INTELLIGENT CURRENCY IDENTIFICATION SYSTEM

Krishnavamsi Sanisetty

Table of Contents

1) Introduction

Evolutions in Machine learning and Artificial intelligence has evolved the world bringing up automation and intelligent systems the trending hot topic in today's time. Automating the task of handling currencies in an automated fashion has always been an arena which has gained attraction from the researchers. Researching about discrepancies and foot holes in the arena of automated currency handling, we thought of developing a system that will enable automatic classification of paper currency image based on the its respective country. Diversities and variations in paper currencies around the world, their different sizes, colour and texture motivated us to work on our own collected dataset.

Globally, there are around 180 currencies. Identification and tracking each currency manually is tough and not feasible for humans. Advent of Google lens has made it possible to scan and detect objects in one click. Blending Machine learning and Image processing can prove beneficial for development of efficient Cash Deposit Machines, Money transfer Machines around the globe.

With this motivation we worked on data modelling system which classifies image into seven different categories i.e countries. This system can identify currencies like US dollar, Euro, Indian Rupees, Chinese Yuan and UAE Dirham. To increase the scope of this project we have included another category which classifies image into invalid category. The dataset for invalid category mainly include old currency bank notes. On 9th November 2016 Demonetisation happened in India which enabled citizens to exchange old 1000 and 500 Indian rupee notes with the new one. Addressing this situation we thought of including this category just to classify whether the currency image is invalid 1000 or 500 Indian rupee note.

2) Business Value

Power of Machine Learning Algorithms and Intelligent Image processing techniques can be used to leverage power of data in order to develop several real world business solutions. Currency identification systems primarily target Banking and Financial Sectors. Globally, In approximately 195 countries there are 180 possible currencies. Development of Intelligent currency image classification system can prove beneficial for efficient bank transactions.

Automated currency detection system when integrated with cash deposit machine can suffice deployment of autonomous banking systems. With this system Cash Deposit Machines can detect the currency based on the country and the denomination and hence it can handle currency transfer transaction in native currency in an automated fashion without any human intervention. Such systems when deployed at airports will allow passengers to transfer currencies into their bank account irrespective of any location, time and native currency. Advent of Technology has created world into a global market diminishing boundaries of business. Around the globe multiple financial transactions happen daily which can be automated with enhanced functionality and updated currencies.

Another possible use case is Money exchange systems. Consider an example of a person at Mumbai airport who want to exchange Indian currency into US dollar. An automated money transfer machine can enable him to exchange his currency notes into US dollar considering the exchange rate at that time stamp. Lastly, currency detection system can help people with special abilities to detect their currencies, convert them or perform any financial banking transaction efficiently.

In a nutshell, Currency Identification can prove a significant factor in improving automation in almost every business sector. With world leading closer to automation and digitisation we believe, this could be the future. Companies such as IBM, Hitachi are currently working in these field in order to enhance banking systems for the customers in order to enhance customer satisfaction and enhance banking transactions globally.

3) Dataset Source and Description

Our project's main idea is to Classify Currency images based on countries. Considering diverse currency patterns, variations in size, colour and denominations we thought of collecting our own dataset.

Our main aim while collecting dataset was.

- Focus on diversity where currencies with multiple patterns and sizes and color will be in the dataset.
- Focus on old currency notes, new updated currency notes and notes which has been categorized as invalid by the government.
- Collection of currencies which belonged to all possible denominations in a particular country. Major denominations of currencies from each country, primarily the 1, 10, 20, 50, 100, 1000 were incorporated In order to keep the dataset as consistent as possible.

To work on the problem statement effectively, we divided our project into levels. These helped us to address issues with the dataset such as diversity, feature engineering and over fitting.

i) Level 1.

For the first level, we wanted our model to successfully classify the currency in two categories, i.e., US Dollar and Non-US currency. For this we had images collected in two categories and then used them for modelling.

In level one we had almost 600 images in train which included over 250 images in each category and 200 images in test.

ii) Level 2.

In level 2 we scaled our dataset to over total 2700 images for Train, Test and Validate. The table above shows the amount of images in each category in test, train and validate subparts.

