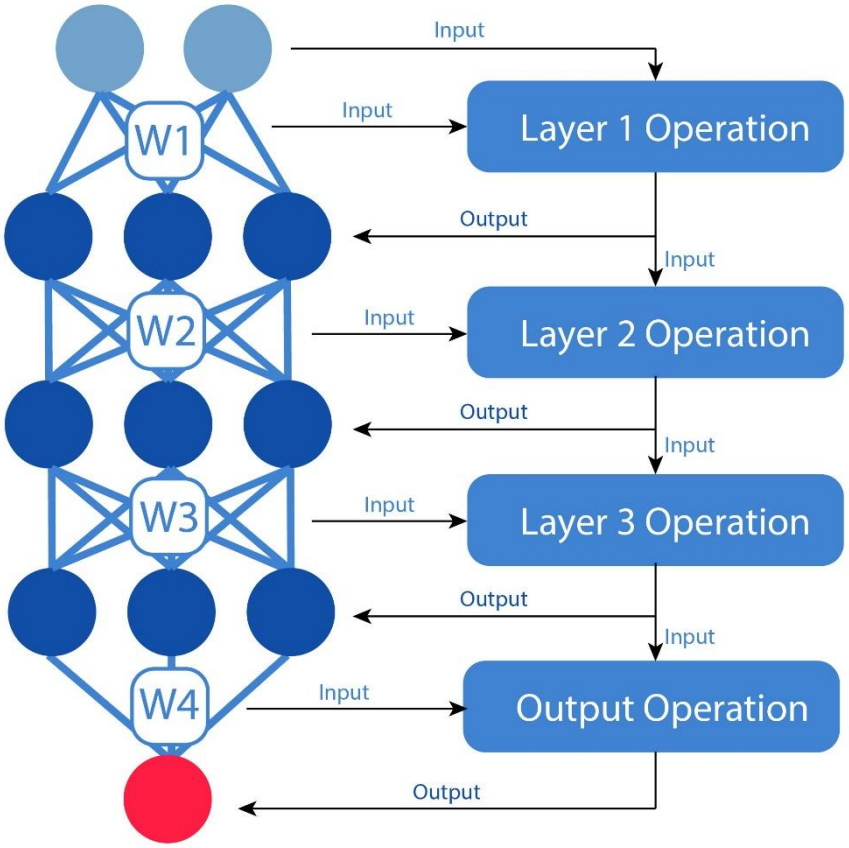
Tensors

Tensors are **multi-dimensional arrays** containing **numerical values** of a certain datatype (int, float, etc). All **numeric data** in the neural network is represented with tensors. Each tensor can contain **multiple dimensions** and will have a **fixed shape**. In certain situations, the tensor will be empty and will take values that we input. This happens at the input layer tensor, it will initially be an empty tensor of a certain shape and it will have values that are passed to it. Then this tensor with values will be passed to the rest of the neural network. These empty tensors which we have to provide values for are called **placeholder** tensors.



Operations

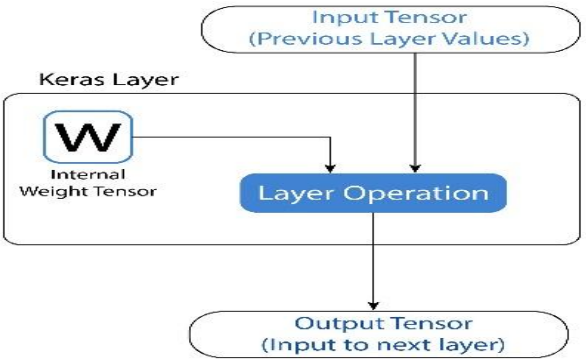
Tensors only contain numerical values, but in a neural network, we need to perform some computation of the numbers to get the final output value of the neural network. We define these computations in the form of **operations.** Operations are simple, they take **tensors as input**, perform certain **mathematical operations** on these tensors, and then **output a tensor**. Operations are essentially the functions that connect all the tensors in a neural network so that it forms a **long chain of computation** from the input tensor to the output tensor.



Here, each layer is essentially an operation that takes two input tensors: the **weight tensor** of that layer and the tensor containing the values of the **previous layer**. The operation will perform the necessary mathematical computation, which in this case is **matrix multiplication** of the two input tensors followed by an **activation function**. This calculation will result in another tensor, which is then **outputted** by the operation. This output tensor essentially contains the **values of the current layer** and this is also an **input tensor to the next layer**.

So essentially, in a neural network, we have several tensors containing the parameters (weights) of the network and several operations take the tensor provided from the input layer and perform computations till the end of the network. Now the name "TensorFlow" might make more sense because deep learning models are essentially a flow of tensors through operations from input to output.

5.3 KERAS LAYERS



When building a neural network with tf.keras is to **define each layer** and **define how they are connected** with the other layers. Each layer will take input from some previous layers except for the input layer.

5.5.1 flow_from_directory Method

This method will identify classes automatically from the training folder name.

```
In [10]: train_generator = train_datagen.flow_from_directory(
             train_dir,  # This is the source directory for training images
             target_size=(150, 150),  # All images will be resized to 150x150
             batch_size=1,
             # Since we use binary_crossentropy loss, we need binary labels
             class_mode='categorical')

         Found 6000 images belonging to 6 classes.
```

This method will identify classes automatically from the test folder name.

```
In [11]: validation_generator = test_datagen.flow_from_directory(
             validation_dir,
             target_size=(150, 150),
             batch_size=1,
             class_mode='categorical')

         Found 480 images belonging to 6 classes.
```

5.4 Training and Validation Accuracy & Loss

5.4.1 Training From Generated Model – 1st

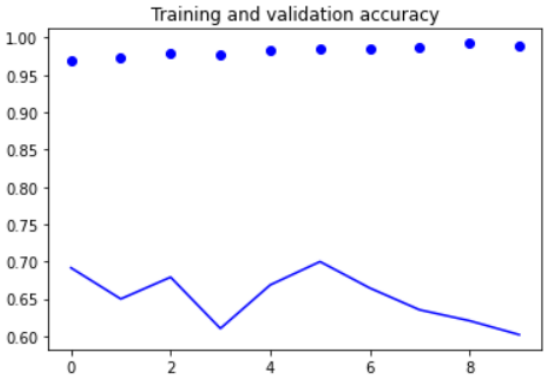
Steps per epoch for training – 6000

Steps per epoch for testing – 480

Epochs Used – 10

```
history = model.fit_generator(
    train_generator,
    steps_per_epoch=6000,  # images = batch_size * steps
    epochs=10,
    validation_data=validation_generator,
    validation_steps=480,  # images = batch_size * steps
    verbose=1)
```

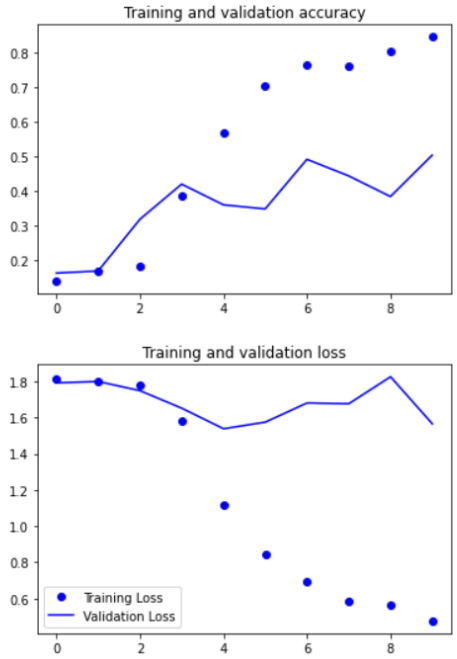
```
In [13]: history = model.fit_generator(
             train_generator,
             steps_per_epoch=6000,  # images = batch_size * steps
             epochs=10,
             validation_data=validation_generator,
             validation_steps=480,  # images = batch_size * steps
             verbose=1)

Epoch 1/10
6000/6000 [==============================] - 346s 58ms/step - loss: 0.1167 - accuracy: 0.9687 - val_loss: 1.9066 - val_accuracy: 0.6917
Epoch 2/10
6000/6000 [==============================] - 342s 57ms/step - loss: 0.1054 - accuracy: 0.9737 - val_loss: 2.2836 - val_accuracy: 0.6500
Epoch 3/10
6000/6000 [==============================] - 324s 54ms/step - loss: 0.0826 - accuracy: 0.9788 - val_loss: 2.8706 - val_accuracy: 0.6792
Epoch 4/10
6000/6000 [==============================] - 327s 54ms/step - loss: 0.1016 - accuracy: 0.9782 - val_loss: 3.2215 - val_accuracy: 0.6104
Epoch 5/10
6000/6000 [==============================] - 349s 58ms/step - loss: 0.0780 - accuracy: 0.9838 - val_loss: 3.6516 - val_accuracy: 0.6687
Epoch 6/10
6000/6000 [==============================] - 340s 57ms/step - loss: 0.0642 - accuracy: 0.9850 - val_loss: 2.9857 - val_accuracy: 0.7000
Epoch 7/10
6000/6000 [==============================] - 339s 56ms/step - loss: 0.0785 - accuracy: 0.9853 - val_loss: 6.0287 - val_accuracy: 0.6646
Epoch 8/10
6000/6000 [==============================] - 324s 54ms/step - loss: 0.0956 - accuracy: 0.9870 - val_loss: 4.5039 - val_accuracy: 0.6354
Epoch 9/10
6000/6000 [==============================] - 329s 55ms/step - loss: 0.0548 - accuracy: 0.9925 - val_loss: 5.2123 - val_accuracy: 0.6208
Epoch 10/10
6000/6000 [==============================] - 315s 52ms/step - loss: 0.0613 - accuracy: 0.9897 - val_loss: 5.7304 - val_accuracy: 0.6021
```