Country (Category)	Train	Validate	Train
Chinese_Yuan	245	49	83
Euro	200	55	82
Indian_Rupee	196	61	93
UK_Pound	230	50	82

United_Arab_Emirates_Dilram	276	54	89
US_Dollar	256	53	95
Old Invalid Currencies	265	48	86

To achieve the diversity, we thought of different ways of collecting the data for our project. Collection of data from different sources such as

- Downloading Google Images from internet,
- Manually taking pictures, taking pictures from many angles,
- Using plug-in to download multiple images,
- Having redundant images of currencies were part of the process.

The input to the model is given as image itself and therefore it was imperative that we keep the resolution of the images consistent and visible. The images are converted to gray scale as data information is important for the network and not the color information.

Sample images from the dataset.

4) Data Exploration and Pre-processing.

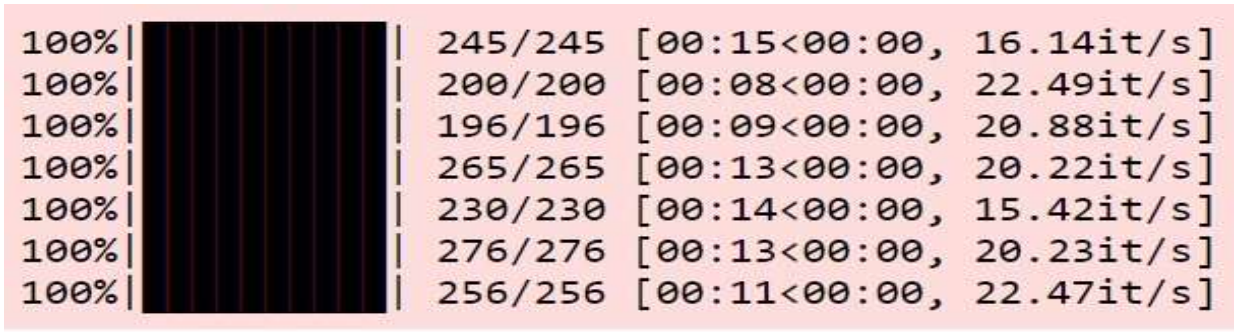
1. Data Exploration.

Our dataset had 1666 images in training subpart, 357 in validation subpart and 609 in test subpart.

```
print(type(training_data))
print(len(training_data))
print(type(testing_data))
print(len(testing_data))
```

```
<class 'list'>
1666
<class 'list'>
609
```

The above figure shows number of images in training and testing subparts.



The above image shows the time taken to import all training images in each category, amount of images in each category and integer value assigned to each category.

The above image shows the time taken to import all testing images in each category, amount of images in each category and integer value assigned to each category.

2. Data Pre-processing.

We divided our project into two categories.

1) Machine Learning model.
2) Deep learning model.

We followed following steps for data pre-processing.

1) Machine Learning model.

    1) Imported each image and stored its pixel value in the arrays.
    2) Resized each image in the resolution of 148 X 148 in order to make all images consistent.
    3) Converted all images to grayscale in order to reduce time during training and increase consistency in the dataset.

2) Deep Learning model

```
#Configuring image generator in order to augument images to enhance dataset.
train_datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=True, rotation_range=20)
valid_datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=True, rotation_range=20)
test_datagen = ImageDataGenerator(rescale=1./255)
```

    1) Imported each image and stored its pixel value in the arrays.
    2) Resized each image in the resolution of 148 X 148 in order to make all images consistent.
    3) Converted all images to grayscale in order to reduce time during training and increase consistency in the dataset.
    4) Augmented each image by rescaling it, flipping it horizontally and rotating it in range of 20. The main purpose of doing this is to add diversity to the dataset and to scale dataset in order to avoid over fitting in the model.

```
#importing images from training folder
train_generator = train_datagen.flow_from_directory(
    directory="F:/dsp_project_all_files/train",
    target_size=(148, 148),
    color_mode="grayscale",
    batch_size=128,
    class_mode="categorical",
    shuffle=True,
    seed=42
)
```

```
Found 1666 images belonging to 7 classes.
```

```
In [19]: #displying images just to check the resolution
some_digit_train = x_train_basic[54]
some_digit_train = some_digit_train.reshape(148,148)
plt.imshow(some_digit_train,cmap="gray")
plt.show()
```

The above image shows the pre-processed image from the dataset.