5.4.2 Training From Generated Model – 2nd

Epochs Used – 20

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/engine/training.py:1940: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/20
6000/6000 [==============================] - 1890s 310ms/step - loss: 1.1824 - accuracy: 0.5490 - val_loss: 1.3993 - val_accuracy: 0.4875
Epoch 2/20
6000/6000 [==============================] - 490s 82ms/step - loss: 0.3958 - accuracy: 0.8760 - val_loss: 1.4366 - val_accuracy: 0.4625
Epoch 3/20
6000/6000 [==============================] - 478s 80ms/step - loss: 0.2388 - accuracy: 0.9225 - val_loss: 1.6206 - val_accuracy: 0.6687
Epoch 4/20
6000/6000 [==============================] - 472s 79ms/step - loss: 0.1506 - accuracy: 0.9502 - val_loss: 1.4678 - val_accuracy: 0.6542
Epoch 5/20
6000/6000 [==============================] - 471s 79ms/step - loss: 0.1567 - accuracy: 0.9622 - val_loss: 2.5090 - val_accuracy: 0.6187
Epoch 6/20
6000/6000 [==============================] - 471s 78ms/step - loss: 0.0866 - accuracy: 0.9787 - val_loss: 3.9742 - val_accuracy: 0.6000
Epoch 7/20
6000/6000 [==============================] - 469s 78ms/step - loss: 0.1064 - accuracy: 0.9777 - val_loss: 3.0811 - val_accuracy: 0.6104
Epoch 8/20
6000/6000 [==============================] - 468s 78ms/step - loss: 0.0705 - accuracy: 0.9827 - val_loss: 4.0050 - val_accuracy: 0.5729
Epoch 9/20
6000/6000 [==============================] - 468s 78ms/step - loss: 0.0923 - accuracy: 0.9830 - val_loss: 5.0668 - val_accuracy: 0.6438
Epoch 10/20
6000/6000 [==============================] - 468s 78ms/step - loss: 0.0970 - accuracy: 0.9828 - val_loss: 6.7178 - val_accuracy: 0.6354
Epoch 11/20
6000/6000 [==============================] - 470s 78ms/step - loss: 0.0841 - accuracy: 0.9863 - val_loss: 5.8344 - val_accuracy: 0.6458
Epoch 12/20
6000/6000 [==============================] - 470s 78ms/step - loss: 0.0478 - accuracy: 0.9903 - val_loss: 5.1437 - val_accuracy: 0.6313
Epoch 13/20
6000/6000 [==============================] - 472s 79ms/step - loss: 0.0649 - accuracy: 0.9902 - val_loss: 5.9583 - val_accuracy: 0.6417
Epoch 14/20
6000/6000 [==============================] - 477s 79ms/step - loss: 0.0620 - accuracy: 0.9890 - val_loss: 7.9089 - val_accuracy: 0.5688
Epoch 15/20
6000/6000 [==============================] - 491s 82ms/step - loss: 0.0695 - accuracy: 0.9895 - val_loss: 10.3382 - val_accuracy: 0.6062
Epoch 16/20
6000/6000 [==============================] - 473s 79ms/step - loss: 0.0719 - accuracy: 0.9933 - val_loss: 7.6254 - val_accuracy: 0.6417
Epoch 17/20
6000/6000 [==============================] - 473s 79ms/step - loss: 0.0665 - accuracy: 0.9903 - val_loss: 6.8434 - val_accuracy: 0.6250
Epoch 18/20
6000/6000 [==============================] - 478s 80ms/step - loss: 0.0540 - accuracy: 0.9937 - val_loss: 8.8634 - val_accuracy: 0.5604
Epoch 19/20
6000/6000 [==============================] - 479s 80ms/step - loss: 0.0620 - accuracy: 0.9925 - val_loss: 11.2156 - val_accuracy: 0.6313
Epoch 20/20
6000/6000 [==============================] - 477s 79ms/step - loss: 0.1151 - accuracy: 0.9885 - val_loss: 8.6058 - val_accuracy: 0.6646
```



5.4.3 Keras model compiling

Model Accuracy = 99 % & Loss = 0.06

After we have made a model using Sequential method, we then need to compile the model before we can start feeding the model some data to train on. By compiling a model we essentially are defining three things for the model:

- The loss function which it will use at the output layers.
- The optimizer which will be used to train the model.
- The metrics which will be used to evaluate the model.

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

This is how we can compile a model after we have made it. If we have multiple output layers and we require a different loss function for each layer, then we can pass a list to the loss parameter in the same order as the list we pass for the output layers while making the model.

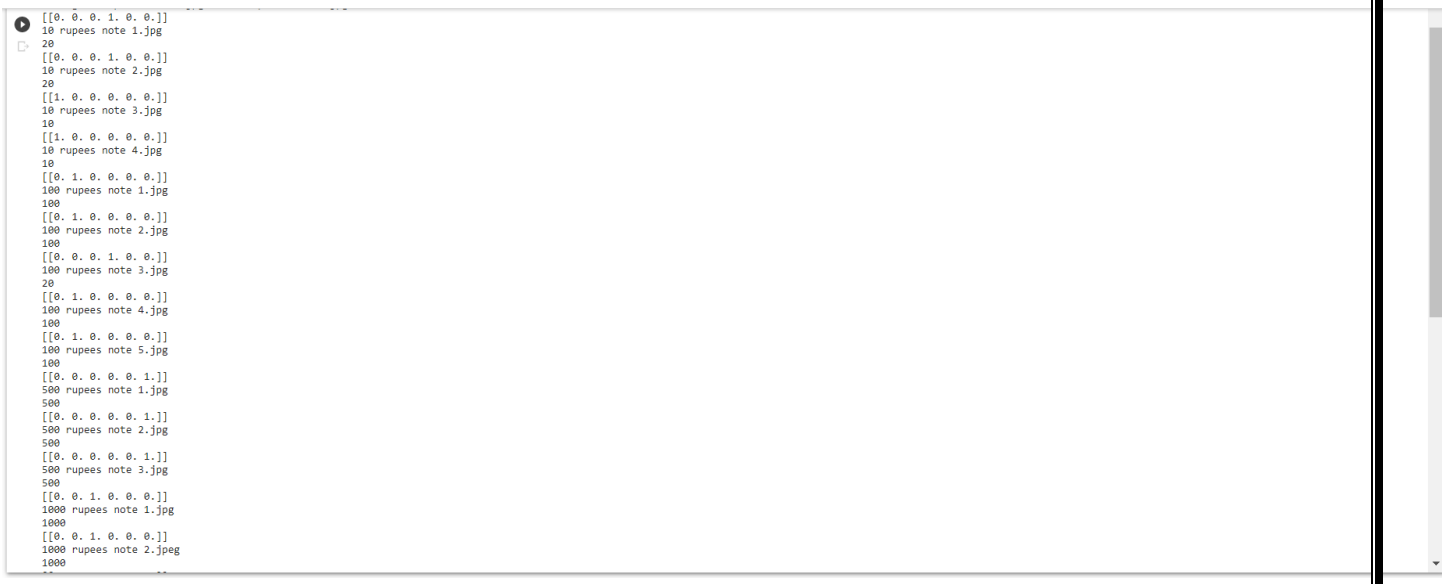
EXPECTED OUTCOMES :-

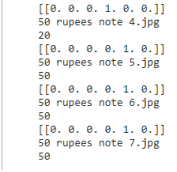
```
[15] Choose Files 30 files
```

- **10 rupees note 1.jpg**(image/jpeg) - 670517 bytes, last modified: 5/31/2021 - 100% done
- **10 rupees note 2.jpg**(image/jpeg) - 715351 bytes, last modified: 5/31/2021 - 100% done
- **10 rupees note 3.jpg**(image/jpeg) - 530846 bytes, last modified: 5/31/2021 - 100% done
- **10 rupees note 4.jpg**(image/jpeg) - 119419 bytes, last modified: 5/31/2021 - 100% done
- **100 rupees note 1.jpg**(image/jpeg) - 1093117 bytes, last modified: 5/31/2021 - 100% done