5) Image Classification Model.

Part 1. Traditional Machine Learning model.

In order to compare performance of traditional machine model and deep learning model we divided our project in two parts where part 1 focused on building classification model using SGD Classifier, Random Forest Classifier and Ensemble Learning algorithm which combined SGD classifier and Random Forest Classifier models.

1) SGD Classifier.

Stochastic Gradient Descent (SGD) is an optimization algorithm which can be used to find the values of parameters of functions that minimize a cost function. In other words, it is used for discriminative learning of linear classifiers under convex loss functions such as SVM and Logistic regression. It has been successfully applied to large-scale datasets because the update to the coefficients is performed for each training instance, rather than at the end of instances.

```
#Using SGDClassifier to train data
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier(random_state=50,learning_rate="adaptive",eta0=0.01)
type(sgd)
```

```
sklearn.linear_model._stochastic_gradient.SGDClassifier
```

We trained our model using SGD Classifier with the following parameters.

a. random_state = 50 - set seed of pseudo random number generated while shuffling the data .

b. learning_rate = adaptive. – define learning rate

c. eta0 = 0.01 – Initial learning rate value

2) Random Forest Classifier.

Random Forest Classifier is an ensemble tree learning algorithm. This algorithm is a set of decision trees generated by random shuffling of the data. It aggregates the votes from different decision trees to decide final class of the test data.

```
#training model using Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
forest_clf = RandomForestClassifier(random_state=42)
y_probas_forest = cross_val_predict(forest_clf, x_train_basic, y_train, cv=3,
 method="predict_proba")

y_scores_forest = y_probas_forest[:, 1]
```

In our case Random forest Classifier with default parameters gave best results hence we kept the parameter value default with random seed set to 42.

3) Ensemble learning Algorithm.

Ensemble learning combines results of several models in order to gain large predictive accuracy. This approach allows the production of better predictive performance compared to a single model. Basic idea is to learn a set of classifiers (experts) and to allow them to vote.

```
#Building model using ensemble learning algorithm which combines results of SGD Classifier and Random Forest Classifier.
from sklearn.ensemble import VotingClassifier
voting_clf = VotingClassifier(
    estimators=[('rf', forest_clf), ('sgd', sgd)],
    voting='hard')
voting_clf.fit(x_train_basic, y_train)
```

Part 2. Deep Learning model.

Convolutional Neural Network.

Part 2 of our project focused on building a deep learning model using Convolutional Neural Network. CNN being one of the most powerful deep learning algorithm input image data and maps them to a feature vector. Convolutional Neural networks allow system to detect features in an image, in other words, Convnets are used to recognize images by transforming the original image through layers to a class scores. CNN was inspired by the visual cortex.

The figure below shows our model architecture with parameters.

```
In [100]: #configuring CNN architecture and compiling model by setting early stopping
          from keras.callbacks import EarlyStopping

          model = Sequential()

          model.add(Conv2D(32, (3, 3), padding='same', strides=(1, 1), input_shape=(148,148,1)))
          model.add(Activation('relu'))
          model.add(MaxPooling2D(pool_size=(2, 2)))

          model.add(Conv2D(64, (3, 3)))
          model.add(Activation('relu'))
          model.add(MaxPooling2D(pool_size=(2, 2)))
          model.add(Dropout(rate=0.2))

          model.add(Conv2D(128, (3, 3)))
          model.add(Activation('relu'))
          model.add(MaxPooling2D(pool_size=(2, 2)))
          model.add(Dropout(rate=0.25))

          model.add(Flatten())  #this converts our 3D feature maps to 1D feature vectors

          model.add(Dense(256))

          model.add(Dense(7))
          model.add(Activation('softmax'))

          model.summary()

          opt = keras.optimizers.Adam(learning_rate=0.0001, beta_1=0.9, beta_2=0.999, amsgrad=False)

          model.compile(loss='categorical_crossentropy',
                        optimizer=opt,
                        metrics=['accuracy'])    #accuracy = loss and optimizer model algorithm
```

Input Dimensions: We kept our input shape to **(148,148,1)** because our image is of resolution 148 X 148 and 1 in the shape denotes that all images are grayscale.