- **100 rupees note 2.jpg**(image/jpeg) - 452175 bytes, last modified: 5/31/2021 - 100% done
- **100 rupees note 3.jpg**(image/jpeg) - 1096446 bytes, last modified: 5/31/2021 - 100% done
- **100 rupees note 4.jpg**(image/jpeg) - 1226609 bytes, last modified: 5/31/2021 - 100% done
- **100 rupees note 5.jpg**(image/jpeg) - 3303778 bytes, last modified: 5/31/2021 - 100% done
- **500 rupees note 1.jpg**(image/jpeg) - 320563 bytes, last modified: 5/31/2021 - 100% done
- **500 rupees note 2.jpg**(image/jpeg) - 352992 bytes, last modified: 5/31/2021 - 100% done
- **500 rupees note 3.jpg**(image/jpeg) - 503878 bytes, last modified: 5/31/2021 - 100% done
- **1000 rupees note 1.jpg**(image/jpeg) - 630522 bytes, last modified: 5/31/2021 - 100% done
- **1000 rupees note 2.jpeg**(image/jpeg) - 93578 bytes, last modified: 5/31/2021 - 100% done
- **50 rupees note 1.jpg**(image/jpeg) - 980945 bytes, last modified: 5/31/2021 - 100% done
- **50 rupees note 2.jpg**(image/jpeg) - 65313 bytes, last modified: 5/31/2021 - 100% done
- **20 rupees note 1.jpg**(image/jpeg) - 709691 bytes, last modified: 5/31/2021 - 100% done
- **20 RS. REP - Copy (2).jpg**(image/jpeg) - 64184 bytes, last modified: 5/31/2021 - 100% done
- **20 rupees note 2.jpg**(image/jpeg) - 157276 bytes, last modified: 5/31/2021 - 100% done
- **20 rupees note 3.jpg**(image/jpeg) - 1137949 bytes, last modified: 5/31/2021 - 100% done
- **20 rupees note 4.jpg**(image/jpeg) - 1431848 bytes, last modified: 5/31/2021 - 100% done
- **20 rupees note 5.jpg**(image/jpeg) - 99207 bytes, last modified: 5/31/2021 - 100% done
- **10note.jpg**(image/jpeg) - 902868 bytes, last modified: 5/31/2021 - 100% done
- **NPL-1000-Front.png**(image/png) - 226447 bytes, last modified: 5/31/2021 - 100% done
- **50 rupees note 3.jpg**(image/jpeg) - 119453 bytes, last modified: 5/31/2021 - 100% done
- **20note.jpg**(image/jpeg) - 83816 bytes, last modified: 5/31/2021 - 100% done
- **50 rupees note 4.jpg**(image/jpeg) - 426257 bytes, last modified: 5/30/2021 - 100% done
- **50 rupees note 5.jpg**(image/jpeg) - 713281 bytes, last modified: 5/30/2021 - 100% done
- **50 rupees note 6.jpg**(image/jpeg) - 581544 bytes, last modified: 5/30/2021 - 100% done
- **50 rupees note 7.jpg**(image/jpeg) - 1412210 bytes, last modified: 5/30/2021 - 100% done