Padding: We kept **padding = same** so that the output has the same length as the original input.

Kernel_size = (3,3) We kept kernel size that is filter size for our model to (3,3). This 3 X 3 matrix will be used for feature detection**.**

Strides: (1,1) – We kept Strides to 1 so that it produces smaller output volume. With strides = 1 the filter will shift by amount of 2 pixels as it traces the input**.**

Max_pooling: This layer is primarily responsible for dimensionality reduction. We kept **pool_size = (2,2)** in the layers as shown in the above architecture.

Activation Function: ReLu activation function is used in the architecture except for last layer which has activation function **Softmax** because it perform multi class classification. ReLu activation function is selected because it is computationally efficient and this activation function does not change the size of the input and does not have any parameters.

Dropout Layer: In the architecture drop out layer has **drop-out – 0.25 and 0.20.** The purpose of this layer is to address overfiting issue in the model. The drop out layer will randomly ignore 25% of the nodes while training the model. This method can be used for regularization as well.

Loss= Categorical_crossentropy – This loss will compute probability over each class for each image.

Optimizer: Adam – Adam is an adaptive learning algorithm which is a combination of RMSprop and SGD with momemtum. Inour architecture we parameterized optimizer keeping **learning_rate at 0.0001, amsgrad= False, beta_1= 0.9, beta_2=0.999.**

Metrics: Accuracy - The evaluation metric for the model is accuracy.

6) Model Evaluation and Results.

Part 1. Traditional Machine Learning Model.

1) SGD Classifier.

```
In [29]: #applying 3 fold cross validation.
         from sklearn.model_selection import cross_val_score
         cross_val_score(sgd, x_train_basic, y_train, cv=3, scoring="accuracy")

Out[29]: array([0.80395683, 0.81621622, 0.7981982 ])
```

The above figure shows the accuracy results of 3 fold cross validation which is applied on the model trained using SGD Classifier.

3 fold cross validation was used with an objective to address over fitting and to make our model more robust. The model had maximum accuracy of 81.62%.

```
In [39]: #Evaluating model using confusion matrix
         from sklearn.metrics import confusion_matrix
         confusion_matrix(y_train, y_train_pred)
```

```
Out[39]: array([[164,  34,  17,   0,  10,   7,  13],
                [ 10, 150,  15,   0,  10,   9,   6],
                [ 13,   7, 133,   4,   6,  15,  18],
                [  0,   0,   3, 262,   0,   0,   0],
                [ 16,  18,   8,   1, 176,   4,   5],
                [  4,   7,  15,   0,   5, 241,   4],
                [  7,   5,   8,   1,   6,  12, 217]], dtype=int64)
```

The above figure shows the confusion matrix after training the model on train data. Confusion matrix helped us to asses our model's performance effectively.

```
In [40]: #Evaluating model using precision, recall and f1_score
         from sklearn.metrics import precision_score, recall_score
         from sklearn.metrics import f1_score
         print("Precision",precision_score(y_train, y_train_pred, average="macro"))
         print("Recall",recall_score(y_train, y_train_pred, average="macro"))
         print("F1_Score",f1_score(y_train, y_train_pred, average="macro"))
```

```
Precision 0.7970333590149039
Recall 0.7970589870449991
F1_Score 0.7961053669197395
```

The above figure shows Precision, Recall, and F1 score values for the model which is evaluated using training data.

Evaluating model using Test data.

```
In [42]: #evauating model based on testing data
         from sklearn.metrics import confusion_matrix
         confusion_matrix(y_test, y_test_pred)
```

```
Out[42]: array([[63,  2,  3,  0,  7,  4,  4],
                [ 2, 66,  4,  0,  5,  2,  2],
                [ 2, 12, 58,  4,  6,  8,  3],
                [ 0,  0, 11, 69,  0,  0,  6],
                [ 5,  8,  8,  0, 50,  6,  5],
                [ 2,  1,  2,  1,  2, 79,  2],
                [12,  4, 10,  3,  5,  4, 57]], dtype=int64)
```

```
In [44]: #Evaluating model using precision, recall and f1_score
         from sklearn.metrics import precision_score, recall_score
         from sklearn.metrics import f1_score
         print("Precision",precision_score(y_test, y_test_pred, average="macro"))
         print("Recall",recall_score(y_test, y_test_pred, average="macro"))
         print("F1_Score",f1_score(y_test, y_test_pred, average="macro"))
```

```
Precision 0.7282402952758371
Recall 0.7281755716808785
F1_Score 0.7256567080113532
```

The above figure shows the model's Precision, Recall and F1_score based on testing data. Looking at the results we think our model performed considerably good without over fitting.

Ensemble Learning algorithm results.

```
In [47]: #Evaluting accuracy of all three models SGD Classifier, Random Forest Classifier and Ensemble Learning Algorithm
         from sklearn.metrics import accuracy_score

         for clf in (forest_clf, sgd, voting_clf):
             clf.fit(x_train_basic, y_train)
             y_pred = clf.predict(x_test_basic)
             print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

```
RandomForestClassifier 0.7684729064039408
SGDClassifier 0.8045977011494253
VotingClassifier 0.7914614121510674
```

The above figure shows evaluation results of all three models that is SGD Classifier, Random Forest Classifier and Ensemble Learning algorithm. In our case SGD Classifier worked best which had an accuracy of approximately 80.45%

Part 2. Evaluating CNN Deep Learning Model.

```
In [101]: ### from keras.callbacks import EarlyStopping

earlystop = EarlyStopping(monitor='val_loss', patience=5, verbose=1, mode='auto')

history = model.fit_generator(
        train_generator,
        steps_per_epoch=100,
        epochs=10,
        validation_data=valid_generator,
            callbacks=[earlystop])
```

```
Epoch 1/10
100/100 [==============================] - 457s 5s/step - loss: 1.8385 - accuracy: 0.2924 - val_loss: 1.4361 - val_accuracy: 0.
5730
Epoch 2/10
100/100 [==============================] - 502s 5s/step - loss: 1.1116 - accuracy: 0.6143 - val_loss: 0.8831 - val_accuracy: 0.
7189
Epoch 3/10