```
Saving 10 rupees note 1.jpg to 10 rupees note 1 (1).jpg
Saving 10 rupees note 2.jpg to 10 rupees note 2.jpg
Saving 10 rupees note 3.jpg to 10 rupees note 3.jpg
Saving 10 rupees note 4.jpg to 10 rupees note 4.jpg
Saving 100 rupees note 1.jpg to 100 rupees note 1.jpg
Saving 100 rupees note 2.jpg to 100 rupees note 2.jpg
Saving 100 rupees note 3.jpg to 100 rupees note 3.jpg
Saving 100 rupees note 4.jpg to 100 rupees note 4.jpg
Saving 100 rupees note 5.jpg to 100 rupees note 5.jpg
Saving 500 rupees note 1.jpg to 500 rupees note 1.jpg
Saving 500 rupees note 2.jpg to 500 rupees note 2.jpg
Saving 500 rupees note 3.jpg to 500 rupees note 3.jpg
Saving 1000 rupees note 1.jpg to 1000 rupees note 1.jpg
Saving 1000 rupees note 2.jpeg to 1000 rupees note 2.jpeg
```

```
+ Code + Text                                                    RAM Disk
Saving 1000 rupees note 1.jpg to 1000 rupees note 1.jpg
Saving 1000 rupees note 2.jpeg to 1000 rupees note 2.jpeg
Saving 50 rupees note 1.jpg to 50 rupees note 1.jpg
Saving 50 rupees note 2.jpg to 50 rupees note 2.jpg
Saving 20 rupees note 1.jpg to 20 rupees note 1.jpg
Saving 20 RS. REP - Copy (2).jpg to 20 RS. REP - Copy (2).jpg
Saving 20 rupees note 2.jpg to 20 rupees note 2.jpg
Saving 20 rupees note 3.jpg to 20 rupees note 3.jpg
Saving 20 rupees note 4.jpg to 20 rupees note 4.jpg
Saving 20 rupees note 5.jpg to 20 rupees note 5.jpg
Saving 10note.jpg to 10note.jpg
Saving NPL-1000-Front.png to NPL-1000-Front.png
Saving 50 rupees note 3.jpg to 50 rupees note 3.jpg
Saving 20note.jpg to 20note.jpg
Saving 50 rupees note 4.jpg to 50 rupees note 4.jpg
Saving 50 rupees note 5.jpg to 50 rupees note 5.jpg
Saving 50 rupees note 6.jpg to 50 rupees note 6.jpg
Saving 50 rupees note 7.jpg to 50 rupees note 7.jpg
[[0. 0. 0. 1. 0. 0.]]
10 rupees note 1.jpg
20
[[0. 0. 0. 1. 0. 0.]]
10 rupees note 2.jpg
20
[[1. 0. 0. 0. 0. 0.]]
10 rupees note 3.jpg
10
[[1. 0. 0. 0. 0. 0.]]
10 rupees note 4.jpg
10
[[0. 1. 0. 0. 0. 0.]]
100 rupees note 1.jpg
100
[[0. 1. 0. 0. 0. 0.]]
100 rupees note 2.jpg
100
[[0. 0. 0. 1. 0. 0.]]
100 rupees note 3.jpg
20
[[0. 1. 0. 0. 0. 0.]]
100 rupees note 4.jpg
100
[[0. 1. 0. 0. 0. 0.]]
100 rupees note 5.jpg
100
[[0. 0. 0. 0. 0. 1.]]
```





Source Code Link:

https://github.com/sushangautam/Nepali-Currency-Classifier--Deep-Learning/blob/main/Nepali_Currency_Classifier.ipynb

6. CONCLUSION AND FUTURE SCOPE

Currency Classifier is one of the most important application of Paper Recognition. Many studies and research are going on Currency Classifier technique. Eventhough it is being made all over the world, there are only a handful of people engaged in Nepali Currency Classifier field. Many paper have been found implementing Neural Network, but barely one or two people have tried implementing it in Nepali currency. Other techniques like Radial Basis Function Network, correlation between the images are also being implemented.

Transfer learning mechanisms are not used. Rather sequentially neural network is being built. In this project, we have implemented Neural Network for six different bills with appropriate layer such that the overall performance of the system increases. Three findings were computed, two of them by varying number of epochs 10 and 20 respectively. ADAM optimizer is being used as a replacement optimization technique for Stochastic Gradent Descent for training the neural network model. Categorical crossentropy is used as loss function. When the epoch size was 10, the training took place 1.5 to 2 hours whereas when varying the epoch size to 20 the time taken was almost 20 hours. Since none of our team has advance GPU inbuilt we trained the model in google collab, so it was time consuming. Nonetheless, this is a basic currency classifier model, there's a possibility of extending this work and implement much stronger neural network model. Flaws of the project would be not feeding the models with more varing epoch size to get better accuracy. The physical state of the currency is not optimum, thus, it will be another challenge.

Furthermore, the model can be implemented using different memory cells, Gated Recurrent Unit (GRU) or Generative Adversarial Network (GAN) in long run. This will help in optimizing the time constriant. The model can be deployed in a website also in an android application for a larger population to use in no time.

7. REFERENCES

[1] Frosini, A., Gori, M., & Priami, P. (1996). A neural network-based model for paper currency recognition and verification. IEEE transactions on neural networks, 7(6), 1482-1490.

[2] Zhang, Q., Yan, W. Q., & Kankanhalli, M. (2019). Overview of currency recognition using deep learning. Journal of Banking and Financial Technology, 3(1), 59-69.

[3] Mirza, R., & Nanda, V. (2012). Paper currency verification system based on characteristic extraction using image processing. International Journal of Engineering and Advanced Technology (IJEAT), 1(3), 68-71.

[4] Kong, F. H., Ma, J. Q., & Liu, J. F. (2006, August). Paper currency recognition using Gaussian mixture models based on structural risk minimization. In 2006 International Conference on Machine Learning and Cybernetics (pp. 3213-3217). IEEE.

[5] Alekhya, D., Prabha, G. D. S., & Rao, G. V. D. (2014). Fake currency detection using image processing and other standard methods. International Journal of Research in Computer and Communication Technology, 3(1), 128-131.

[6] Atchaya, S., Harini, K., Kaviarasi, G., & Swathi, B. (2016). Fake Currency Detection Using Image Processing. International Journal of Trend in Research and Development, special issue, 72-73.

[7] Bhurke, C., Sirdeshmukh, M., & Kanitkar, M. S. (2015). Currency recognition using image processing. Int. J. Innov. Res. Comput. Commun. Eng, 3, 4418-4422.

[8] Althafiri, E., Sarfraz, M., & Alfarras, M. (2012). Bahraini paper currency recognition. Journal of Advanced Computer Science and Technology Research, 2(2), 104-115.

[9] Debnath, K. K., Ahmed, S. U., Shahjahan, M., & Murase, K. (2010). A paper currency recognition system using negatively correlated neural network ensemble. Journal of Multimedia, 5(6), 560.

[10] Yadav, B. P., Patil, C. S., Karhe, R. R., & Patil, P. H. (2014). An automatic recognition of fake Indian paper currency note using MATLAB. Int. J. Eng. Sci. Innov. Technol, 3, 560-566.

[11] Ahmadi, A., Omatu, S., & Yoshioka, M. (2002, August). Implementing a reliable neuro-classifier for paper currency using PCA algorithm. In Proceedings of the 41st SICE Annual Conference. SICE 2002. (Vol. 4, pp. 2466-2468). IEEE.

[12] Abreu, G., Neves, R., & Horta, N. (2018). Currency exchange prediction using machine learning, genetic algorithms and technical analysis. arXiv preprint arXiv:1805.11232.

[13] Baasher, A. A., & Fakhr, M. W. (2011, October). Forex trend classification using machine learning techniques. In Proceedings of the 11th WSEAS international conference on Applied computer science (Vol. 1, No. 1, pp. 41-47). World Scientific and Engineering Academy and Society (WSEAS).

[14] Takeda, F., & Omatu, S. (1995). High speed paper currency recognition by neural networks. IEEE Transactions on Neural Networks, 6(1), 73-77.

[15] Alnowaini, G., Alabsi, A., & Ali, H. (2019, December). Yemeni paper currency detection system. In 2019 First International Conference of Intelligent Computing and Engineering (ICOICE) (pp. 1-7). IEEE.