100/100 [==============================] - 521s 5s/step - loss: 0.7463 - accuracy: 0.7431 - val_loss: 0.6593 - val_accuracy: 0.
7622
Epoch 4/10
100/100 [==============================] - 508s 5s/step - loss: 0.5804 - accuracy: 0.8085 - val_loss: 0.4762 - val_accuracy: 0.
8216
Epoch 5/10
100/100 [==============================] - 513s 5s/step - loss: 0.4455 - accuracy: 0.8571 - val_loss: 0.4479 - val_accuracy: 0.
8595
Epoch 6/10
100/100 [==============================] - 515s 5s/step - loss: 0.4911 - accuracy: 0.8554 - val_loss: 0.4147 - val_accuracy: 0.
8541
Epoch 7/10
100/100 [==============================] - 519s 5s/step - loss: 0.3055 - accuracy: 0.9116 - val_loss: 0.3236 - val_accuracy: 0.
8919
Epoch 8/10
100/100 [==============================] - 513s 5s/step - loss: 0.2812 - accuracy: 0.9175 - val_loss: 0.5248 - val_accuracy: 0.
8649
Epoch 9/10
100/100 [==============================] - 522s 5s/step - loss: 0.2049 - accuracy: 0.9479 - val_loss: 0.4795 - val_accuracy: 0.
8595
Epoch 10/10
100/100 [==============================] - 514s 5s/step - loss: 0.2362 - accuracy: 0.9389 - val_loss: 0.1889 - val_accuracy: 0.
9027
```

The above figure shows the results of the convolutional neural network which has been set to train the data for 10 epoch. Looking at the final epoch results we got

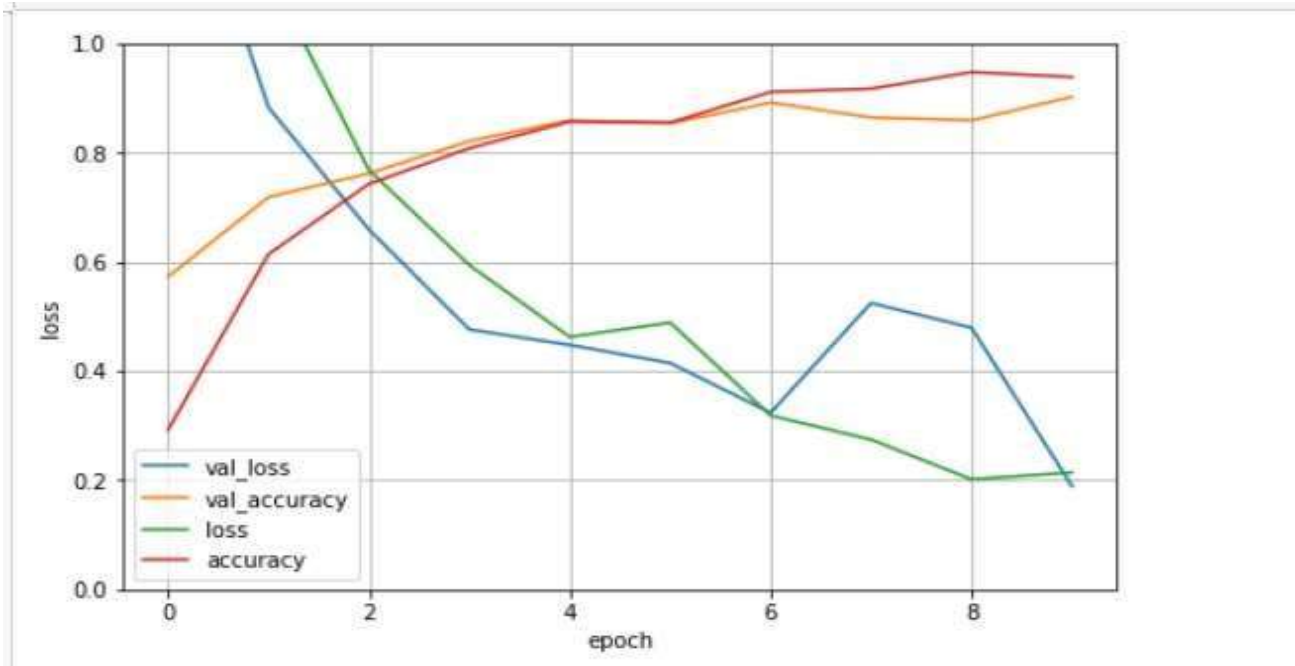
Accuracy on Training Data – 93.89%

Accuracy on Validation Data - 90.27%

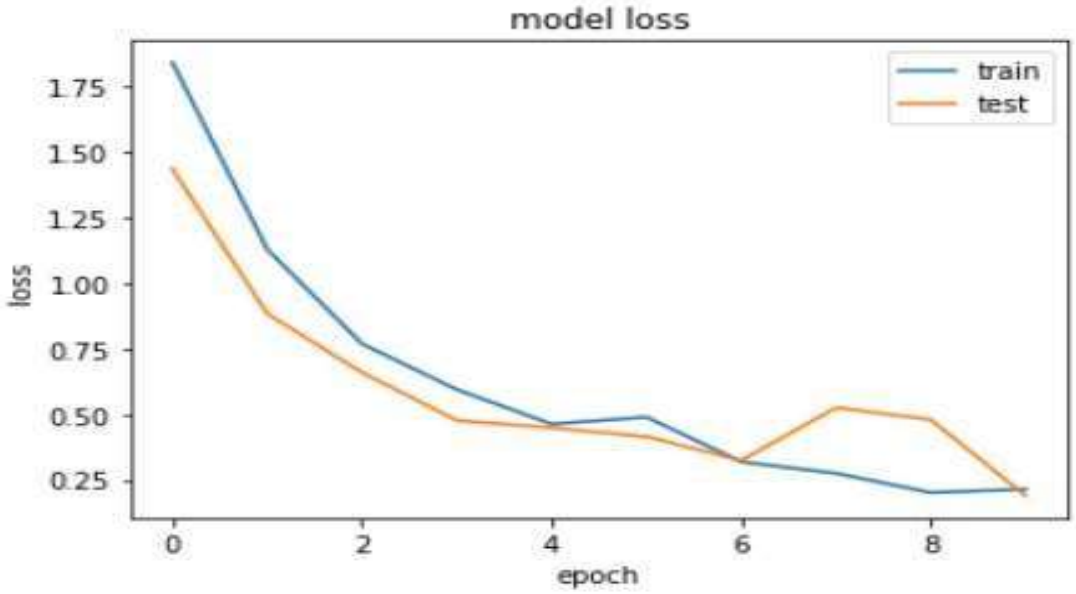
```
test_loss, test_acc = model.evaluate_generator(test_generator)

print(test_acc)
```

```
0.8128078579902649
```

Accuracy on Test Data – 81.28%



The above figure plots training accuracy, validation accuracy, training loss and validation loss across each epoch. From the above figure we can see that training accuracy and validation accuracy increases through the epochs. There is not much difference between training accuracy and validation accuracy which shows that the model and not Overfitting.



The above figure plots training loss and validation loss across each epoch. From the above figure we can see that training loss and validation loss decreases through each epochs. There is not much difference between training loss and validation loss which shows that the model is training effectively.

7) Future Scope:

The main purpose of this project is to add business value in Financial and Banking sector through autonomous currency detection. This project will help to automate and revolutionize the way a cash transaction is handled, across the world.

Our future work includes:

- **Scaling:** There are 195 countries in the world with total of 180 different currencies recognized by the UN. Scaling of data and including the currencies and denominations from around the world is our top priority. For 'single' currency across the world, it is important to scale data right.

- **Trying out different models:** Exploring models like ResNet-101, R-CNN models to increase the model efficiency and trying out faster RCNN's. In addition to these we focus to combine power of CNN and RNN in order to build custom model.

- **Fake Currency Detection:** Recognizing a fake bill is one of the use cases that we feel excited about. Including fake currencies in our dataset to recognize the patterns when a 'fake' is encountered.

- **Conversion:** For autonomous currency handling transaction system it is important that along with detecting the 'country of origin' the money also gets converted with respect to the rate at that moment and the customer receives the desired currency.

- **Accuracy:** Dealing with glitches and trying to improve the existing model to perform better and handle any situation with more accuracy. Money is imperative part of an individual's life, and handling money with utmost care will remain a high priority task.

- **Deploying:** Deployment of model as a web service to see how the model will perform with real-world and dynamic data in initial stage using technologies like flask, etc.

8) Conclusion:

Automation helps in increasing efficiency and productivity of any existing institution. ATM's brought the much-needed revolution in how banking services ran, but it is time for how the money and particularly the cash-transactions are handles when there is foreign cash involved in the transaction. We through this project look to bring this idea of Currency recognition and further down road convert the money and thus provide a working system for the same through image processing and concepts of data science and deep learning.

To get image data we collected data from different sources including collecting data manually. We pre-processed data to make the data consistent for inputting the data to models. Currency recognition was split in different stages, with first being the binary type recognition of whether the currency is from either United States or other. Second stage was scaling the dataset and increase the categories to recognize, and finally, to model the scaled data to find the best model with high accuracy. Algorithms like SGD classifier, Random Forest, Ensemble and CNN were explored and concepts such as unconventional data pre-processing, epochs, hidden layers, activation functions, regularization were included. We trained our model and then selected the best one, we got satisfactory results in our testing dataset. In future, we look forward to scale our project to satisfy other use-cases and business needs.

In a nutshell, this project enabled us to gain hands on experience in utilizing power of python data science libraries in leveraging power of currency image data in order to extract significant information which can help Business domain to improve efficiency and enhance automation in regular business operations.

9) References

1) https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

2) https://towardsdatascience.com/boost-your-cnn-image-classifier-performance-withprogressive-resizing-in-keras-a7d96da06e20

3) https://www.researchgate.net/publication/328960034_Effects_of_Varying_Resolution_on_Performance_of_CNN_based_Image_Classification_An_Experimental_Study

4) https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization6be9a291375c

5) https://blog.paperspace.com/deploying-deep-learning-models-flask-web-python/

6) https://ieeexplore.ieee.org/document/8284300

7) https://www.youtube.com/watch?v=j-3vuBynnOE

8) https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deeplearning-99760835f148

9) https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-onthe-internet-fbb8b1ad5df8

10) https://becominghuman.ai/what-exactly-does-cnn-see-4d436d8e6e52